Usage patterns for user-centric service composition

UNIVERSITY OF TWENTE.

Edwin Vlieg

Software Engineering
Faculty of Electrical Engineering, Mathematics & Computer Science
University of Twente

A thesis submitted for the degree of

MSc Computer Science

November 2010

Supervisors
Luís Ferreira Pires
Eduardo Gonçalves da Silva
Betsy van Dijk

Abstract

Service composition is a technique applied in software systems and has the goal to compose application services in order to achieve value-added functionality. With the Internet increasingly woven into our daily lives, access to user friendly services becomes increasingly important. Services composition provides a way to fulfill complex goals of end-users by combining the power of several services. Mashups are an example of a successful service composition approach, but in this approach a professional developer is responsible for developing the composition.

User-centric service composition aims at providing a user-aware software platform for creating compositions of services in order to fulfill the goals of the user. This enables less skilled users to create compositions that achieve value-added functionality.

This thesis describes the engineering of a prototype for user-centric service composition in the e-commerce and entertainment domain based on the DynamiCoS framework. The e-commerce and entertainment domain contain realistic scenarios for service composition and have different types of users, namely users that lack technical knowledge. An architecture for the prototype and interaction patterns with the DynamiCoS framework are introduced. Furthermore the prototype is evaluated with end-users.

Acknowledgements

I would like to thank the people that supported me during the writing of my thesis.

First of all, I would like to thank Eduardo Gonçalves da Silva and Luís Ferreira Pires for their supervision. Doing research was sometimes hard for me as an entrepreneur, but you managed to keep me on track. Eduardo, I really enjoyed our discussions and work sessions. Luís, thanks for the feedback on my writing and monitoring my progress. Furthermore I would like to thank Betsy van Dijk for being my second supervisor. Although our meetings were scarce, they always gave me new insights into my work from your HMI point of view.

My colleagues and friends always supported me during my graduation. My business partners at MoneyBird gave me the opportunity to experience the great adventure of being part of a startup, but still allowed me to get my degree.

Finally, I would like to thank my family. You have always supported me in completing my study, but also gave me the freedom to follow my passions in music and entrepreneurship. Thanks!

Edwin Vlieg
November 2010, Enschede

Contents

1

Introduction

This chapter presents an introduction and the motivation for our research on usage patterns for user-centric service composition. Furthermore, it presents the approach taken in this research and the structure of this thesis. This chapter is structured as follows: section 1.1 contains the motivation for our research; section 1.2 states our objectives; section 1.3 presents the project research questions; section 1.4 presents the structure of the thesis.

1.1 Motivation

Internet services are increasingly woven into the daily lives of many people. The range of services on the Internet is huge and creates many new challenges for software engineers. Internet-based application services are increasingly being used to support everyday tasks. This trend is not limited to professional users, but also more and more consumers are using such services. It is a challenge for software engineers to create user-friendly services and even more to create services personalized to a wide range of possible target users.

An approach to simplify access to Internet services (web services) for the end-user is to reduce the complexity of the user interface. Often this means to focus on solving a single problem at a time for the end-user. This contrasts with traditional software, which tends to resolve all requirements of menu end-users in a single software system, by offering complex user interfaces with too many options. Many companies are very successful in solving only one particular problem through a web application, including

Google (with GMail [15] and Google Apps [16]), 37signals (with Highrise, Basecamp and Backpack [1]) and FreshBooks [14].

In the domain of personal digital lifestyle, services are also becoming more and more important. An example is e-commerce, where more consumers are buying their goods online, using different services in the purchasing process. Examples of services are Kelkoo to compare prices, CNET to review products and Google Maps to find the nearest store. Entertainment and e-health are yet two other domains concerned with personal digital lifestyle that are taking advantage of Internet services.

As a result from focussing on a single problem at a time, one single service often does not fulfill all requirements of the end-user. A composition of various services is often a suitable solution to solve a complex problem of a user. For example, a company that uses a billing service retrieves client information from its customer relationship service and the project's hours from the project management service. The consumer who wants to buy a new phone can make an assessment based on the price on Kelkoo, check reviews on CNET and search for the nearest store via Google Maps.

Mashups

Service-Oriented Architectures [13] address the issues around the creation, use and sharing of services. A popular approach for service composition with focus on the end-user is to build mashups [8]. In a mashup, it is often the software engineer who decides which requirements of the end-user he is going to fulfill. Based on these requirements, he creates a composition of services to be used by end-users. Often the requirements of an end-user are not completely fulfilled. Furthermore, the number of possible combinations of services is "endless" so that it creates a so-called long tail [3]. When service composition is able to address the requirements in the long tail, a larger number of users can be satisfied.

An often used example of a mashup is housingmaps.com [30]. This website is a combination of Google Maps and Craigslist. Google Maps enables the user to zoom in on a particular region, and the mashup automatically shows advertisements of houses for rent in the visible region. This mashup is useful for end-users who wish to find a flat to rent in a particular city. The usefulness of this solution is quickly limited once the user adds additional requirements, such as finding a flat in the range of at most 5 metro stops from the office.

Enterprise mashups overcome the major disadvantage of the above described mashups by empowering business users to combine services [19]. By leveraging peer production [4] and collaboration, an enterprise mashup is more user-centric than the mashups described above. Enterprise mashups primarily focus on business end-users, while service composition can be applied in domains with very different end-users, so a solution to service composition that also takes other types of end-users into account is needed.

User-centric service composition

Enterprise mashups are still very dedicated to business experts. Simpler approaches are needed to make service composition better comply with the requirements of end-users in other domains. Firstly, the approach should enable an end-user to communicate his requirements to the system. For example, research on natural language processing and knowledge representation offer the possibility to convert natural language into machine readable requirements. Secondly, it is important to allow the user to properly interact with the system. Services can be applied in different domains, focussing on different kinds of end-users. Therefore it is plausible to take into account the less skilled users.

There are different types of users. In [10] two factors are identified for defining the skills of the end-user: the user's technical knowledge and domain knowledge.

For a proper composition of services, a certain degree of *technical knowledge* is required. For example, a user with technical knowledge on service composition may be able to create a composition in the Business Process Execution Language (BPEL) [29]. On the other hand, a user with little technical knowledge requires a more intuitive and abstract interface in the process of service composition.

Domain knowledge is required to determine in which way service composition can be applied within the domain. Service knowledge is a more specific kind of domain knowledge concerning the knowledge on the services offered in the domain. A user with little knowledge about available services, needs more support in discovering and choosing the right services to fulfill his requirements, in contrast with a user with extensive knowledge of the domain and the services. The domain expert can easily determine the various services to be combined to fulfill a certain requirement.

In [10] four different types of users are distinguished: *layman*, *domain experts*, *technical experts* and *advanced users*. In this classification, the layman is a user without any technical or domain knowledge. The advanced user has both domain and technical

knowledge. The type of user influences the patterns of interaction between a user and the service composition system.

To address the problem of user-centric service composition, the DynamiCoS framework [9, 11] is being developed at the University of Twente. The goal of the framework is to satisfy the requirements of different types of users, enabling them to access and compose services in a user-centric manner. DynamiCoS aims at shielding its users from the complexity of the service composition process by guiding the user through the service composition process.

In order to enable user-centric service composition to its full extent, extended knowledge about the domains, types of users and user interaction patterns is required. Service composition software with simpler user interfaces allows users, regardless of their background, to compose services from large sets of available services. With this empowerment of the user, service composition and the provided services can get to the next level of integration into our digital lives.

1.2 Objectives

The main objective of this thesis is to evaluate a user-centric service composition approach. We take a pragmatic approach to define possible applications of user-centric service composition in existing realistic software scenarios in different domains. By applying user-centric service composition to different domains, we can identify usage patterns.

To achieve this objective, we have designed and developed a prototype to test the support for service composition and to perform usability tests with end-users. The prototype consists of a client application, which uses the DynamiCoS framework to manage the service composition process. The domain of the prototype has to contain realistic scenarios for service composition. Furthermore, the domain has to contain users lacking technical knowledge or domain knowledge in order to test the usability of the prototype for these types of users.

The extended knowledge about usage patterns obtained with the research should improve the flexibility of the DynamiCoS framework. More flexibility enables a broader range of application of the framework in different domains.

1.3 Research questions

The following questions have been posed in our research:

1. Identify and characterize possible application domains for user-centric service composition. Before we are able to create a prototype for user-centric service composition and perform tests with end-users, we need to identify suitable application domains for user-centric service composition. With a literature study we try to determine domains that are representative for the application of user-centric service composition.

2. What are the application scenarios? After identifying domains with realistic scenarios for service composition, we define a set of application scenarios to test the design procedure and usability of the client application. These scenarios have to reflect the different types of end-users and should represent user-centric service composition situations.

3. How should the architecture of the client application look like? The architecture of the client application will be based on a supporting framework providing service composition functionality. The goal of such a supporting framework is to provide a generic solution for service composition. The architecture of the client application should allow the client application to work with such a generic framework and still provide a domain specific user experience to the end-user.

4. How can the client application interact with the supporting framework? The architecture of the client application defines the interaction patterns between the client application and a supporting framework. The supporting framework has to support the interactions performed by the client application.

1.4 Structure

The structure of the thesis resembles the research questions we have answered during our research. The remainder of this thesis is structured as follows:

- **Chapter 2** gives an overview of possible application domains for user-centric service composition. This overview is used to define a realistic scenario for service composition and gain more insight in the usage patterns of end-users with the service composition supporting system;
- **Chapter 3** introduces details about the selected domains. The chapter starts with an overview of services and users in the domain. It concludes with scenarios which the client application should support;
- **Chapter 4** presents the architecture of the client application we have created to support the scenarios we described in chapter 3. It also presents the DynamiCoS framework;
- **Chapter 5** introduces the requirements the DynamiCoS framework should fulfill in order to enable the client application to support its users;
- **Chapter 6** contains the evaluation of the client application we have created. The client application has been tested by users in an user test. The evaluation is based on our experience of developing the client application and the results of the usability test;
- Finally, **chapter 7** concludes our research and suggests further work.

2

Application domains for user-centric service composition

Service composition can be applied in many different application domains. Our research focuses on domains where users with different background knowledge create and use service compositions, specially at runtime. In order to define a realistic scenario for service composition and gain more insight in the interaction patterns of users with the service composition supporting system, we start our research by giving an overview of possible application domains for user-centric service composition. Section 2.1 contains the criteria used to assess the domains; section 2.2 describes the application domains; section 2.3 gives our conclusions on the studied application domains.

2.1 Assessment of domains

We limit our overview to currently relevant domains on which information is publicly available and domains that contain valuable scenarios for the creation of an experimental prototype. Valuable scenarios contain different types of users with different domain and technical knowledge, allowing us to test an user-centric approach for service composition.

We will use four criteria for assessing the domains:

- The *types of users* in the domain: we describe their roles and knowledge in terms of domain knowledge and technical knowledge;

- The *types of services* in the domain and the possible compositions that can be made from these services;
- The *human-computer interaction* between the users of the domain and the service composition supporting system. The user devices used in the domain also influence how the users can interact with the service composition system;
- The *conditions of composition* in the domain. The conditions in which a composition is made can limit the usability of a service composition system.

The assessment is accompanied by an example of the application in the domain to illustrate the usability and interaction patterns.

2.1.1 Types of users

Systems to support service composition fulfill two main tasks: create new compositions of services and execute a composition of services. Given this, we can distinguish between two different user roles:

- A user with the **composer role** creates a composition of available services in the domain;
- A user with the **end-user role** uses service compositions made available by composers.

When a service composition is created, the objective is to satisfy a given set of requirements for a given end-user. In the context of our work we will have users in the role of composer and users in the end-user role. Any user can play both roles, we will refer to this user as the 'user' of the service composition supporting system.

Another important factor to describe and characterize users is their knowledge. In [10], three types of knowledge are identified: *application domain knowledge*, *service knowledge* and *technical knowledge*.

Application domain knowledge This knowledge about the domain is essential in order to define which requirements a service composition should fulfill. Domain knowledge consists of knowledge of the concepts and tasks in a domain. The knowledge can be gathered by learning, experience or advertisement. The users' knowledge about

the domain can improve when the user interacts with software and domain experts in the domain.

Service knowledge Domain knowledge is general to all activities in the domain and does not imply specific knowledge about the services in a domain. Service knowledge is the awareness and understanding the user has of the services available in the domain. Service knowledge is required to select a service for a composition.

Technical knowledge Besides application domain knowledge, a user creating a service composition needs to have some degree of technical knowledge. Service composition techniques are currently mainly used by professionals, leveraging techniques like WSDL (Web Service Description Language) [6] for service description and BPEL (Business Process Execution Language) [29] for service composition. However, service composition should not be limited to users with technical knowledge. Nowadays, the number of available services is increasing constantly and users may require the composition of the functionality of several services to realize their objectives. Since not every user is a professional developer, users without much technical knowledge require simpler, more abstract, supporting environments.

Based on the domain knowledge, service knowledge and technical knowledge, it is possible to create a users' classification, shown in table 2.1. The *layman* is a user with no domain or technical knowledge, meaning that he needs the most guidance in the service composition process. On the other hand, the *advanced user* has both domain and technical knowledge, meaning that he has knowledge and skills to create a service composition with almost no guidance.

The *domain expert* has knowledge about the domain and services, but lacks the technical knowledge to create the composition. Therefore, he requires some assistance to be able to create a service composition. The *technical expert* has technical knowledge to create a composition, but does not have domain or service knowledge. The users' classification shows a clear need for certain users to be guided during the composition process.

Table 2.1: Types of users defined in [10]

Type of End-user	Application Domain Knowledge	Service Knowledge	Technical Knowledge
Layman	No	No	No
Domain expert	Yes	Yes	No
Technical expert	No	No	Yes
Advanced	Yes	Yes	Yes

2.1.2 Services & compositions

For a system to support service composition to be effective in a domain, the domain should contain services that can be used in compositions. A composition of services should create value-added functionality for the user. In order to participate in a composition the service should have a clearly defined interface.

2.1.3 Human-Computer Interaction

Human-Computer Interaction is the study of interactions between a human and a computer [5]. These interactions often take place through the user interface of a computer system. Human-Computer Interaction is an important aspect in user-centric service composition. In the overview of domains we will take the devices and user interface of these devices into consideration, because both should support effective service composition.

Some types of devices can introduce constraints on the way in which compositions of services can be created and executed. For example, a small screen makes it harder to use a complex user interface or display large results from services.

2.1.4 Conditions of service composition

We consider the conditions in which the composition of services takes place in the domain, because it influences the way users create compositions. The conditions include the available time to create a composition, the available information when a composition is created and the persistence of compositions.

Time and location aspects Given a user, a list of services and a device in the domain, the time and location aspect can influence the way in which the user interacts with the service composition supporting system. A user sitting behind a computer has

more time to create a complex composition compared to a user walking in a city using a mobile device.

Another aspect of time is the moment at which a composition is made. This can either be at design-time or at runtime. Design-time composition takes place before the composition is executed. Runtime service composition takes place when the user requires the service for execution.

Information and context aspects From the users' classification we described earlier we concluded that some users need guidance from the system to create a service composition. This guidance can be provided by the system, but can also be provided otherwise in the domain. In some domains it can be feasible to take lessons before using the system or to be assisted by an expert.

Persistence of compositions The combination of the domain, users, services and devices also forces constraints on the persistence of compositions. If the user needs to put a lot of effort in creating a composition, maybe due to limitations of the device or complexity of the composition, we expect that the user is not willing to create the composition over and over again. It is important to meet the expectations of the user and make it possible to make compositions persistent if requested by the user.

2.2 Domains

Below we discuss domains in which service composition supporting systems can be applied. For each domain we use the criteria as identified in the previous section to characterize the supporting system.

2.2.1 Enterprise

The enterprise domain is where Service-Oriented Architectures [13] and service composition have been applied most often. The benefits of Service-Oriented Architectures include cost and time reductions, which allows companies to adapt to changes in their environment quicker. Originally this only worked for the larger companies, but recently there has been more and more interest from smaller companies.

2. APPLICATION DOMAINS FOR USER-CENTRIC SERVICE COMPOSITION

The proliferation of the Internet and the adoption of Software as a Service (SaaS) solutions allow smaller companies to make use of their software when and wherever they want. The accessibility of SaaS is very good, but software offered in this way mostly focuses on solving one particular problem of a company. This problem can be, for example, customer relationship management, project management or billing. Service composition is required in order to create value-added compositions from these services to support the requirements of small enterprises.

Types of users

The types of users in the enterprise domain vary, because there are a lot of different enterprises, with employees with different skills. We expect large enterprises to have employees with domain and technical knowledge, allowing them to create compositions. Small companies, however, have a higher chance of having employees that lack these specific skills. The lack of technical knowledge will be the largest challenge; entrepreneurs will have some degree of domain knowledge or are willing to invest in the benefits the services can offer them.

In large companies the composer and end-user are more likely to be two different persons. Users of SaaS solutions in small enterprises will create the composition by themselves and become the end-user.

Services & compositions

The services in the enterprise domain can be split into two categories:

1. Private services used by large enterprises. These services are implemented based on explicit requirements to enable proper compositions with existing services.

2. Services offered via the Internet (SaaS applications). The service often starts as a standalone service offered to entrepreneurs, but grows into a service that needs to connect with other services. Creating a composition of these kind of services is often hard, mostly due to problems with the harmonization of the interfaces.

Human-Computer Interaction

The enterprise domain is often a highly technical domain. The users are used to work with computers for their daily job, which makes service composition via computers possible. We see a trend in the usage of mobile devices for enterprise applications. Using a computer as input and output device does not give restrictions on the way compositions can be made.

Conditions of service composition

The conditions of service composition in the enterprise domain does not impose constraints on how compositions can be created and used. The compositions are normally created at design-time and are used more than once, allowing the user to invest time into creating the composition. The domain furthermore enables users to learn about the service compositions and the available services.

Example composition

A company is using a service for project management and a service for billing. At the end of the month, the owner of the company wants to generate invoices based on the hours registered for a certain project. By creating a composition of the project management service and billing service, he is able to use the registered hours to create a new invoice in the billing service.

2.2.2 Building automation

Although building automation (domotics) is a known phenomenon in office complexes, the application in private homes is currently very limited. Home automation is expected to take a flight in the upcoming years, partially fueled by the need to be sustainable. Service composition is used to improve the performance and simplicity of home appliance networks [28]. Building automation is a kind of smart environment, in which service composition is applied, for example, to facilitate dynamic reconfiguration of services in the environment [32].

Many different services can be found in this domain, especially in a combination of sensors, actuators and communication services. The sensors make it possible to, for example, measure temperature or to determine the current time. Based on this

information devices can be controlled, for example, by turning the heating or the coffee machine on or off.

Types of users

When we talk about building automation in private housing, most users have little technical knowledge. Knowledge of the domain and the services will be more present, but a service composition system will certainly have to give guidance in the composition process. It is also possible that the composition of services is done by an advanced user (e.g. an installer), as is more common in consumer electronics.

Services & Compositions

The services in the building automation domain will mainly consist of sensors, actuators and controlling software. Standardization of these sensors and devices is of great importance and already in full swing. An example is the Open Services Gateway Initiative (OSGi) [25]. The standardization efforts facilitate the composition of services, although there are currently many standards for building automation, which causes harmonization problems during service composition.

Human-Computer Interaction

The interaction of the user with the service composition supporting system can occur in many ways. The creation of compositions for a building automation system after installation can be done on a computer. After installation it is possible that simple compositions are made via a mobile device or via consumer electronics. The complexity of a composition of services has influence on the devices suitable for creating and executing these compositions.

Conditions of service composition

In the process of creating service compositions, we can distinguish two scenarios for which the conditions are also different:

1. The user installs the building automation system by creating compositions of services at design-time. These compositions will be used more than once, so the user will have no problem to invest time in making the compositions.

2. Short-term service compositions, for example when a temporary setting for the heater changes depending on the weather on a given day. This composition should quickly be made, possibly from a mobile device and at runtime.

Example composition

[32] describes a scenario for service composition in smart environments:

> John is coming home from work. At home, he sits down in his living room. Since he has a music service selected in his personal profile, his favorite music starts playing when he enters the room. After some time he walks into the kitchen to prepare some food. The music in the living room stops playing when John leaves the room. Once he enters the kitchen, his new location is detected by the eHome and the music service starts playing Johns music again in his new location using the speakers integrated in the kitchen wall. The music service resumes playing from the last position where John was listening in the living room.

2.2.3 E-commerce

Selling over the Internet is a major source of income for many companies. Consumers are eager to exploit the opportunities to purchase products and services via the Internet. Examples include the purchase of consumer electronics and the booking of flights and hotel rooms. E-commerce can be applied in many different domains, including other domains studied in this chapter.

Currently, services in e-commerce are often only applied in the background and the user is not able to create compositions. Webshops are using the services of various providers to enable consumers to easily compare prices or to make one-stop shopping possible. In [31], two examples of e-commerce solutions in the travel domain based on service composition are mentioned:

1. A travel arrangement service allows a user to create a reservation for several transportation means. By combining the various providers in one shop the user can make use of so called 'one-stop shopping'. The purchase of multiple items in a shop is usually rewarded with a lower price.

2. A flight comparison service gives the user the opportunity to compare flights from different suppliers. By having a large assortment of flights, the chance the user can find a flight matching his needs is bigger.

Currently there is almost no use of user-centric service composition in the e-commerce domain. User-centric service composition would enable the user to perform more powerful and personalized searches over the available information, improving users' overall experience.

Types of users

Users in this domain will generally have little knowledge of service composition. This is probably also the reason why there are so few examples of user-centric service composition. Advertisement and previous experiences can increase the domain knowledge of the user, but there is a strong need for guidance during the composition process. Technical knowledge may not exist, so it is necessary to offer a simple solution for creating service compositions in this domain.

Services & Compositions

The range of services in the e-commerce domain varies. Many providers offer closed interfaces to the information about their products. Their availability is limited and often not accessible for the user. We think that making services better accessible can improve the usage of service composition in this domain.

Human-Computer Interaction

E-commerce transactions take place via the Internet, so, any device with access to the Internet can be used to create service compositions. Traditionally personal computers are used to buy things on the Internet, but younger generations make increasingly use of mobile devices to purchase goods.

Conditions of service composition

In the e-commerce domain, a conversion occurs when a customer takes the marketer's intended action. E-commerce websites always try to optimize the conversion rate in order to increase their profits. Service composition can influence the conversion rate,

because the user lacks domain or technical knowledge to create a service composition. E-commerce websites are more likely to apply service composition in their websites when it also improves conversion ratios [22].

In some cases it may be desirable for a composition to be saved, for example, to compare prices at a later time. We see primarily applications for runtime based service composition in the e-commerce domain.

Example composition

A user wants to book a holiday. Therefore he first chooses airlines that fly to his destination and books a flight. He uses the date and time of the flight from the airline service to compare the prices of car rental services in the neighborhood. Via Google Maps he plans the route from the airport to a nearby hotel. Through a hotel booking service he books a room for the first night after arriving at his destination.

2.2.4 Entertainment

We define the entertainment domain as a domain that contains activities that provide diversion for people in their leisure time. Increasingly more people use computer systems for entertainment, for example, by using multimedia, gaming, mobile applications and social networks to entertain themselves. Companies are harvesting from this need for entertainment by offering paid services, giving the entertainment domain a large overlap with the e-commerce domain.

For example, within social networks, service composition can be found in the widgets provided by service providers. These widgets contain information from external services and are displayed on the profile of the user. The composition of services is unobtrusive for the user. Widgets do not allow complex compositions in which more than two services are involved, but we envision how widgets can start combining information from multiple services into a new service.

Another example of service composition in the entertainment domain is using a website like TripIt [21] to plan a holiday or a day out. The website uses information from external services to support the user to fulfill its goals. These types of websites can be found in both the entertainment and e-commerce domain.

Types of users

As with the e-commerce domain, the users in the entertainment domain can have very different background knowledge. Currently service composition is often not done by the user directly, making the user primarily an utilizer of compositions. Lack of technical knowledge about service composition is probably the main problem in this domain.

Knowledge about the domain and services in the domain will generally be available. For example, for users familiar with social networks, social networks have the power to guide the users in discovering the services a social network is offering.

Services & Compositions

Social networks are offering a lot of services to be used in service compositions. The services have an open character, allowing developers to work with the services and create compositions. Currently these compositions are not complex, but we envision situations where the services become more complex.

Human-Computer Interaction

The users in the entertainment domain will use computers, mobile devices and consumer electronics to create service composition. Users in the composer role will primarily use computers for service composition, where the execution of compositions can also take place through mobile devices.

Conditions of service composition

In the entertainment domain there are a lot of applications for the usage of service composition. The user has time to create a composition at runtime, but also expects it to be available after composition. Users in the entertainment domain are easily influenced by their network and advertisement, making it easy to extend the knowledge of the user about the available services and the composition process. Most compositions of services on the entertainment domain are created at runtime.

Example composition

A good example of the application of service composition in the entertainment domain is the iPhone application Siri [20]. With this 'Virtual Personal Assistant' it is possible

to find location-based information about events, restaurants and movies. Users can create real-time compositions of services by talking to the application. Based on the question of the user, Siri can make recommendations for events in the neighborhood or can make a table reservation in a restaurant.

2.2.5 E-Health

E-health denotes healthcare practices supported by electronic processes and communication. E-health is available both for caregivers in the healthcare sector as for patients and consumers. Prevention of healthcare through the use of smart systems is becoming more popular, allowing people to get insight in their health and get recommendations on how to live healthier. An example of recent innovations in healthcare is the Electronic Patient Dossier in the Netherlands [17].

Service composition in e-health can be compared with building automation. Caregivers are able to create service compositions, for example, by using sensors on a patient's body. The data from these sensors can be used to control actuators. This enables, for example, caregivers to monitor a patient from a distance.

Types of users

In the healthcare sector, the caregivers will be the primary users of service composition systems. Other technical innovations have shown that caregivers are reserved towards changes in their workflow. When introducing service composition this should be considered. We assume caregivers have knowledge about the domain, but the level of technical knowledge will differ between the users.

Users of preventive e-health will use service composition for their own good. We expect that these users may not be fully informed about the possible services in the domain and the level of technical knowledge is even lower.

Services & Compositions

The services in the e-health domain can be compared to the services in the building automation domain. Some components of the systems are sensors and actuators. Composite services containing sensors and actuators are widely available.

Like the services in building automation, the services in the e-health domain rely on standardization in the domain. The compositions can become complex due to the number of required input parameters of a service.

Human-Computer Interaction

The Human-Computer Interaction in the e-health will primarily take place via computers or mobile devices. Computers are used to create the compositions, whereas mobile and health-specific devices are used for the execution of the compositions.

Conditions of service composition

The conditions of service composition in the e-health domain are complex, mainly because critical situations can exist involving patients. We can image this imposes timing constraints on service composition systems. Caregivers should be able to learn how to use service composition to complete their tasks. We think it is feasible for caregivers to learn new techniques like service composition by following courses and proper training.

Example composition

MobiHealth is a company that offers e-health solutions to caregivers [23]. Via mobile sensors the caregiver is able to monitor the health conditions of the patient. A composition of services can be created based on the conditions of the patient and knowledge of the caregiver about these conditions. For example, the caregiver can create a composition of the following services: a location-based service that knows the location of the patient, a mobile sensor that measures the vitals of the patient and a messaging service for sending notifications to the caregiver. A composition of these services enables the caregiver to receive a notification if the vital signs of the patient deteriorate and the patient is not in hospital.

2.3 Conclusions

In this section we select a domain for experimenting with user-centric service composition based on the assessment of the domains in section 2.2.

2.3.1 Conclusions on criteria

Types of users The assessment of domains shows that there are different types of users in each domain. Every domain has users lacking technical knowledge, requiring guidance with service composition. Users lacking domain knowledge are also common across all studied domains. We consider users in the enterprise and e-health domain to have the most domain knowledge.

Services & Compositions The availability of services differs per domain. Home automation and e-health can profit from the standardization efforts going on in these domains. In the enterprise domain, the Software as a Service trend provides in a steady stream of new services. The e-commerce domain mainly lacks open services.

Human-Computer Interaction Human-Computer Interaction in the domains takes place primarily through personal computers and mobile devices. We expect the building automation domain and e-health domain to use more consumer electronics or health-specific devices, but we also see a trend towards using general purpose devices in these domains. An example is the iPhone or iPad with specialized applications for healthcare.

Conditions of service composition

Time constraints during the composition can make service composition a real challenge. In some domains the user has limited time to create a composition or learn how to create a composition. Especially in e-commerce, service compositions should be created and used very quickly. All studied domains except e-commerce have the benefit that users are willing to learn about service composition.

The persistence of compositions is a common requirement among all domains. Each domain has scenarios in which a persistent composition is required.

2.3.2 Selection of domains

Table 2.2 gives an overview of our conclusions about the different domains studied.

For further research in our project we select a domain in which realistic scenarios for service composition can be devised. Our research concentrates on user-centric composition and we want to investigate how different types of users can be supported by a service composition solution to fulfill their goals.

We have not selected building automation and e-health, because some users in these domains are technical experts and may be prepared to learn about service composition. The enterprise domain is not considered because there are no time constraints and because domain experts should be involved in the composition process in this domain.

This leaves us with the e-commerce and entertainment domain. Both domains have users with little technical knowledge and little domain knowledge. The e-commerce domain is always optimizing the conversion rates, therefore the usability of a service composition solution should be high. We expect that proper application scenarios for user-centric service composition can be defined by combining the e-commerce and entertainment domains. We define the application scenarios for the selected domains in the next chapter.

Table 2.2: Key concepts of domains

	Types of users	Services & Compositions	Human-Computer Interaction	Conditions of composition
Enterprise	Domain experts; lack of technical knowledge	Many services; harmonization hard	Computers & mobile devices	No time constraints; Willingness to learn
Building automation	End-users have no technical knowledge; technical experts available	Sensors & actuators; standardization	Computer, mobile devices & consumer electronics	Time constraints; willingness to learn
E-commerce	No domain or technical knowledge	Not many open services	Computers & mobile devices	Conversions important; no time to learn
Entertainment	No technical knowledge; domain knowledge can be learned	Many services	Computers & mobile devices	Willingness to learn; social aspect important
E-health	Domain knowledge; less technical knowledge	Sensors & actuators; standardization	Computers, mobile devices & health specific devices	Willingness to learn; adoption of new techniques hard

3

User-centric service composition scenarios

In Chapter 2 we identified e-commerce and entertainment as domains with realistic scenarios for service composition. Both domains target consumers specially in their free time. We decided to develop a prototype which enables consumers to plan a day out using a composition of services. Services to support this goal are available and compositions of these services improve the ability to satisfy the users' goals.

This chapter starts with an overview of services in section 3.1 and a description of the users in the domain in section 3.2. We define scenarios for the prototype in section 3.3.

3.1 Services

For each service we have used in the scenarios and prototype we identify the goals of the service and the web services that support these goals. We focus on services available in the Netherlands, because the prototype we want to test will primarily be used by Dutch people.

The selection of services is primarily based on the availability of open services. There are many websites offering the information used in our scenarios, but most of these websites don't offer the information as a web service. We selected services that offer a publicly available API and are easy to integrate into the prototype.

3.1.1 Events & activities: Ikdoe

Ikdoe is a Dutch website offering visitors a broad range of information about activities and events in the Netherlands. By selecting a region, type of event or period of time, the website gives suggestions about things to do. The types of activities and events is very broad. Examples of categories with activities are theatre shows, expositions and festivals.

The information offered via the website is made available via a web service. The web service enables external parties to create new events and search events. The event search operation returns events based on several criteria:

- Location: Given a city and some distance, the search returns events in the neighborhood;
- Date: Given a start and end date, the search returns events for a given period of time;
- Category: Given a keyword or a category name, the search returns events in a certain category (e.g., theater shows, exposition or festivals).

The Ikdoe service is able to provide information to fulfill the goal of the user to plan a day out. With the information about an event or activity from Ikdoe, the user can start planning the day. Table 3.1 contains the inputs and outputs of the Ikdoe search operation. Furthermore we describe the type of the input and output parameters in terms of the "IOTypes" ontology. We will give more details about this ontology in section 4.1.

3.1.2 Restaurant information: Iens & Seatme

The websites Iens.nl and Seatme.nl have information on a large collection of restaurants and reviews of these restaurants. Visitors of the websites can find restaurants, read reviews and make reservations. Iens.nl offers information about restaurants in all medium and large cities in the Netherlands, whereas Seatme.nl only focuses on cities like Amsterdam, Rotterdam and The Hague.

Both services provide the user with information about restaurants. Restaurants can be sought based on the current location of the user and the preference for some type

Table 3.1: Inputs and outputs of the Ikdoe search operation

Endpoint	`http://ikdoe.nl`	
Input	keyword	IOTypes.owl:Keyword
	category	IOTypes.owl:ActivityType
	start_date	IOTypes.owl:Date
	end_date	IOTypes.owl:Date
	city_name	IOTypes.owl:City
Output	category	IOTypes.owl:ActivityType
	location	IOTypes.owl:Coordinates
	date	IOTypes.owl:Date

of kitchen (e.g., Italian or Chinese). Tables 3.2 and 3.3 contain the inputs and outputs of the Iens and Seatme services, respectively.

Table 3.2: Inputs and outputs of the Iens search operation

Endpoint	`http://iens.nl`	
Input	city	IOTypes.owl:City
	kitchen	IOTypes.owl:KitchenCategory
Output	address	IOTypes.owl:Address

Table 3.3: Inputs and outputs of the Seatme search operation

Endpoint	`http://seatme.nl`	
Input	city	IOTypes.owl:City
	kitchen	IOTypes.owl:KitchenCategory
Output	address	IOTypes.owl:Address

3.1.3 Hotel information: Kayak

Kayak is a travel search site and contains information about flights and hotels. Based on the location of the user and the date of travel, Kayak can assist the user in finding a hotel. Additional input is the number of guests and the number of rooms the user

wants to reserve. The inputs and outputs of the Kayak service are presented in Table 3.4.

Table 3.4: Inputs and outputs of the Kayak search operation

Endpoint	`http://api.kayak.com`	
Input	city	IOTypes.owl:City
	checkin_date	IOTypes.owl:Date
	checkout_date	IOTypes.owl:Date
	guests	IOTypes.owl:NumberOfGuests
	rooms	IOTypes.owl:NumberOfRooms
Output	address	IOTypes.owl:Address
	city	IOTypes.owl:City

3.1.4 Weather information: Buienradar

Buienradar.nl is one of the most popular websites with weather information in the Netherlands. Besides the rain radars, they provide a broad range of weather information. The Buienradar API offers an indication of the rainfall on a certain location. The rainfall is expressed in a graph, therefore this service is only interesting for displaying purposes and not for forwarding information to other services in a composition. Table 3.5 contains the inputs of the Buienradar service.

Table 3.5: Inputs and outputs of the Buienradar service

Endpoint	`http://gps.buienradar.nl/getrr.php`	
Input	location	IOTypes.owl:Coordinate
Output	weather information	picture

3.1.5 Location: Google Geocoding & Maps

The Google Maps service enables users to view maps and plan routes. Information about the location of an event and the optimal route to the location are useful when planning a trip. Google Maps provides several web services, including a geocoding API for translating addresses into coordinates (Table 3.6), a reverse geocoding API for

translating coordinates into addresses and cities (Table 3.7) and a directions API for getting driving directions on a map (Table 3.8).

Table 3.6: Inputs and outputs of the Google Geocode service

Endpoint	`http://maps.google.com/maps/api/geocode/`
Input	address IOTypes.owl:Address
Output	location IOTypes:owl:Coordinates

Table 3.7: Inputs and outputs of the Google Reverse Geocode service

Endpoint	`http://maps.google.com/maps/api/geocode/`
Input	location IOTypes.owl:Coordinates
Output	address IOTypes:owl:Address

Table 3.8: Inputs and outputs of the Google Directions service

Endpoint	`http://maps.google.com/maps/api/directions/`
Input	origin IOTypes.owl:Address destination IOTypes.owl:Address
Output	directions textual description

3.2 Users

Users in the entertainment domain are often called consumers. We assume that every user is able to control the device he is using and is able to use the user interface. Furthermore we decided to abstract from the technical layer, which implies that the user does not need any specific technical knowledge to use the service composition system, except from the ability to use the end-user device.

We will group the users of the prototype in two different categories: layman and domain expert. A *layman* is new to the scenario and has no domain knowledge. The system should guide this user in fulfilling his goals, for example, by giving suggestions of the available services. With the suggestions of the system, the user learns which

functionality the service can offer and how this functionality can be composed to achieve his goal. A *domain expert* has more knowledge about the services in the domain. He may have gathered this knowledge during previous interactions with the system or through other means like advertising.

The distinction between layman and domain expert is momentary and relative to the goal and application domain. A certain user can be a domain expert if he wants to find a restaurant, but a layman if he wants to plan the route to an event. For each new goal, the user needs to be classified again. This should be taken into consideration when presenting user interfaces to users, since the user interface depends on the type of user.

3.3 Scenarios

Based on the available services and the description of the users in the domain we have defined eleven scenarios that the prototype should be able to support.

Scenario 1: Find an activity

The user should be able to find an activity in 'Enschede' in the category 'Education' in the Ikdoe database. The activity should take place between 1 August and 10 September 2010. After finding the activity, the user should be able to select the activity.

Scenario 2: Find a restaurant

The user should be able to find Italian restaurants in 'Enschede' and select one of these restaurants.

Scenario 3: Find a hotel

The user should be able to find hotels in 'Amsterdam' with available double rooms. The arrival date is 5 September 2010, the duration of the stay is 2 days. The user should be able to select one of these hotels.

Scenario 4: Get directions

The user should be able to get directions from 'Enschede' to 'Amsterdam'.

Scenario 5: Find a restaurant given an activity

The user should be able to find restaurants given he already selected an activity

to attend. The user should be able to use the location of the activity to find restaurants in the neighborhood of the activity and select one of these restaurants.

Scenario 6: Find a hotel given an activity

The user should be able to find hotels given he already selected a restaurant to eat. The user should be able to use the location of the restaurant to find hotels in the neighborhood of the restaurant. The user should be able to use the date of the activity as the start or end date of his stay at the hotel and select a hotel.

Scenario 7: Find an activity given a hotel

The user should be able to find activities given he already selected a hotel to sleep. The user should be able to use the location of the hotel to find nearby activities and select one of these activities.

Scenario 8: Find an activity given a hotel and a restaurant

The user should be able to find activities given he already selected a hotel to stay and a restaurant to eat. The user should be able to use the location of the hotel or restaurant to suggest nearby activities and select one of these activities.

Scenario 9: Get weather information given an activity

The user should be able to get weather information given an activity he has planned. The weather information should be specific for the location of the activity.

Scenario 10: Get directions given a restaurant

The user should be able to get directions to or from an already selected restaurant. The origin or destination of the directions should be based on the location of the restaurant.

Scenario 11: Get weather information using an address

The user should be able to get weather information. Weather information is provided based on a coordinate on a map. The user should be able to use a geocode service to translate an address to a coordinate.

The scenarios will be used to define the architecture of the client application in the next chapter.

4

Architecture of the client application

In this chapter we present the architecture of the client application we have created to support the scenarios described in the previous chapter. Figure 4.1 displays the architecture of the prototype containing the client application.

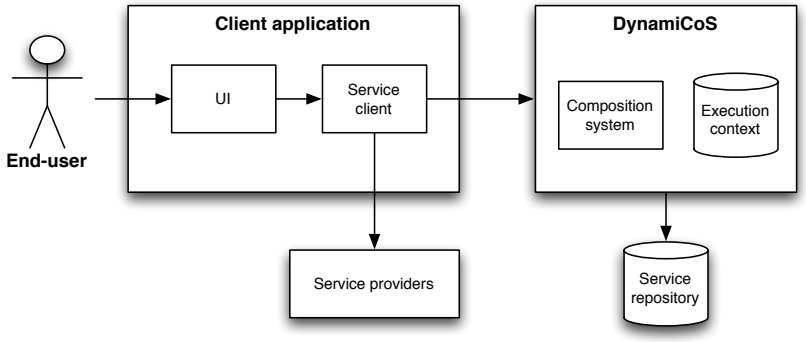

Figure 4.1: Architecture of the prototype - The architecture containing the client application and DynamiCoS

The prototype contains the client application with a user interface and an interface to the DynamiCoS framework [9, 11]. The DynamiCoS framework provides support for dynamic service composition and is accessible as a web service, the framework is introduced in section 4.1. The interactions between the client application and the framework are described in Chapter 5.

Section 4.2 introduces the architecture to present the user with the appropriate user

interfaces for accessing services. The architecture for enabling service composition is presented in section 4.3.

4.1 DynamiCoS framework

The DynamiCoS framework provides a generic solution for user-centric service composition. It's goal is to support the different stakeholders in the composition process allowing them to access services capable of satisfying their needs. DynamiCoS can be accessed as a web service and interaction takes place via predefined types of interaction. The interaction types required to support the client application in the selected application domain will be presented in Chapter 5.

DynamiCoS has access to a repository of services containing information about all available services for the composition process. The services are described using SPATEL [2], which allows to semantically describe services. Each service description contains information about input parameters, output parameters and service goals, accompanied with a semantic type. The semantic type is used in the composition process, but also provides valuable information for the user interface.

4.2 Domain specific services

The goal of the client application is to use the DynamiCoS framework and present the generic service composition solution in a user-centric user interface. The semantic information provided by the DynamiCoS framework should be translated to user interface elements and the input and output of the services. The process of translating a semantic description to a syntactic value is called *grounding*, the translation from a syntactic value to a semantic description is called *lifting* [26].

In order to make the client application as user-centric as possible, we decided to handle the grounding and lifting in the client application instead of in the framework. Grounding and lifting is done in three parts of the client application: when providing user interface element for the input of a service, when executing the service and when presenting the output of the service.

4.2.1 Domain- & service-specific input

Each service has a certain number of input parameters. The client application is responsible for displaying an appropriate user interface element to allow the user to provide this input. For example, if a restaurant service requires a 'type of kitchen' as input, the user should be presented with a list of kitchen types instead of an empty text field.

The DynamiCoS framework provides a semantic type for input of a service. By grounding this semantic type we are able to select an appropriate user interface element. The mapping below is used to ground the semantic type to an user interface element.

```
case semantic_type
when "IOTypes.owl#Coordinates"
  :map
when "IOTypes.owl#Date"
  :date_field
when "IOTypes.owl#KitchenType"
  :kitchen_type_selector
when "IOTypes.owl#ActivityType"
  :activity_type_selector
else
  :text_field
end
```

4.2.2 Domain-specific execution information

We decided to implement the management of service execution into the client application instead of the DynamiCoS framework. There are too many differences between the used protocols and authentication to make a generic implementation for service execution in the DynamiCoS framework. Furthermore, much domain-specific knowledge is required to interpret, ground and lift the input and output parameters of the services.

Protocols

SOAP [37], XML-RPC [35] and REST [33] are three commonly used protocols for implementing web services. Although the number of protocols is limited, there are a lot of technical distinctions between services. Implementing the service execution in the client application allows us to take into account these distinctions.

Another advantage of keeping the implementation of the service call in the client application is the use of libraries. For many commonly used web services there are libraries written in the most popular programming languages. For example, for Ruby [27], which has been used in the client application, the Google Geocode API is offered in the Geokit gem [24].

Grounding & lifting

The services used in the client application need appropriate input from the user. The user provides service inputs via the user interface. Each service input is of a given semantic input. Grounding enables the translation of semantic inputs into an appropriate input for the service. The grounding is done by the client application. The grounding process takes also service-specific situations into consideration. An example is an input parameter with semantical type IOTypes.owl:Coordinate. A $\langle longitude, latitude \rangle$ pair is stored within this input parameter. Web services with a coordinate input often contain two input parameters for the coordinate, namely one for longitude and one for latitude.

The output of a service is translated to an appropriate semantic description for interpretation by the composition process, this is called lifting. The result of a web service contains often more information than expressed in the definition of the service in the repository, because not all returned information is valuable for the composition process.

An example result from a service might look like:

```
<restaurant>
  <id>1</id>
  <name>My Italian Restaurant</name>
  <location>Enschede</location>
  <img>http://utwente.nl/my_italian_restaurant.jpg</img>
</restaurant>
```

In this result, all four properties of a restaurant have different semantic value, but also different meanings in the composition. The `id` property can be used to reason about the entity and the `name` and `img` property are solely for display purposes. Only the `location` property can be used in the service composition process. The lifting

mapping should map the service output to the valuable output parameters in the service definition.

Result selection

Services can return different types of results. The client application should anticipate on the type of result and present the user with the results or inform the framework of the results. We distinguish between three types of values a service can return:

Display-only values: Some services return information which cannot be used in the composition of other services. The result should be displayed to the user, but is not mapped to any output parameters of the service. Examples of these services are weather forecasts and driving directions.

Single value: Some values are not directly useful for the user and are therefore not displayed to the user. The value should be mapped to an output of the service in order to be used in the service composition process. An example of these services is a geocode service, which translates an address to a coordinate. The coordinate can be used as input for other services, but the user does not need to select the value nor needs to view the coordinate.

Set of values: In case the user performs a search action via a service, the service usually returns a set of values. The user should be able to select a value from the set. After selection, the client application should add the valuable information in the value to the output parameters of the service.

4.2.3 Domain-specific output

After the execution of a service has been completed, the user is presented with the results. Only in case of display-only and multiple values the client application should display the values. The values are service-specific and are therefore implemented for each service. Examples of service-specific user interfaces are a picture with weather information or a map with driving directions.

In case the service returns multiple values, the user interface should allow the user to select a value. The selection of a value can also be service-specific. In case of a

restaurant or hotel locator, the selection of a result might be the result of a reservation for a restaurant or hotel. We decided to abstract from making reservations and only allow the user to select a certain result in the client application.

4.3 Service composition

The client application should enable the user to create a composition of services. Figure 4.2 shows a composition of two services: the FindRestaurant service and the FindLocation service. The FindRestaurant service requires a coordinate as input, but the user does not know the coordinates of his current location. He therefore uses the FindLocation service to fill in the coordinate input parameter. For the FindLocation service to be executed he needs to fill in a location (typically a city). The output of FindLocation serves as input for the coordinate input parameter.

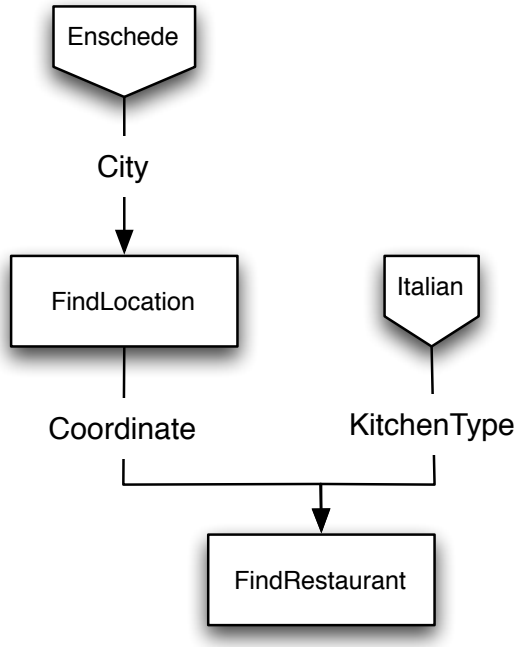

Figure 4.2: Service composition of FindRestaurant and FindLocation -

We identify two methods for service composition. Figure 4.3 shows a *full composition* in which the services in the composition are only executed once all input parameters of all services are available. In this case, the user can leave certain input parameters

empty in the initial form and order to execute the service. The service composition supporting system will respond to this by selecting alternative services which can give a value for the input parameter, replacing the original input with the input parameters of the alternative service.

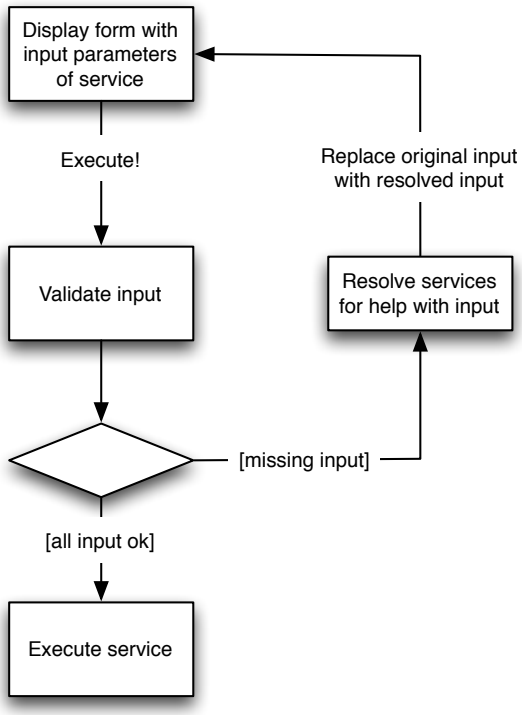

Figure 4.3: Full service composition - The composition is only executed once all input parameters for all services in the composition are available

Figure 4.4 displays a *virtual composition*, which is an alternative composition method. In virtual composition, the services can be executed intermediately to get a result. The resolving service for a certain input parameter is executed and the result is displayed in the original form of the service.

We decided to support virtual service composition in our client application because it gives the user better feedback and flexibility in the composition process. In the case of full composition, the user should leave fields empty in the form. We consider this unnatural and should therefore be explained to the user. With virtual composition, the user is presented with a button to start resolving the input parameter and find new services that can provide the missing input value. In this way help to the user to fill

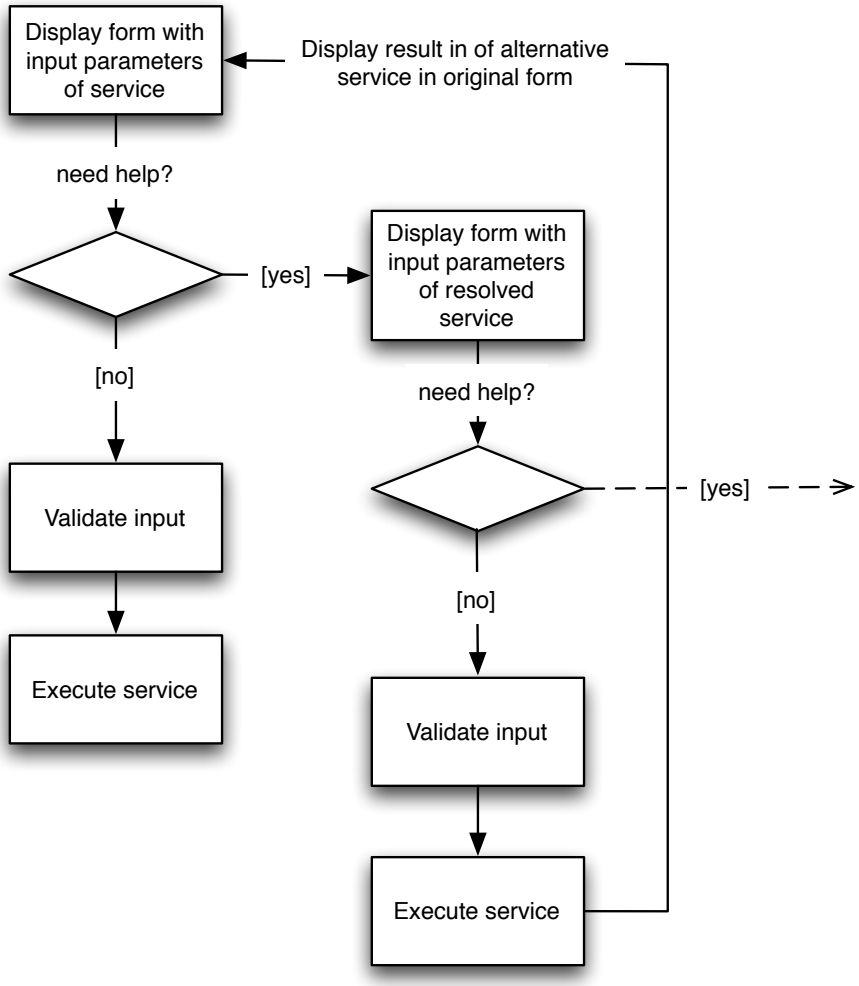

Figure 4.4: Virtual service composition - The resolving service is executed and the result is used in the original form

out the form is not required.

4.3.1 User execution context

In order to support virtual service composition, the client application should be aware of the context of the user. After the execution of the composed service, the client application should redirect the user back to the initial service being executed. DynamiCoS stores information concerning the conversation with the user, therefore the client application can be stateless.

The client application has a local storage containing the results from the service execution. The results are stored for later reference in the user interface and to display the results allowing the user to select a result.

5

Interaction with DynamiCoS

This chapter contains a description of the requirements the DynamiCoS framework should fulfill in order to enable the client application to achieve the user-centric support presented in previous chapters. The framework should be able to keep track of the context in which the user is operating by storing the results of service execution. Based on the context the framework should facilitate the creation of service compositions and guide the execution of the services in the composition. Section 5.1 starts by defining the concept of context. Section 5.2 describes the interactions between the client application and DynamiCoS, which are then presented in sequence diagrams in section 5.3.

5.1 Context

In the process of creating a service composition, the end-user specifies information about the goals he wants to fulfill. Furthermore, the client application can also collect information about the user situation, or user context. This information can be used to create service compositions and to improve the user-experience. Context-awareness of computer applications was first introduced by Schilit et al. in 1994 [34]. In [7], Coutaz et al. introduce the notion of context not only as a state, but part of a process the end-users are involved in. Dey and Abowd [12] define context as:

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

The information collected by the end-user during the composition process can be added to the context of the framework to characterize the situation of the end-user. By using the context in the client application and in the composition process, the user-experience will improve. The context will change during the process of composing services as described by Coutaz et al [7].

An example of context-awareness of the prototype: when an end-user wants to fulfill two goals, say 'visit a festival' and 'have diner', he starts with a service that fulfills the first goal. The 'FindFestival' service allows the user to find a festival, once the end-user selects a festival to visit, his context is enriched. Based on his selection, information about the date, time and location of the festival can be added to the context. Once the user starts to fulfill his second goal 'have diner', the system will suggest a 'FindRestaurant' service. Based on the information in the context, the system is able to give the user suggestions for restaurants in the neighborhood of the festival.

Context awareness of the service composition process can be realized in two different ways: the framework can have mechanisms to make itself aware of the user's context or the client application can make the framework aware of the user's context. In order to make a clear distinction between the two options, we first define the types of context information the context can have.

5.1.1 Types of context information

We define three types of information that can be stored in the context. Figure 5.1 presents these types.

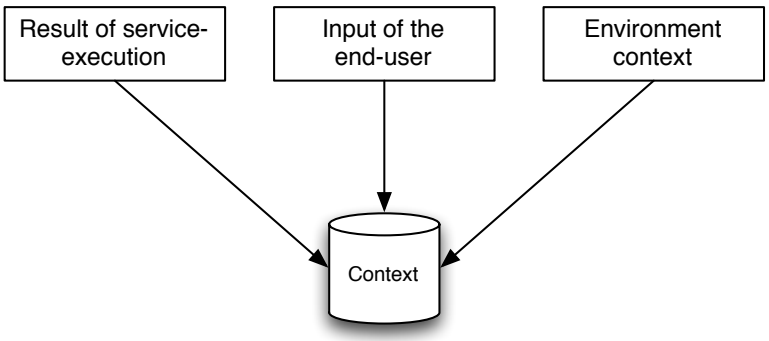

Figure 5.1: Context information - Types of context information.

Results from service execution

The result of a service can be input for another service in the composition. The result of a service execution should therefore be stored in the user execution context. In realistic scenarios a service does not always return a single value and not all returned values will be interesting for the end-user. Therefore, the selection of output parameters with a value for the service composition process is a service specific action.

Input of the end-user

Before executing a service, the end-user fills in the required input parameters of the service. This input should be stored in the context to allow its reuse in other services the user may wish to use. These values are specially appropriate to be used as suggestions for other service inputs.

Environment context

Even before a service is executed or the composition process started, it is possible to add information to the context. For example, when using location aware services it is necessary to know the location of the user. Mobile devices and even certain web browsers are able to get the location of the device and use this information to enrich the context of the framework. Besides location information the client application might be aware of the user using the system for example via a login. The client application can provide information to the context about the user, his preferences, his profile or even previous compositions.

5.1.2 Context awareness of the framework

Context awareness of the framework can be realized in two ways: the framework can use mechanisms to make itself aware of the context or the client application can make the framework aware of the user's context.

In the first case, only results from executed services can be stored in the context. Therefore the framework should execute all services and be aware of the valuable properties in the results. In some cases a service might return a set of results, in this case the framework should be notified by the client which result the end-user selects. In

section 4.2.2 we have identified service execution as a domain specific action, therefore it is complex for the framework to make itself aware of the context.

Letting the client application make the framework context aware is more approproate than the framework making itself aware of all possible users execution contexts. The responsibility of executing the services lies with the client application, making the framework much more flexible. Furthermore the selection of interesting properties, even from a set of results, is easier for the client application because it already needs to parse the results to display them in the user interface. Via an interaction with the framework the client application can add information to the context of the framework. Therefore we decided to create an interaction which adds results from services and predefined knowledge about the context to the context in the framework, the `add_to_context` interaction. The framework is able to use the input parameters provided by the end-user and add these to the context.

5.1.3 Session management

A user can have multiple session, and furthermore the framework can support multiple users. Because the context is closely related to the end-user of the client application, the framework provides means for session management. The client application sends a unique session id along with each request. Based on this session id the framework provides the right context for the end-user.

5.2 Interaction types

This section introduces the interaction types between the client application and the DynamiCoS framework. Certain interactions return a service description. Figure 5.2 depicts the information stored inside a service description.

5.2.1 Service discovery

The framework supports two types of service discovery: based on info and based on the type.

Info based discovery allows the user to type in a question with his goal. Based on a mapping the framework translates this goal into one or more service types. For example, if the user provides the info “I want to eat”, the framework can respond with

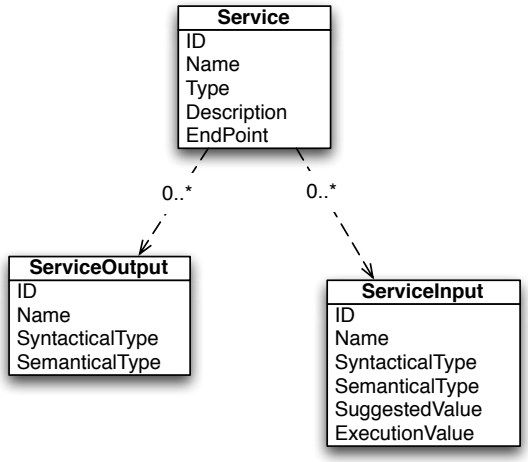

Figure 5.2: Service information - Service information data structure

services that have both the type FindSupermarket and FindRestaurant, because both services support the goal 'eat'.

Type based discovery is the more low-level variant of info based discovery, it refers to the types of services that exist in the registry. Table 5.1 contains the definition of this interaction type. An example request and response for this interaction type can be found in Listing A.1 and A.2 in Appendix A.

Name	$ServTypeDiscovery$
Description	Discovers services based on a service type.
Interface	In: List{$ServType$} Out: List{$ServTypes$, List{$MatchServs$}}
Dependencies	$ServType$ has to refer to a concept from the goal ontology of the framework.

Table 5.1: $ServTypeDiscovery$ Interaction

5.2.2 Resolve service

The resolve service interaction type starts a service composition process in the framework. Once the user requests help with filling in a certain input for a service, the resolve service returns services that can provide an output that semantically matches this input. The user should select a service with the 'select service' interaction (section 5.2.3) to inform the framework of which service to use in the composition.

Table 5.2 contains the definition of the resolve service interaction type. The request in Listing A.3 in Appendix A asks the framework for services capable of providing an output for the input parameter in the `InvalidInputID` tag of the request. The framework responds by returning a list of services and information about which output of these services matches the semantical type of the input parameter, as displayed in Listing A.4 in Appendix A.

Name	*ResolveServ*
Description	Finds services that can provides outputs to match inputs/preconditions of a service previously selected by the user.
Interface	In: *List*{*ServiceID, List*{*InputToResolve*}} Out: *List*{*Servs*}
Dependencies	A service has to have been discovered and selected before.

Table 5.2: *ResolveServ* Interaction

5.2.3 Select service

The service composition interaction between the client application and the framework informs the framework about which service the user wants to execute to complete his goals. The framework returns details (Figure 5.2) about the service, including the input and output parameters required for execution of the service.

The select service interaction is defined in table 5.3. The response from the select service request in Listing A.6, Appendix A, returns full information about the service selected. Included in the response is a `SuggestedValue` tag containing a suggestions for the input parameter. The framework extracts these suggestions from the values in the execution context, that are semantically relevant.

The select service interaction type has an important variant which is used in the composition process. After calling the resolve service interaction type, the client application should inform the framework about the composition it wants to make. Listing A.7, Appendix A, displays the XML request required to inform the framework about which composition should be made. The input parameter id of the first service is connected to the output parameter of the resolved service.

Name	*SeleServ*
Description	Selects a service from a set of services to execute it.
Interface	In(*SeleForType*): *List*{*SeleServID, Type*} In(*SeleForInput*): *List*{*SeleServID, MatchServID*} Out: *List*{*ServInfo*}
Dependencies	Services have to be discovered beforehand, by executing for example $ServInfoDiscovery$ or $ServTypeDiscovery$. There are two types of services selection, one is to select a service for a given service type (*SeleForType*), another is to select a service with a given output to be composed with another service input (*SeleForInput*).

Table 5.3: *SeleServ* Interaction

5.2.4 Validate inputs

Before executing a service, the framework can provide information about the validity of the input the user provided. This interaction stores the input of the user in the context, so it can also be used for temporary storage of input parameters in the virtual composition process.

Table 5.4 contains the details of the validate inputs interaction. Listing A.8, Appendix A, shows a request for validating the input of the end-user. When an input parameter is empty, the `InvalidValue` tag should be send, otherwise the value of the input parameter can be supplied in the `InputedValue` tag. The framework responds with a message containing the invalid input parameters which should be corrected (Listing A.9, Appendix A).

5.2.5 Execute service

The execute service interaction returns the service which should be executed next. This interaction can return three results:

- A service with **ReadyForExec=true**: the service has all inputs for execution and can be executed;

Name	*ValidateInputs*
Description	Validates the service inputs that the user has introduced (*InputedValues*) selected (*SelectedValues*) or that does not know (*InvalidInputs*). In case an inputed/selected input does not match the syntactical type of the input being validated, it is set as Invalid, and returned in the response message of the Interaction.
Interface	In: *List{ServiceID, List{InputID, (InputedValue | SelectedValue | InvalidInput)}}* Out: *List{InvalidInputs}*
Dependencies	The service has to have been selected before its inputs can be validated.

Table 5.4: *ValidateInputs* Interaction

- A service with **ReadyForExec=false**: this service is missing inputs, the user should be presented with the form to input the missing values, or ask for help to resolve them;

- No services: there are no services to be executed.

Table 5.5 contains the definition of the interaction type for executing a service. Listing A.11, Appendix A, displays a response with a service to be executed. The `ExecutionValue` tag contains the value to be used for the input during the execution of the service.

Name	*ExecServ*
Description	Executes a service, or service composition, previously selected by the user. It returns the next service(s) to be executed.
Interface	In: - Out: *ServicesToExecute*
Dependencies	To execute a service it has to have been discovered, selected and validated before hand.

Table 5.5: *ExecServ* Interaction

5.2.6 Add to context

This interaction enables the client application to store information in the execution context of the framework. The definition of this interaction can be found in table 5.6. When a service returns information to be stored in the context, a request as displayed in Listing A.12, Appendix A, is send. The `ServiceOutputValue` tag contains the value to be stored as output of output parameter `ValueID` of service `ServiceID`.

Some services do not result in information which should be added to the context. To inform the framework about the performed execution of the service, the client application should mark the service as executed. This can be done with a request as displayed in Listing A.14, Appendix A, by filling in the service id in the `ServiceIdToAdd` tag.

Name	*AddToContext*
Description	Adds information (user input, service output/result, user context info) to the user execution context. This information is used for future composition, i.e., to provide input information for selected services.
Interface	In: {*ServiceOutputs* | *ServiceInputs* | *UserContextInfo*} Out: -
Dependencies	In case of *ServiceInput* and *ServiceOutputs* the service to which they are attached need to be previously selected.

Table 5.6: *AddToContext* Interaction

5.2.7 Clean context

This interaction cleans the user session context in the framework, allowing the user to start over with the composition process. It also removes session that are not being used from the framework.

5.3 Interaction patterns

In section 5.2 we defined seven interactions between the client application and framework. The client application should be able to fulfill the scenario's defined in section 3.3.

Diagram 5.3 and 5.4 contain sequence diagrams which show the interactions between the end-user, client application, framework and external services.

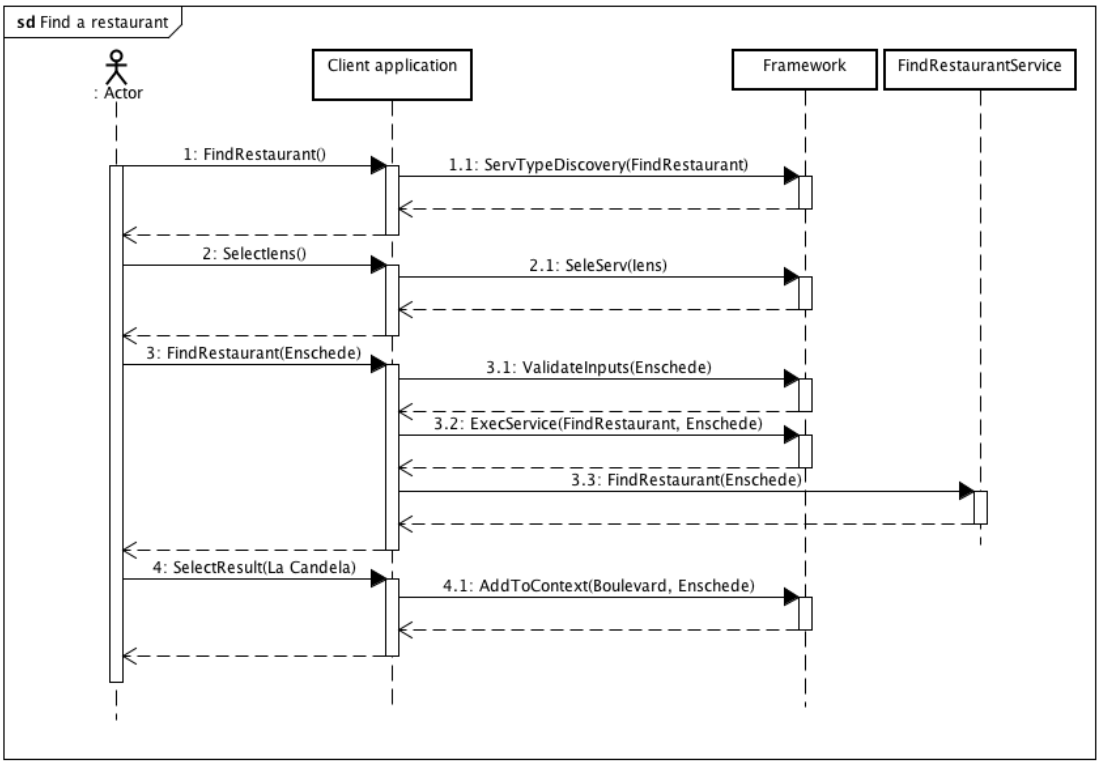

Figure 5.3: Find restaurant - Sequence diagram displaying the messages between the end-user, client application and framework.

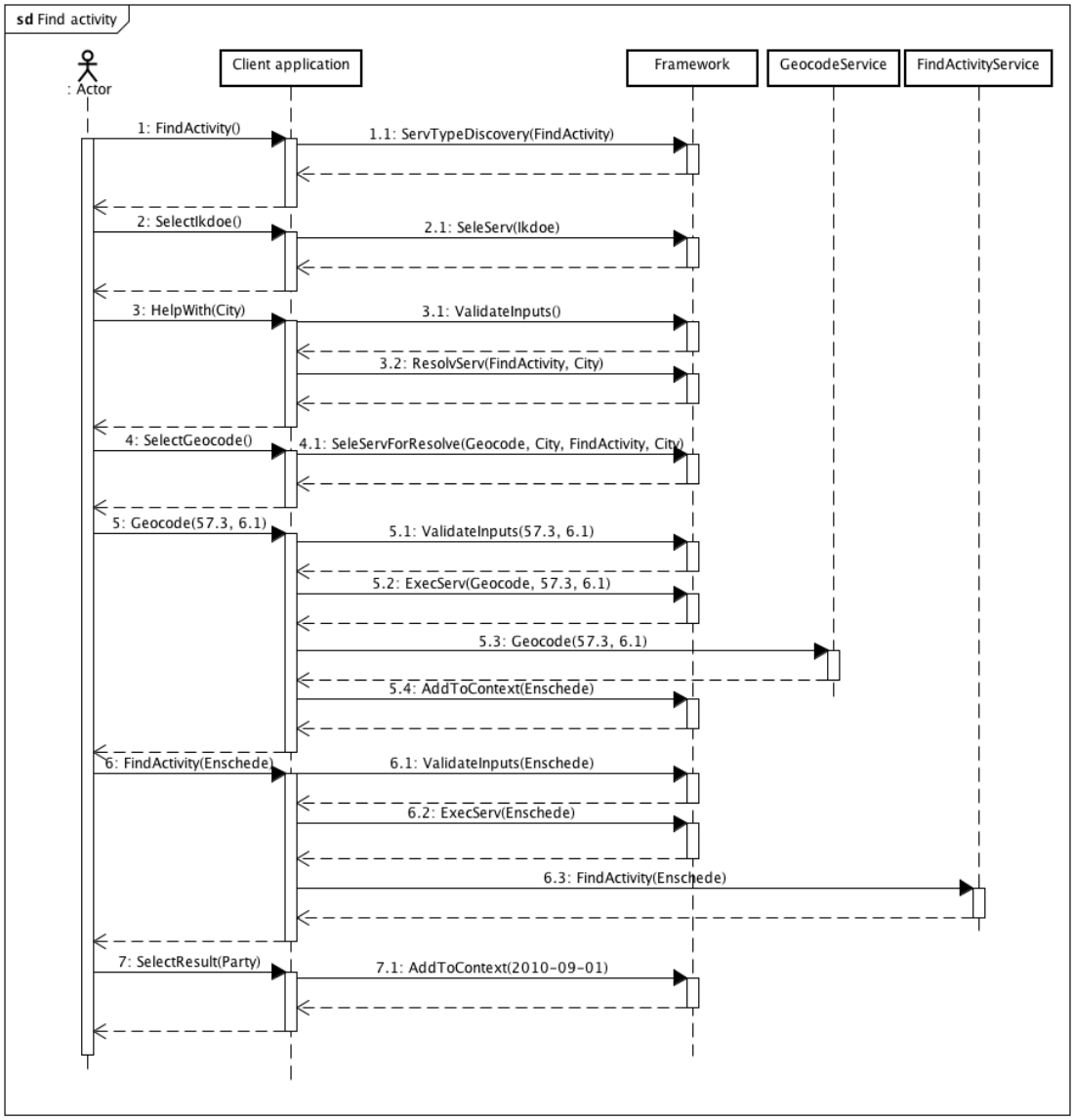

Figure 5.4: Find activity composition - Sequence diagram displaying the messages between the end-user, client application and framework.

6

Evaluation of the prototype

This chapter presents the evaluation of the prototype we have implemented. In section 6.1 we describe the method of acceptance testing we applied to test the scenarios of application of the prototype. Furthermore, we have designed a user test to see how users respond to the developed prototype. The evaluation of the prototype is based on the feedback we gathered during the tests with end-users. Section 6.2 contains the approach we have taken to evaluate the prototype with our user test, and gives the results of this test. Section 6.3 contains an analysis of the results.

6.1 Acceptance test

In section 3.3, we defined eleven scenarios that should be supported by the developed prototype to support user-centric service composition on the entertainment and e-commerce domain. In order to test if the prototype supports these scenarios, we performed an acceptance test [36]. We used the Cucumber [18] test suite to define the scenarios and run the tests. Cucumber supports behavior-driven development and allows end-users and software engineers to define scenarios in a domain specific language. The scenarios can be executed with the prototype, so that the functionality of the prototype can be tested.

The domain specific language allows one to define the steps the end-user performs in the prototype. Each step expresses a part of a test:

- **Preconditions**: A `Given` clause defines preconditions for the test, for example, stating the end-user has selected an activity;

- **Actions**: A `When` clause defines the actions the end-user performs in the prototype, for example, clicking on a link or filling in a form;
- **Assertions**: A `Then` clause defines the postconditions that should hold after the tests, for example, some text should be displayed.

The following example defines a scenario where the end-user wants to find a restaurant. By filling in a form in the prototype, the end-user is presented with a list of restaurants and is able to select a restaurant.

```
Given I am on the homepage
 When I follow "Reserve restaurant"
  And I follow "Find restaurant via Iens"
  And I fill in "City" with "Enschede"
  And I select "Italiaans" from "Kitchen"
  And I press "Go!"
 Then I should see "Result of Find restaurant"
  And I should see "La Candela"
  And I should see "Punto Pasta"
 When I follow "Select La Candela"
 Then I should see "You are going to eat at La Candela"
```

All scenarios for the acceptance tests of the prototype can be found in Appendix B. The prototype has passed the tests for all scenarios.

6.2 User test

In order to evaluate the prototype, we have designed a user test to evaluate the end-users' experience with the prototype.

6.2.1 Approach

The target audience for the test are both technical experts and users with no technical knowledge. The introduction preceding the user test gives the end-users some domain knowledge. By letting them to know which services are available in the prototype, their domain knowledge increases.

The test is performed with five end-users. Three of these end-users are identified as technical experts because they know the concept of service composition. Two end-users are layman since they lack technical knowledge about service composition.

The user test contains of an introduction and a series of tasks the end-user should perform. The introduction informs the end-user of the goals of the prototype: 'enabling end-users to plan a day out'. After the introduction, the end-users were asked to perform the following tasks:

- Find an activity to perform;
- After an activity is found, find a restaurant in the neighborhood of the activity;
- After a restaurant is found, find a hotel in the neighborhood of the activity or restaurant;
- Get directions to the activity or hotel;
- Find the weather conditions in a given city.

These tasks cover all functionality of the developed prototype, including adding information to the context, getting suggestions for services' inputs from the prototype and making a composition of services. The tester presents the end-users with one task at a time, asks the end-user to think out loud and logs the actions and comments of the end-user.

6.2.2 Results

Table 6.1 contains the results from the usability test with end-users.

Besides the general usability results, we also noticed a difference in expectations between end-users. Some end-users, in particular technical experts, did not expect the prototype to suggest values for service input and started looking for these inputs on other websites. For example, when booking a hotel near an activity, they opened the page with information about the activity to find the date of the activity. On the other hand, some end-users, in particular layman, were expecting the prototype to suggest all these values, when the prototype failed to provide these suggestions they remarked that.

No.	Result
R1	After selecting a goal (like 'Find activity'), the end-user is presented with a list of services that can fulfill this goal. All tested end-users find it hard to choose between several services because there is no clear distinction between the services;
R2	If an end-user is presented with a form with many fields, the end-user intends to fill all the fields. Some services do not require an input for all fields for successful service execution, this should be communicated;
R3	Two of the tested end-users, both a laymen and a technical expert, experienced the selection of a result as an odd step. They expect the option to book tickets or reserve a restaurant;
R4	The question mark button in the composition process does not give a clear hint about the actual goal of the button, none of the tested end-users used the button without guidance from the tester;
R5	All end-users did not find the suggestions for the input of services because these suggestions are only displayed once the user starts to type;
R6	When the end-user discovered the suggestions, all tested end-users experienced the suggestions as unclear. If the prototype suggested an address, the end-users asked if the address belongs to the hotel or to the restaurant;
R7	The integration of Google Maps was experienced as difficult by all end-users. In order to select a location on the map, the end-user should make a composition with Google Geocode to translate an address to a coordinate. End-users assumed this selection could be done in the Google Maps instance;
R8	Some services return too many results. For example, finding a hotel in Enschede returns also hotels in neighboring cities. Layman did not expect this behavior, the technical experts knew more about this type of searches and did not found this behavior incorrect;
R9	One of the tested end-users, a laymen, found the flow in the prototype unclear. After getting a weather forecast, the end-user wants to return to the homepage, but did not find an appropriate link for this action.

Table 6.1: Usability test results

6.3 Analysis

Based on the results of the user test presented in the previous section, we analyze the usability of the prototype. We do this by analyzing the *domain specific user-interfaces*, the *composition process* and the *selection of results* in the prototype. We present suggestions for improvements in the interaction patterns between the prototype and the framework.

6.3.1 Domain specific user-interfaces

The framework to support the service composition process, and specifically its primitive interaction types are the basic constructions of the prototype we have developed. The purpose of the framework is to be a generic solution for service composition. During the development of the prototype we have followed this approach. On the other hand, the user interface is more domain specific, i.e., we have given the prototype knowledge of the domain to present a clearer and appropriate user interface. Nevertheless, the usability test shows the prototype still lacks domain specificity. This happens because it serves as client for different types of services, which makes it difficult to have an optimal interface for each service.

The services we use within the framework have their own user interface on their website. This interface is very specific to the service, for example, navigation on a site for finding a restaurant is clearly different from navigating Google Maps. The interface elements for these services ensure optimal usability for the services and make the service centered around the user and his goals.

Within the prototype, we present some service-specific user interface elements, but the complete experience is not specific enough. End-users experienced the selection of a service as difficult (result *R1*). A domain specific user interface for the selection of service could make the selection easier for the end-user (result *R2, R7, R8, R9*).

In chapter 5, we identified interaction patterns between the prototype and the framework. These patterns do not limit the usage of domain specific user interfaces to increase the usability for the end-user. Improving the prototype by making the user interface more specific for the domain is serves can be achieved without changing the defined interaction patterns.

6.3.2 Composition process

An important part of the prototype is the guidance of the service composition process. The prototype provides two ways to perform service composition: virtual compositions and suggestions to the user based on inputs, context information and results of previous service executions.

The virtual composition scenario starts with a first service for which the user needs to input a certain value for a parameter. When the user does not know this value, in

some cases, help can be offered by letting another service fill in the missing value. In the prototype, this action is displayed as a request for help: the end-user asks for help when filling in a value for a parameter. The prototype displays a question mark button to the user, enabling them to find services for a service composition.

The usability test shows that users do not find this help button clear enough (*R4*) and that the button does not fully explains the functionality behind the button. Most end-users associate the question mark with help functionality.

The second form of service composition consists of providing suggestions for input parameters. Based on services previously executed, it is possible to get information from the context and propose these values as input. The advantage of this form of service composition is that the user immediately sees the results and with relatively few actions can obtain a service composition.

The usability tests show that the suggestions in the current prototype provide too little information about the context (*R5, R6*). For an input value of 'place', for example, the cities 'Enschede' and 'Amsterdam' are given as suggestions. For the end-user it is difficult to determine why these values are given as a suggestion.

We identified two improvements to the composition process to improve the usability of the client application:

1. It would be positive for the user interface when both types of service composition are merged. Virtual composition has the disadvantage that it is difficult to communicate the end result with a single button. An option might be to display a list of suggested values in combination with a list of services that can be composed with the input. For example: "Suggestions for 'city': Enschede, Hengelo or search city by hotel". By displaying the input of the resolved service, the user can decide whether or not he can provide this input and the resolved service is valuable for him;

2. To solve the problem with the suggestions, we recommend the suggestions to include information from the context. A valuable suggestion to the end-user may be: "Suggestions for 'city': Enschede (because of Restaurant La Candela) or Hengelo (because of Eden Hotel Hengelo)".

We currently do not know how to incorporate the first improvement above into the interaction types defined in chapter 5. In order to incorporate the second suggestion,

the following changes could be made to the interaction types between the prototype and the framework:

1. Extend the `AddToContext` interaction type with a `Description` field, allowing the prototype to inform the framework about the description of the added information. For example:
   `AddToContext(Enschede, Restaurant La Candela);`

2. The description in the context of the framework should be provided in the response of the `SeleServ` interaction type. This allows the prototype to communicate the description with the suggestion.

6.3.3 Selecting results

Many services return multiple results after service execution, this requires to end-user to make a selection of a result after execution. Adding information to the context after execution is only possible if the end-user selects a result. Selecting a result from a search action is an action the end-user is familiar with, but the purpose in the prototype is slightly different. Usually the selection of a result causes a webpage to be displayed. The selection does not have any consequences for future actions and it is always possible to return to the search results and select a different webpage. In our prototype, the selection of a result means that information is added to the user's context. Multiple selections result in double information in the context.

The selection of results in the current prototype is an unfamiliar action for end-users (*R3*). This action can be improved by making the selection more domain-specific. In the entertainment domain, results of searches consist of activities the end-users wants to perform: visit an event, eat in a restaurant or sleep in a hotel. The selection of an event, restaurant or hotel usually consists of booking a ticket, or reserving a room or a seat. These actions can be used to add information to the context of the prototype and would make the process of selecting a result more intuitive.

The lack of functionality for booking tickets or reserving restaurants and hotels is not only caused by the limitations of the prototype. Many service providers only provide meta-information about their services. They do not offer API's for booking tickets or making reservations. In many situations, the end-user should be redirected to the website of the provider for these actions. For the prototype to be effective, the

end-user should be redirected back to the prototype after completing the reservation process on the provider's website. Based on information given by the provider, the context can be enriched.

The evaluation of the prototype shows us that there a many possibilities to improve the usability of the prototype. By making the prototype more domain-specific, the prototype can present a more natural and usable interface independently of the user characteristics. The framework offers a flexible solution for service composition, but we also identified improvements to the interaction types to better support a user-centric service composition process.

7

Final remarks

In this chapter we give our conclusions and suggestions for further work. Section 7.1 contains the conclusions and answers the research questions we stated in section 1.3. Suggestions for further work can be found in section 7.2

7.1 Conclusions

The result of our research consists of a prototype in the entertainment domain that enables end-users to plan a day out. Services in the prototype allow end-users to find activities, restaurants, hotels, and check weather forecasts and directions. The goals of the end-user are supported in a composition of the existing services. The service composition process is supported by the DynamiCoS framework.

Our research answers the research questions we stated in section 1.3.

1. Identify and characterize possible application domains for user-centric service composition. In order to develop a valuable prototype for user-centric service composition, we started with a literature study on possible application domains for service composition. In section 2.3, we concluded that service composition can be applied in many different application domains, but the characteristics per domain are very different. The building automation and e-health domain give access to domain experts and technical experts in the composition process, but also provide an environment where it is realistic to assume that the users can learn about the advanced technologies to support the service composition process. The enterprise domain can roughly be divided into larger enterprises and small enterprises. We see a trend in

the usage of Software as a Service in small enterprises, increasing the need for service compositions between SaaS providers.

The e-commerce domain and entertainment domain both focus on consumers and impose constraints on the learning curve users want to experience. The domains offer different services and different types of users for which service compositions can be made. The prototype we developed covers both the e-commerce domain and the entertainment domain.

2. What are the application scenarios? For the e-commerce and entertainment domain, we gave an overview of available services and a characterization of the users in the domain. The number of available services is limited, since many large websites with information about events, restaurants and hotels do not give access to this information through web services. We decided to focus on both layman and domain experts user types in the prototype and, in section 3.3, we defined 11 user-centric scenarios which the prototype should support.

3. How should the architecture of the client application look like? In Chapter 4, we defined how the architecture of the client application should look like. The client application should be domain-specific in order to maximize the experience of the end-user, but still follow the support delivered by the DynamiCoS framework. In our architecture the client application handles the domain-specific input of the services, the execution of the services and the domain-specific output of the services. In this way we can present a domain-specific interface to the end-user. Execution of the services is performed in the client application, which allows the framework to be generic and overcome problems with selecting results from service execution.

4. How can the client application interact with the supporting framework? The architecture of the client application determines the interactions between the client application and the framework. In Chapter 5, we defined 7 interactions between the client application and framework. These interactions allow the client application to use the service composition facilities offered by the framework. The interactions allow the framework to be flexible and suitable for a broad range of domains with different types of users.

Chapter 6 gives an evaluation of the prototype, from which we concluded that the prototype is still not domain-specific enough. Providers of the services we use offer service-specific user interfaces to support the goals of the end-user. Improvements in the service composition process and user interface can make the client application more usable.

Our final prototype does not have the level of detail required to create a successful e-commerce or entertainment application, but it is already a suitable proof-of-concept for user-centric service composition. The prototype would be more appropriate for consumers once it is made more domain-specific. We observed that the DynamiCoS framework is able to support the goals of the prototype and offers a flexible solution for dynamic service composition.

7.2 Further work

Our current research shows the potential of the DynamiCoS framework for dynamic service composition. We applied the framework in a prototype within the e-commerce and entertainment domain. Research on applications in other domains should be valuable to improve the flexibility and application of the framework. Especially the enterprise domain, with services where large amounts of data are used, promises to be insightful.

For the e-commerce and entertainment domain, it would be nice to see a more domain-specific prototype. We think a commercial application of the framework in a web application offering service composition to end-users would give more insight in the applicability of the framework in market ready implementations.

In our prototype we experienced that virtual service composition is a process unfamiliar for end-users. This could be improved by providing more insight into the result of the composition. Providing intermediate results from services in the virtual composition process gives some insight for the end-user, but this can be improved. We suggest to do research on proactive execution of services for which the input is known in the context and that return a single result which can be added to the context without interaction of the end-user. This would improve the suggestions to the end-user.

The last challenge we see is one in the presentation of results from services. In some situations the end-user can choose between several services to achieve the same goal. In some situations it would be better to execute several services and get a combination

of results presented. This imposes constraints on the domain-specific visualization of the results and requires harmonization of the output of the services, but would improve the overall user experience.

Appendix A

Example interactions

This appendix contains examples of the interactions types between the client application and the framework.

Listing A.1: Service type discovery request

```
<tns:ServTypeDiscoveryRequest>
  <tns:ServiceType>
    <tns1:Type>Goals.owl#FindRestaurant</tns1:Type>
  </tns:ServiceType>
</tns:ServTypeDiscoveryRequest>
```

Listing A.2: Service type discovery response

```
<ns2:ServTypeDiscoveryResponse>
 <ns2:MatchingService>
  <Type>Goals.owl#FindRestaurant</Type>
  <ServiceInfo>
   <Name>findRestaurant</Name>
   <ID>63ACA6A0-AB8A-11DF-A6A0-BFB463DC9520-5017</ID>
   <Type>Goals.owl#FindRestaurant</Type>
   <Description>
    Seatme.nl contains information of restaurants in Amsterdam and
    allows you to make a reservation for a restaurant.
   </Description>
   <Inputs>
    <Name>city</Name>
    <ID>63ACA6A0-AB8A-11DF-A6A0-BFB463DC9520-5017-InputID-1</ID>
   </Inputs>
   <Inputs>
    <Name>kitchen</Name>
    <ID>63ACA6A0-AB8A-11DF-A6A0-BFB463DC9520-5017-InputID-2</ID>
   </Inputs>
   <Outputs>
    <Name>address</Name>
```

```
    <ID>63ACA6A0-AB8A-11DF-A6A0-BFB463DC9520-5017-OutputID-1</ID>
    <SemanticalType>IOTypes.owl#Address</SemanticalType>
    <SyntacticalType>String</SyntacticalType>
   </Outputs>
  </ServiceInfo>
 </ns2:MatchingService>
</ns2:ServTypeDiscoveryResponse>
```

Listing A.3: Resolve service request

```
<tns:ResolveServRequest>
  <tns:InputToResolve>
    <tns1:ServiceID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618</tns1:ServiceID>
    <tns1:InvalidInputID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618-InputID-3</
        tns1:InvalidInputID>
  </tns:InputToResolve>
</tns:ResolveServRequest>
```

Listing A.4: Resolve service response

```
<ns2:ResolveServResponse>
 <ns2:MatchingService>
  <ServiceID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618</ServiceID>
  <InputMatching>
   <InputID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618-InputID-3</InputID>
   <MatchingValue>
    <MatchingValueID>53DDECF0-A9FC-11DF-B5A7-A8CCE460C335-1112-OutputID-2</
        MatchingValueID>
    <ServiceInfo>
     <Name>findAddressOfLocation</Name>
     <ID>53DDECF0-A9FC-11DF-B5A7-A8CCE460C335-1112</ID>
     <Type>Goals.owl#FindAddressOfLocation</Type>
     <Description>Google Geocode translates a location on a map to the corresponding
          address and city.</Description>
     <Inputs>
      <Name>location</Name>
      <ID>53DDECF0-A9FC-11DF-B5A7-A8CCE460C335-1112-InputID-1</ID>
     </Inputs>
    </ServiceInfo>
   </MatchingValue>
  </InputMatching>
 </ns2:MatchingService>
</ns2:ResolveServResponse>
```

Listing A.5: Select service request

```
<tns:SeleServRequest>
  <tns:ServiceSelectionInfo>
    <tns1:SelectedServiceID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618</
        tns1:SelectedServiceID>
```

```
    <tns1:ServiceType>Goals.owl#FindRestaurant</tns1:ServiceType>
  </tns:ServiceSelectionInfo>
</tns:SeleServRequest>
```

Listing A.6: Select service response

```
<ns2:SeleServResponse>
 <ns2:ServiceInfo>
  <Name>findRestaurant</Name>
  <ID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-9122</ID>
  <Type>Goals.owl#FindRestaurant</Type>
  <Description>Iens is the largest library of restaurants in Holland, containing
      reviews from users. It enables you to find a restaurant in your city and make
      a reservation.</Description>
  <Inputs>
   <SuggestedValue>
    <ValueID>F702BAA0-A542-11DF-BD8F-B171C3DB2465-6049-InputID-5</ValueID>
    <Value>Enschede</Value>
   </SuggestedValue>
   <Name>city</Name>
   <ID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-9122-InputID-3</ID>
   <SemanticalType>IOTypes.owl#City</SemanticalType>
   <SyntacticalType>String</SyntacticalType>
  </Inputs>
  <Inputs>
   <Name>kitchen</Name>
   <ID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-9122-InputID-4</ID>
   <SemanticalType>IOTypes.owl#KitchenType</SemanticalType>
   <SyntacticalType>String</SyntacticalType>
  </Inputs>
  <Outputs>
   <Name>address</Name>
   <ID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-9122-OutputID-2</ID>
   <SemanticalType>IOTypes.owl#Address</SemanticalType>
   <SyntacticalType>String</SyntacticalType>
  </Outputs>
 </ns2:ServiceInfo>
</ns2:SeleServResponse>
```

Listing A.7: Select service request after resolve

```
<tns:SeleServRequest>
  <tns:ServiceSelectionInfo>
    <tns1:SelectedServiceID>53DDECF0-A9FC-11DF-B5A7-A8CCE460C335-1112</
        tns1:SelectedServiceID>
    <tns1:SeleServForInput>
      <tns1:SelectedServiceValueID>53DDECF0-A9FC-11DF-B5A7-A8CCE460C335-1112-
          OutputID-2</tns1:SelectedServiceValueID>
      <tns1:ServiceToResolveID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618</
          tns1:ServiceToResolveID>
```

```
      <tns1:InputID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618-InputID-3</
          tns1:InputID>
    </tns1:SeleServForInput>
  </tns:ServiceSelectionInfo>
</tns:SeleServRequest>
```

Listing A.8: Validate inputs request

```
<tns:ValidateInputsRequest>
  <tns:InputToValidate>
    <tns1:ServiceID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618</tns1:ServiceID>
    <tns1:InputToValidate>
      <tns1:InputID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618-InputID-3</
          tns1:InputID>
      <tns1:InvalidValue />
    </tns1:InputToValidate>
    <tns1:InputToValidate>
      <tns1:InputID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618-InputID-4</
          tns1:InputID>
      <tns1:InputedValue>amerikaans</tns1:InputedValue>
    </tns1:InputToValidate>
  </tns:InputToValidate>
</tns:ValidateInputsRequest>
```

Listing A.9: Validate inputs response

```
<ns2:ValidateInputsResponse xmlns:ns2="http://dynamicos/interactions/ValidateInputs"
     xmlns="http://dynamicos/interactions/BasicTypes">
 <ns2:ServiceValidation>
  <ServiceID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618</ServiceID>
  <InvalidInputID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-7618-InputID-3</
      InvalidInputID>
 </ns2:ServiceValidation>
</ns2:ValidateInputsResponse>
```

Listing A.10: Execute service request

```
<tns:ExecServRequest>
  <tns:ExecServParams>NEXT-SERVICES</tns:ExecServParams>
</tns:ExecServRequest>
```

Listing A.11: Execute service response

```
<ns2:ExecServResponse>
 <ns2:ServiceInfo>
  <Name>findRestaurant</Name>
  <ID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-2659</ID>
  <Type>Goals.owl#FindRestaurant</Type>
  <Description>
```

```
   Iens is the largest library of restaurants in Holland, containing reviews from
       users. It enables you to find a restaurant in your city and make a
       reservation.
  </Description>
  <Inputs>
   <ExecutionValue>Enschede</ExecutionValue>
   <Name>city</Name>
   <ID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-2659-InputID-3</ID>
   <SemanticalType>IOTypes.owl#City</SemanticalType>
   <SyntacticalType>String</SyntacticalType>
  </Inputs>
  <Inputs>
   <ExecutionValue>italiaans</ExecutionValue>
   <Name>kitchen</Name>
   <ID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-2659-InputID-4</ID>
   <SemanticalType>IOTypes.owl#KitchenType</SemanticalType>
   <SyntacticalType>String</SyntacticalType>
  </Inputs>
  <Outputs>
   <Name>address</Name>
   <ID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-2659-OutputID-2</ID>
   <SemanticalType>IOTypes.owl#Address</SemanticalType>
   <SyntacticalType>String</SyntacticalType>
  </Outputs>
  <Endpoint>http://iens.nl/</Endpoint>
  <ReadyForExec>true</ReadyForExec>
 </ns2:ServiceInfo>
</ns2:ExecServResponse>
```

Listing A.12: Add to context request

```
<tns:AddToContextRequest>
  <tns:ValueToAdd>
    <tns1:ValueID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-2659-OutputID-2</tns1:ValueID
        >
    <tns1:ServiceOutputValue>Boulevard 1945 292, 7511 AJ, Enschede</
        tns1:ServiceOutputValue>
    <tns1:ServiceID>9DF029E0-A53B-11DF-A9E0-D18C41B6DDFB-2659</tns1:ServiceID>
  </tns:ValueToAdd>
</tns:AddToContextRequest>
```

Listing A.13: Add to context response

```
<AddToContextResponse>
 <Result>OK</Result>
</AddToContextResponse>
```

Listing A.14: Mark a service as executed in the context

```
<tns:AddToContextRequest>
```

```
   <tns:ServiceIdToAdd>55D479E0-AA09-11DF-B9E0-DEE7FF3DF2D9-2906</tns:ServiceIdToAdd>
 </tns:AddToContextRequest>
```

Appendix B

Scenarios for acceptance tests

B.1 Find an activity

```
Given I am on the homepage
 When I follow "Find activity"
  And I select "Overige" from "Category"
  And I fill in "City name" with "Enschede"
  And I fill in "Start date" with "2010-08-01"
  And I fill in "End date" with "2010-09-10"
  And I press "Go!"
 Then I should see "Result of Find activity"
  And I should see "Businessclub Jong Gedaan Netwerkborrel"
 When I follow "Businessclub Jong Gedaan Netwerkborrel"
 Then I should see
          "You are visiting Businessclub Jong Gedaan Netwerkborrel"
```

B.2 Find a restaurant

```
Given I am on the homepage
 When I follow "Reserve restaurant"
  And I follow "Find restaurant via Iens"
  And I fill in "City" with "Enschede"
  And I select "Italiaans" from "Kitchen"
  And I press "Go!"
 Then I should see "Result of Find restaurant"
```

```
  And I should see "La Candela"
  And I should see "Punto Pasta"
 When I follow "Select La Candela"
 Then I should see "You are going to eat at La Candela"
```

B.3 Find a hotel

```
Given I am on the homepage
 When I follow "Book hotel"
  And I fill in "City" with "Amsterdam"
  And I fill in "Rooms" with "1"
  And I fill in "Guests" with "2"
  And I fill in "Checkin date" with "2010-09-05"
  And I fill in "Checkout date" with "2010-09-07"
  And I press "Go!"
 Then I should see "Result of Find hotel"
  And I should see "Mila's Vila B&B"
 When I follow "Mila's Vila B&B"
 Then I should see "You are staying at Mila's Vila B&B in Amsterdam"
```

B.4 Get directions

```
Given I am on the homepage
 When I follow "Get directions"
  And I follow "Find directions via Google"
  And I fill in "Origin" with "Enschede"
  And I fill in "Destination" with "Amsterdam"
  And I press "Go!"
 Then I should see "Result of Find directions"
  And I should see "Merge onto A1"
```

B.5 Find a restaurant given an activity

```
Given I am on the homepage
  And I am visiting
        "Businessclub Jong Gedaan Netwerkborrel" in "Enschede"
```

```
 When I follow "Reserve restaurant"
  And I follow "Find restaurant via Iens"
 Then I should get "Enschede" suggested for "City"
 When I select suggestion "Enschede" for "City"
  And I select "Italiaans" from "Kitchen"
  And I press "Go!"
 Then I should see "Result of Find restaurant"
  And I should see "La Candela"
  And I should see "Punto Pasta"
 When I follow "Select La Candela"
 Then I should see "going to eat at La Candela"
```

B.6 Find a hotel given an activity

```
Given I am on the homepage
  And I am visiting
        "Businessclub Jong Gedaan Netwerkborrel" in "Enschede"
 When I follow "Book hotel"
 Then I should get "Enschede" suggested for "City"
 When I select suggestion "Enschede" for "City"
  And I fill in "Rooms" with "1"
  And I fill in "Guests" with "2"
  And I fill in "Checkin date" with "2010-09-05"
  And I fill in "Checkout date" with "2010-09-07"
  And I press "Go!"
 Then I should see "Result of Find hotel"
  And I should see "Eden Hotel de Broeierd"
 When I follow "Eden Hotel de Broeierd"
 Then I should see "staying at Eden Hotel de Broeierd"
```

B.7 Find an activity given a hotel

```
Given I am on the homepage
  And I am staying at "Eden Hotel de Broeierd" in "Enschede"
 When I follow "Find activity"
  And I follow "Find activity"
```

```
 Then I should get "Enschede" suggested for "City name"
 When I select suggestion "Enschede" for "City name"
  And I select "Overige" from "Category"
  And I fill in "Start date" with "2010-08-01"
  And I fill in "End date" with "2010-09-10"
  And I press "Go!"
 Then I should see "Result of Find activity"
  And I should see "Businessclub Jong Gedaan Netwerkborrel"
 When I follow "Businessclub Jong Gedaan Netwerkborrel"
 Then I should see "visiting Businessclub Jong Gedaan Netwerkborrel"
```

B.8 Find an activity given a hotel and a restaurant

```
Given I am on the homepage
  And I am staying at "Eden Hotel de Broeierd" in "Enschede"
  And I am eating at "La Fontanella" in "Oldenzaal"
 When I follow "Find activity"
  And I follow "Find activity"
 Then I should get "Enschede" suggested for "City name"
 When I select suggestion "Enschede" for "City name"
  And I select "Overige" from "Category"
  And I fill in "Start date" with "2010-08-01"
  And I fill in "End date" with "2010-09-10"
  And I press "Go!"
 Then I should see "Result of Find activity"
  And I should see "Businessclub Jong Gedaan Netwerkborrel"
 When I follow "Businessclub Jong Gedaan Netwerkborrel"
 Then I should see "visiting Businessclub Jong Gedaan Netwerkborrel"
```

B.9 Get weather information given an activity

```
Given I am on the homepage
  And I am visiting
    "Businessclub Jong Gedaan Netwerkborrel" in "Enschede"
 When I follow "Weather forecast"
 Then I should get "(52.2214,6.89513)" suggested for "Location"
```

```
 When I select suggestion "(52.2214,6.89513)" for "Location"
  And I press "Go!"
 Then I should see "Result of Rain forecast"
```

B.10 Get directions given a restaurant

```
Given I am on the homepage
  And I am eating at "La Candela" in "Enschede"
 When I follow "Get directions"
  And I follow "Find directions via Google"
 Then I should get "Boulevard 1945 292, 7511 AJ, Enschede"
          suggested for "Origin"
  And I should get "Boulevard 1945 292, 7511 AJ, Enschede"
          suggested for "Destination"
 When I fill in "Origin" with "Drienerlolaan 5, Enschede"
  And I select suggestion "Boulevard 1945 292, 7511 AJ, Enschede"
          for "Destination"
  And I press "Go!"
 Then I should see "Result of Find directions"
  And I should see "At the roundabout,
          take the 2nd exit onto Auke Vleerstraat"
```

B.11 Get weather information using an address

```
Given I am on the homepage
 When I follow "Weather forecast"
  And I request help for "Location"
  And I follow "Find location via Google"
  And I fill in "Address" with "Drienerlolaan 5, Enschede"
  And I press "Go!"
 Then the "Location" field should contain "(52.2411776,6.8518822)"
 When I press "Go!"
 Then I should see "Result of Rain forecast"
```

References

[1] 37signals. **Simple small business software, collaboration, CRM: 37signals** [online]. Available from: `http://37signals.com`. 2

[2] J Almeida, A Baravaglio, and M Belaunde. **Service creation in the spice service platform**. In *The Proceedings of the 17th Wireless World Research Forum Meeting (WWRF17)*, Heidelberg, Germany, November 2006. 34

[3] C Anderson. *The long tail: how endless choice is creating unlimited demand*. Business Books, Januari 2006. 2

[4] Y Benkler and H Nissenbaum. **Commons-based Peer Production and Virtue**. *The Journal of Political Philosophy*, **14**(4):394–419, 2006. 3

[5] T Chairman-Hewett. **ACM SIGCHI curricula for human-computer interaction**. Technical report, ACM, 1992. 10

[6] R Chinnici, J Moreau, A Ryman, and S Weerawarana. **Web Service Description Language (WSDL) Version 2.0 Part 1: Core Language** [online]. 2007. Available from: `http://www.w3.org/TR/wsdl20/`. 9

[7] J Coutaz, J Crowley, S Dobson, and D Garlan. **Context is key**. *Commun. ACM*, **48**(3):49–53, 2005. 43, 44

[8] J Cupri and C Warner. **Enterprise Mashups Part 1: Bringing SOA to the People**. *SOA Magazine*, **XVIII**, May 2008. 2

[9] E da Silva, L Pires, and M van Sinderen. **DynamiCoS framework** [online]. Available from: `http://dynamicos.sourceforge.net/`. 4, 33

[10] E da Silva, L Pires, and M van Sinderen. **On the Design of User-centric Supporting Service Composition Environments**. In S. Latifi, editor, *IEEE International Conference on Information Technology: New Generations*, pages 666–671, Los Alamitos, April 2010. IEEE Computer Society Press. 3, 8, 10

REFERENCES

[11] E da Silva, L Pires, and M van Sinderen. **On the Support of Dynamic Service Composition at Runtime**. In *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, **6275** of *Lecture Notes in Computer Science*, pages 530–539. Springer Berlin / Heidelberg, 2010. 4, 33

[12] A Dey, G Abowd, P Brown, N Davies, M Smith, and P Steggles. **Towards a better understanding of context and context-awareness**. In *HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307, Karlsruhe, Germany, 1999. 43

[13] Thomas Erl. *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007. 2, 11

[14] FreshBooks. **FreshBooks** [online]. Available from: `http://freshbooks.com`. 2

[15] Google. **Gmail** [online]. Available from: `http://www.gmail.com`. 2

[16] Google. **Google Apps** [online]. Available from: `http://www.google.com/apps`. 2

[17] Dutch Government. **Elektronisch Patiëntendossier** [online]. Available from: `http://www.rijksoverheid.nl/onderwerpen/elektronisch-patientendossier`. 19

[18] A Hellesøy. **Cucumber** [online]. 2010. Available from: `http://cukes.info/`. 55

[19] V Hoyer and K Stanoevska-Slabeva. **Towards a Reference Model for Grassroots Enterprise Mashup Environments**. In *Proceedings of the 17th European Conference on Information Systems (ECIS 2009)*, Verona, Italy, 2009. 3

[20] Siri Inc. **Siri - Your Virtual Assistant** [online]. Available from: `http://siri.com/`. 18

[21] TripIt Inc. **TripIt - Trip Planner** [online]. Available from: `http://www.tripit.com/`. 17

[22] H Kuan, G Bock, and V Vathanophas. **Comparing the Effects of Usability on Customer Conversion and Retention at E-Commerce Websites**. In *HICSS '05: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, page 174.1, Washington, DC, USA, Jan 2005. IEEE Computer Society. 17

[23] P Lems, R Bults, and D Knoppel. **MobiHealth - Putting care in motion** [online]. Available from: `http://www.mobihealth.com/home/en/home.php`. 20

[24] A Lewis and B Eisenhauer. **Geokit for Ruby & Rails** [online]. Available from: `http://geokit.rubyforge.org/`. 36

[25] D Marples and P Kriens. **The Open Services Gateway Initiative: an introductory overview**. *Communications Magazine, IEEE DOI - 10.1109/35.968820*, **39**(12):110–114, 2001. 14

[26] D Martin, M Burstein, D Mcdermott, S Mcilraith, M Paolucci, K Sycara, D McGuinness, S Evren, and N Srinivasan. **Bringing semantics to web services with OWL-S**. *World Wide Web*, **10**(3):243–277, 2007. 34

[27] Y Matsumoto. **Ruby Programming Language** [online]. Available from: `http://www.ruby-lang.org/`. 36

[28] A Mingkhwan, P Fergus, and O Abuelma'Atti. **Dynamic service composition in home appliance networks**. *Multimedia Tools Appl.*, **29**(3):257–284, 2006. 13

[29] OASIS. **Services Business Process Execution Language Version 2.0** [online]. Available from: `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`. 3, 9

[30] Paul. **HousingMaps** [online]. Available from: `http://housingmaps.com`. 2

[31] D Postmus. *The supply chain of enterprise software: strategy, structure, and coordination.* PhD thesis, University of Groningen, 2009. 15

[32] Daniel Retkowitz and Mark Stegelmann. **Dynamic adaptability for smart environments**. In *DAIS'08: Proceedings of the 8th IFIP WG 6.1 international conference on Distributed applications and interoperable systems*, pages 154–167, Berlin, Heidelberg, 2008. Springer-Verlag. 13, 15

[33] A Rodriguez. **RESTful Web services: The basics** [online]. Available from: `https://www.ibm.com/developerworks/webservices/library/ws-restful/`. 35

[34] B Schilit, N Adams, and R Want. **Context-Aware Computing Applications**. *Mobile Computing Systems and Applications, IEEE Workshop on*, **0**:85–90, 1994. 43

[35] Userland Software. **XML-RPC Home Page** [online]. Available from: `http://www.xmlrpc.com/`. 35

[36] Methods & Tools. **Acceptance TDD explained** [online]. 2007. Available from: `http://www.methodsandtools.com/archive/archive.php?id=72`. 55

[37] W3C. **SOAP Version 1.2** [online]. April 2007. Available from: `http://www.w3.org/TR/soap/`. 35