Deliverable reference:

Date:

12th Feb 2009

Title:

Final report

Project Title:

Semantic Interfaces for Mobile Services

SIMS

Contract no. 027610

Responsible partner:

SINTEF

Editors:

Richard Sanders

Approved by:

Technical Manager Jacqueline Floch

SIXTH FRAMEWORK PROGRAMME:
PRIORITY 2.3.2.3
SPECIFIC TARGETED RESEARCH OR
INNOVATION PROJECT

Classification: (Confidential or open)

Confidentiality: Open

Dissemination Level: PU (Public)

Abstract:

The core idea of SIMS was that semantic interfaces provide new means to specify and design service components and to guarantee compatibility in static and dynamic component compositions. Compared to the well known static interfaces currently in use, semantic interfaces also define the dynamic behaviour and the goals of the collaboration across an interface. This enables safety and liveness properties to be checked effectively and to support service discovery and service composition at runtime with compatibility guarantees. Ontologies are used to enrich the services with semantics.

SIMS aimed at providing tools for design and validation of service components with semantic interfaces, and middleware that enables discovery and validation of service opportunities between peers in ad-hoc interactions. SIMS ran from January 2006 through October 2008.

This document presents the final activity report, and is suitable for direct publication by the Commission; hence it is intended to be broadly comprehensible to an interested general reader. It describes the challenges that SIMS addressed, who benefits from the results, highlights the achievements and the potential impact.

SIMS Consortium

SIMS (Contract No. 027610) is a Specific Targeted Research or Innovation Project (STREP) within the 6th Framework Programme, Priority 2.4.5 (Mobile and Wireless Systems beyond 3G).

The results of SIMS are available from the following web sites:

- The web site for the project, http://www.ist-sims.org/, contains all public deliverables and articles, as well as the project brochure and related links. The SIMS open source tools and the accompanying tool tutorial are available for the general public for experimentation use.
- WUT hosts a SIMS-related web site (http://meag.tele.pw.edu.pl/sims). The site makes available all ontologies developed, describes ontology-based techniques, and publishes ontology-based software developed by WUT within the SIMS project.

The SIMS consortium members are:

SINTEF ICT (Co-ordinator) NO-7465 Trondheim, Norway Phone: +47 73 59 30 00 Contact person: Richard Sanders Email: richard.sanders@sintef.no	
France Telecom España, S.A. (withdrew Dec'07) Paseo del Club Deportivo, 1 Edificio 8 28223 Pozuelo de Alarcón, Madrid, Spain Phone: +34 9350 21836 Contact person: Zulima Saenz Email: zulima.saenz@orange-ftgroup.com	**Appear Networks AB** Kista Science Tower, 16451 Kista, Sweden Phone: +46 8 545 91 370 Contact person: Xavier Aubry Email: xavier.aubry@appearnetworks.com
Gentleware AG Ludwigstrasse 12, 20357 Hamburg, Germany Phone: +49 40 24 42 53 30 Contact person: Marko Boger Email: marko.boger@gentleware.com	**Gintel AS** Otto Nielsens vei 12, 7004 Trondheim, Norway Phone: +34 913 44 574 Contact person: Bjørn Gulla Email: bjorn.gulla@gintel.no
Norwegian University of Science and Technology (NTNU) NO-7491 Trondheim, Norway Phone: +47 73 59 50 00 Contact person: Peter Herrmann Email: herrmann@item.ntnu.no	**Warsaw University of Technology (WUT)** Pl. Politechniki 1, 00-661Warsaw, Poland Phone: +48 22 825 1409 Contact person: Jaroslaw Domaszewicz Email: domaszew@tele.pw.edu.pl

Table of Contents

1 Introduction

1.1 SIMS motivation and background

The core idea of SIMS is that semantic interfaces provide new means to specify and design service components and to guarantee compatibility in static and dynamic component compositions. Compared to the well known static interfaces currently in use, semantic interfaces also define the dynamic behaviour and the goals of the collaboration across an interface. This enables safety and liveness properties to be checked effectively and to support service discovery and service composition at runtime with compatibility guarantees.

The SIMS solution is not about making services which cannot otherwise be realized; it's about realizing them in a better way by ensuring behavioural compatibility, facilitating deployment and design driven composition, and enabling independent actors to participate in a service marketplace.

1.2 Structure of this document

In chapter 2 we present the challenges that SIMS set out to address, and in describe in chapter 3 how we have attempted to solve them.

In chapter 4 we discuss who can benefit from the results that are highlighted in chapter 5.

In chapter 6 we define the key results of the project, relate them to market needs, identify current and future competitors, and present a SWOT analysis. We also list the contribution to standards and Open Source. Chapter 7 sketches the pilot services developed in SIMS.

Chapter 8 tells what results are available to the general community, while chapter 9 discusses the potential impact of the result to different kinds of stakeholders.

1.3 SIMS deliverables

All public deliverables are available on the project web site, http://www.ist-sims.org/. They include:

- D2.3 *SIMS Principles and Techniques for Service Engineering* and D3.4 *Techniques for ontology-driven semantic interface artefacts*, which describe the design and validation techniques
- D4.3 *Open Source design tool components* describes the design tool support
- D5.5 *Overall SIMS architecture* which describes the mechanisms of the middleware support and service marketplace
- D6.5 *Trial services* describes the Courier Services developed to assess the benefits of the approach
- D6.6 *Evaluation of SIMS approach* gives our own assessment of what has been achieved
- D7.8 *Final plan for using and disseminating knowledge* tells how the partners plan to make use of the knowledge gained by the project, and how the project has contributed to Open Source and standards.

2 The Challenge

The Service Oriented Architecture paradigm (SOA) is increasingly gaining acceptance, influencing the way people understand and define services. However, one should be aware of the fundamental limitation of SOA as it is currently understood. In SOA, services are provided by a service provider to a service consumer. This service provider is normally a “passive object” in the sense that it never takes any initiatives towards a service user. It merely awaits requests and responds to them.

Collaborative services on the other hand are performed by objects that may take initiatives towards the service users. This is typical for telecom services, but also for many new services such as attentive services, context aware services, notification services and ambient intelligence. Such services in general entail a collaboration among several active objects, and therefore we call them *collaborative services*. Most contemporary uses of SOA fail to consider collaborative services. The SIMS project addresses this gap and defines a service architecture and delivery platform that supports loosely-coupled autonomous service components. This may be seen as a generalization of contemporary SOA, allowing for a wider class of services.

With the number of models of mobile devices and the complexity of mobile OS increasing, collaborative services will face increasing challenges. In other words, there needs to be “compatibility middleware” and “tools to design compatible services” to ensure that devices will be able to communicate/collaborate with each other.

An intricate market
Today’s mobile world is made out of a multitude of different platforms, devices, software, formats, service providers, and applications. We are facing the problem of compatibility between heterogeneous devices with complex applications. Moreover, services are typically exclusive to closed phone operators and there is no environment for developing services that could communicate with different networks/operators/software providers. Mobility is the trend in the industry and the challenge is to develop applications that can work for all.

2.1 Problems faced by developers and software vendors

<u>Modelling of service components:</u>
Currently, protocols need to be specified, implemented and tested manually, which is costly in terms of time and money. There are no integrated solutions for designing services and each new coding takes a lot of time, effort and includes a high risk of errors.

<u>Testing and validation of service components</u>:
It is difficult for software developers to ensure that their development is correct, since there are no easy-to-use validation techniques for distributed service components. The existing validation methods require specialists and are not widely used.

<u>Distribution of service components</u>
It is difficult for developers to sell their service components to closed operators. Due to this, the end-users have only access to service components that were chosen by their service provider.

2.2 Problems faced by operators

<u>Quality of applications</u>
When Service Providers buy components that were developed by different companies and then connected together, it is very difficult to find validation techniques that work for all components, and this makes the validation process long and costly. As a result, small service providers are afraid to buy service components from different vendors because of the challenge of integration.

Also, when applications are corrected manually, the risk of error is higher and it is costly to provide support to end-users that experience problems and errors.

Distribution of applications

The distribution mechanisms and marketing solutions needed to attract more users to adopt mobile applications and services seem to have come to a dead end. Without opt in from end users it is hard to push anything onto their devices. High complexity of OS and device software as well as feature richness of the same devices hinder users to find and adopt new services during the short life span of products. It is always a challenge for service providers to find how to market a service component and make it used extensively, and they constantly look for new and better ways to generate more revenue.

2.3 Problems faced by end-users

Limited choice of service components

End users face a rigid offer of service components, meaning that they only have access to the service components chosen by their device manufacturer and service provider. Two mobiles from different brands and /or service providers can have difficulties performing services together.

Manual search of service components

Finding services and combining them is a challenge. Today the user needs to look actively for service components and download them from the web. It is difficult for the end-users to know about additional service components and to know if they will function with other devices they want to communicate with. Today it is impossible to benefit from a dynamic reconfiguration between devices.

3 Addressing the Challenge: the SIMS solution

3.1 The SIMS development process – an overview

D2.3 *SIMS Principles and Techniques for Service Engineering* describes three different development processes: (1) *top-down*, (2) *bottom-up* and (3) *reuse*. For simplicity, we outline only the first of these.

The top-down process is used to design the services from scratch. This process progresses by refinement, starting from a high level specification of a system, until a fine grained specification is reached. The process has five steps, as shown in Figure 1.

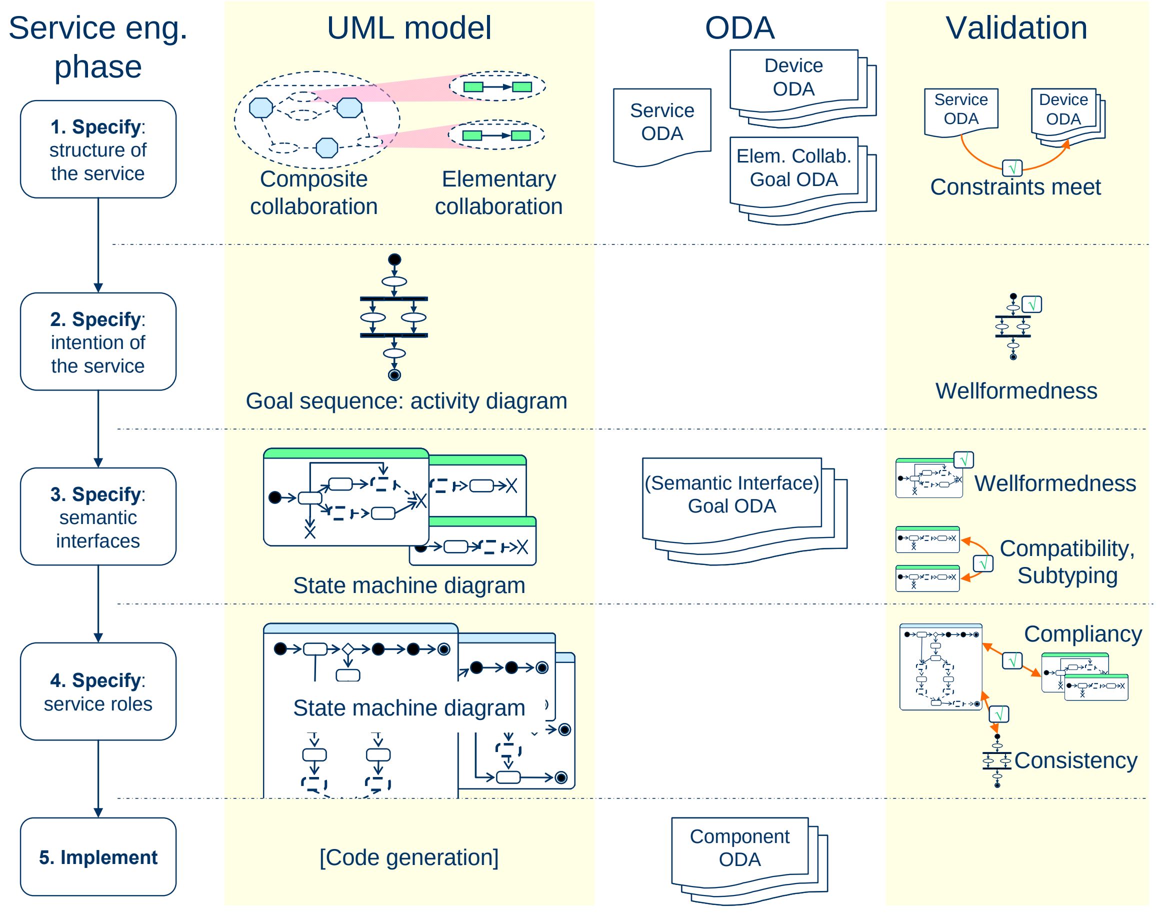


Figure 1: SIMS top-down approach

As the figure shows, SIMS is about engineering collaborative services in UML, describing service semantics using so-called *Ontology-driven artefacts* (ODAs), and validating service behaviour and consistency using integrated tools. Each of these are described in the sections following, as well as how this fits into a service marketplace and how SIMS middleware can support this at runtime:

- Section 3.2 focuses on the kind of services SIMS is dealing with, namely collaborative services, where different entities interact with each other, thus providing the service to the user;
- Section 3.3 gives a quick glance on how such services are specified: how service components are connected, how they behave, or what is the intention of the service they provide;
- Section 3.4 deals with the validation of services in SIMS: checking for errors and usefulness;
- Section 3.5 discusses the principles of the service marketplace and how SIMS supports this;
- Finally section 3.6 presents the middleware support SIMS offers at runtime for validation, deployment and service discovery.

3.2 Collaborative services

We are used in the telecommunication domain to think about a service as some functionality provided to a group of users where the users are distributed and behave independently. Therefore, distribution, concurrency and peer-to-peer communication among users are inherent properties of the services. These properties have influenced the engineering approach used in the telecommunication domain. For example we talk of active objects with concurrent behaviour, asynchronous communication and stateful service sessions. In SIMS we believe that the merit of these concepts is not limited to the telecom domain, and propose to use them for the engineering of collaborative services in general.

By *collaborative services* we mean services that involve collaborations (interactions) between autonomous entities that may take initiatives concurrently. They may involve multiple parties/users. Telecom services are typically collaborative services, but so are many new services such as attentive services, context aware services, notification services and ambient services.

Figure 2 illustrates the concept of collaborative services. The service results from the collaboration between a number of active entities. Entities may be deployed on user devices or server nodes.

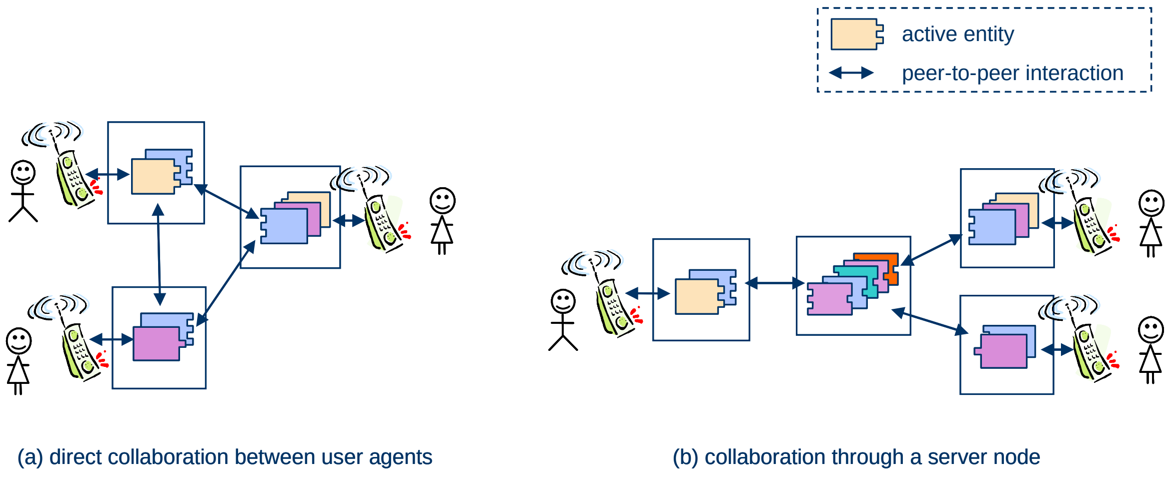

active entity
peer-to-peer interaction
(a) direct collaboration between user agents
(b) collaboration through a server node

Figure 2: The concept of collaborative service

When several entities behave concurrently, *mixed initiatives* may occur, meaning that the interacting entities may take initiatives simultaneously. An example is the case of simultaneous calls, which occurs when two users initiate a call to each other at the same time. Mixed initiatives must be properly handled in order to avoid errors such as deadlocks. A deadlock is a situation where two or more service components are unable to proceed because they wait endlessly for each other. The problem of mixed initiatives is inevitable in collaborative services. When identified, it can be simply resolved.

The telecom domain has traditionally used good tools for dealing with distribution and concurrency. Abstract and formal modelling is used to promote human understanding, and enables formal analysis of the correctness of complex behaviours. This makes it possible to reduce the number of errors and hence increase the quality of systems. In SIMS, we likewise promote abstract modelling and provide validation techniques that enable to check that collaborating entities interact in an error-free and useful way. By useful, we mean that it is possible for entities to achieve some desirable behaviour. For instance, there is no point in having a call setup that is constantly rejected.

Note to the readers familiar with the Service Oriented Architecture (SOA):

Collaborative services differ from the concept of service as usually understood in SOA. In SOA, services are provided by a service provider to a service consumer. This service provider is normally a "passive object" in the sense that it never takes any initiatives towards a service user. It merely awaits requests and responds to them. Collaborative services on the other hand are performed by objects that may take initiatives towards the service users. A "SOA service" would be modelled by a *semantic interface* in SIMS, see below.

3.3 Specification of collaborative services

The rest of the chapter (this section and the following ones) presents shortly the main SIMS concepts. We give a quick overview of how services are specified. Readers who are interested will find more details in D2.3 *SIMS Principles and Techniques for Service Engineering*.

3.3.1 Service specification and service realisation

In SIMS we distinguish between the specification of a service and its realisation, in order to abstract away unnecessary implementation details. By making a clear separation between the service logic and its implementation, the designer is free to focus more on service functionality.

A *service component* (or *component* for short) is a unit of deployment at runtime. A service component may participate in one or more services and provide different functionalities in a service. A component's functionality in a service is modelled by a *service role*. A *service* results from the collaboration between concurrent service roles, see Figure 3. We develop both service roles and service components when specifying the service (at design time).

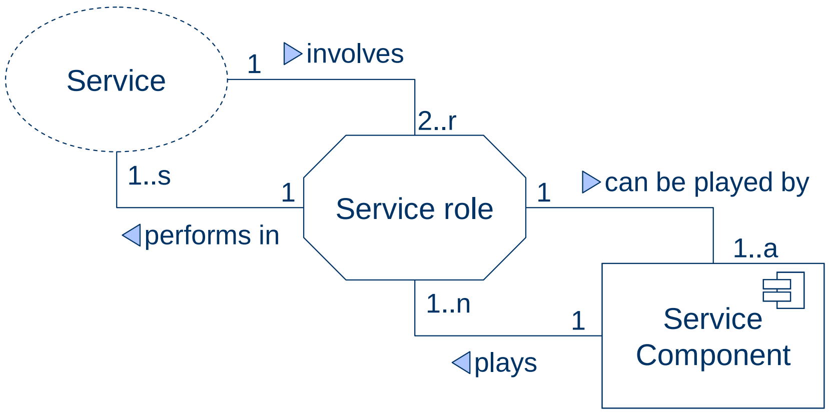


Figure 3: Services, service roles and service components

The distinction between service roles and components also facilitates the design, implementation and validation of components: a service component can play a number of service roles, each service role taking part in a service. This is illustrated in Figure 4, where three components are involved in different manners in two services: component *A* is answering a call made by component *B*, while component *B* is at the same time involved with component *C* in another service. The three service roles shown in the figure, namely *rCAller*, *rCallee* and *rCalleeBusy*, are specified and validated independently.

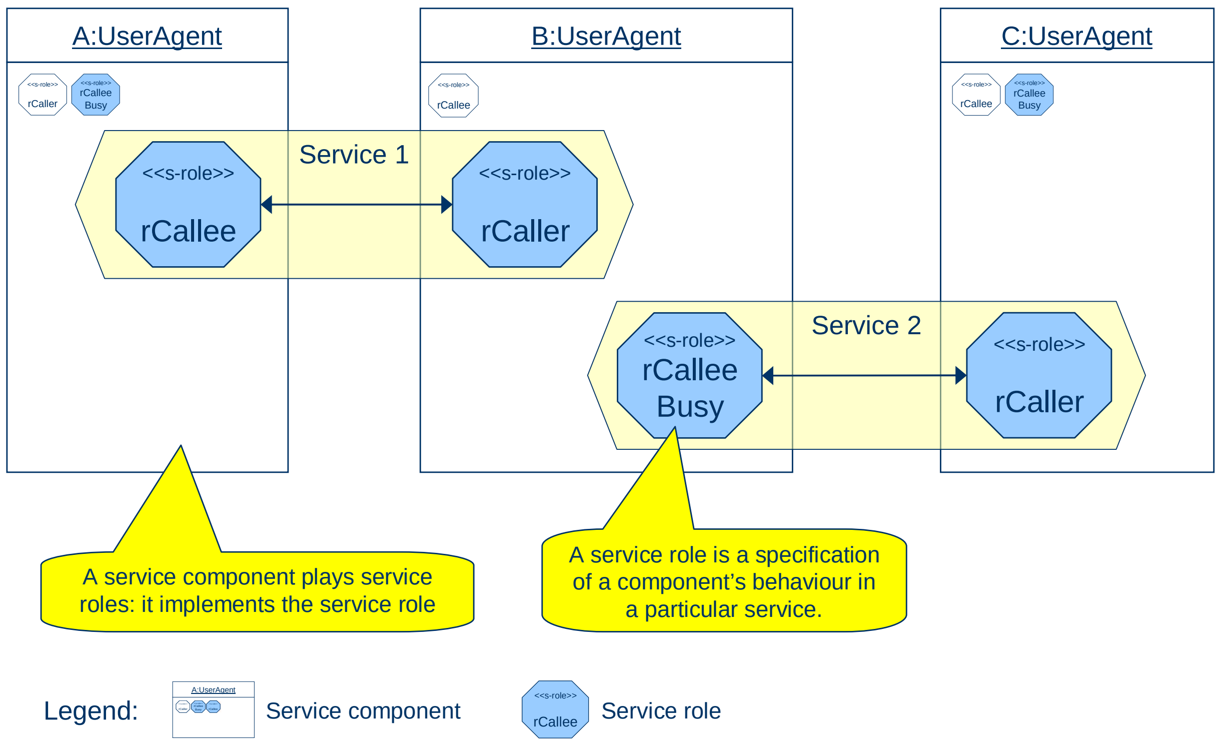


Figure 4: Service components play (implement) service roles

3.3.2 Collaborations and goal sequences

Services are modelled by UML2 collaborations. UML2 collaborations define the cooperation between roles; in SIMS we distinguish between *elementary* and *composite* collaborations.

- Elementary collaborations specify micro-protocols between exactly two parts in a service, called *semantic interfaces*, as well as the goals of the collaboration. Semantic interfaces specify interface behaviour (i.e. how to interact with a service role), while *service goals* (or goals for short) specify the desired outcome of that behaviour.
- Composite collaborations, on the other hand, specify how the service roles will cooperate in the service. Composite collaborations are composed of elementary collaborations. A service role can be bound to a number of semantic interfaces, which each appear as ports on the service role.

Both the elementary and composite collaborations are annotated with ontological descriptions, hence providing descriptions of what the service, collaborations or other entities are about. The ontology provides the basic terminology for a specific domain (such as telecommunications, courier/transportation services, etc.). For instance, it contains a number of concepts of the telecommunication domain relating to services, actions, activities and their attributes. The ontology is not usually changed by the service designers.

The ontological representation of an entity is called an *ontology-driven artefact (ODA)*. An ODA does not specify service behaviour (this is done in UML), but rather describes the purpose of the service, its characteristics and core features, from the end-user perspective. ODAs are created by designers, using the ontology (see D3.4).

Elementary and composite collaborations are illustrated in Figure 5. The composite collaboration *Secretary_TravelReservation*, on top of the figure, specifies a service where the service role *Secretary* interacts with the service roles *Hotel* and a *Plane* in order to plan a travel for his/her *Boss*. The interactions are defined by the elementary collaborations *PlanTravel*, *ReserveHotel*, and *ReservePlane*. The whole service (the composite collaboration) is annotated with an ODA, which describes the purpose of the service, e.g., “a travel reservation service which involves a secretary”[1].

The structure of the elementary collaboration *ReservePlane* is illustrated at the bottom of Figure 5: the intention of that elementary collaboration is to achieve a specific goal, named *SeatReserved.* This goal is specified as an ODA, “Reserve a seat in a plane.”

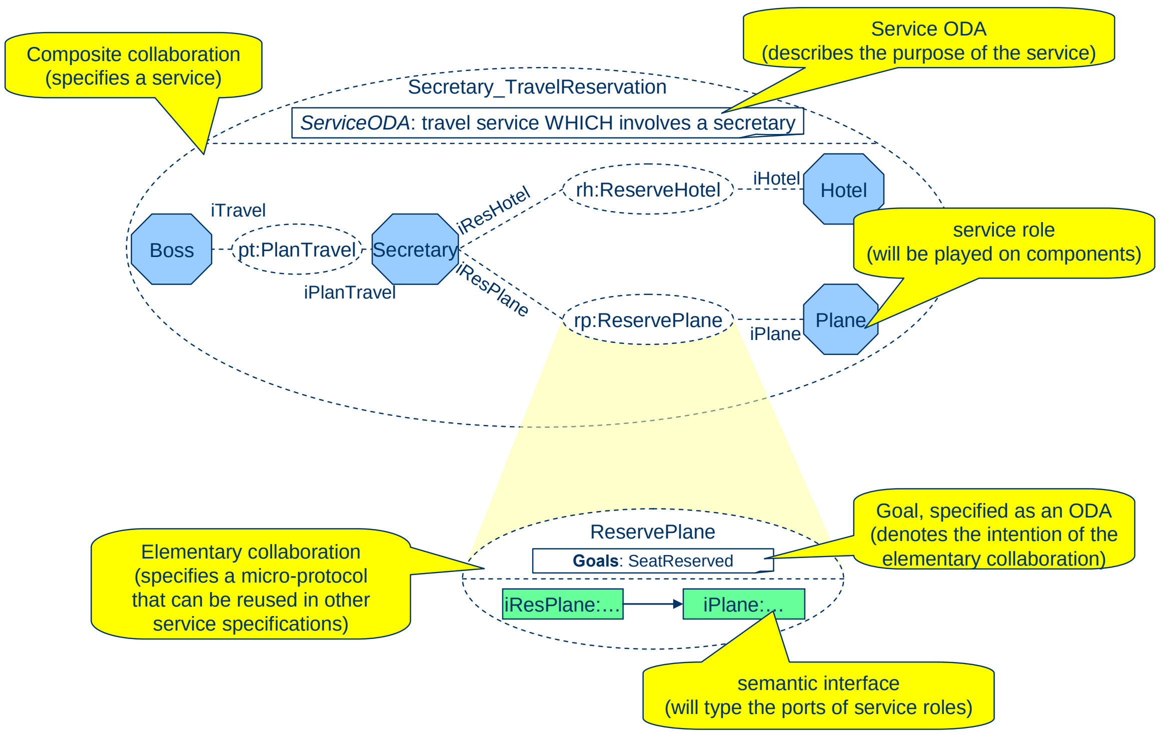
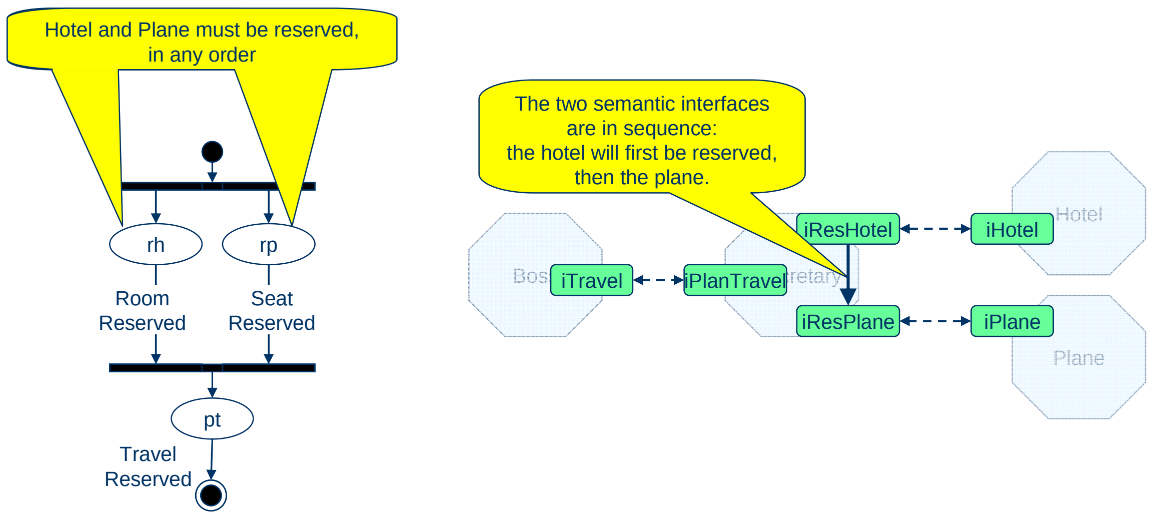

Composite collaboration (specifies a service)
Service ODA (describes the purpose of the service)
Secretary_TravelReservation
ServiceODA: travel service WHICH involves a secretary
Boss
iTravel
pt:PlanTravel
iPlanTravel
Secretary
iResHotel
iResPlane
rh:ReserveHotel
iHotel
Hotel
rp:ReservePlane
iPlane
Plane
service role (will be played on components)
Elementary collaboration (specifies a micro-protocol that can be reused in other service specifications)
ReservePlane
Goals: SeatReserved
iResPlane:...
iPlane:...
Goal, specified as an ODA (denotes the intention of the elementary collaboration)
semantic interface (will type the ports of service roles)

Figure 5: Composite collaboration composed of elementary collaborations

Elementary collaborations specify micro protocols, whose intentions are described by service goals. Composite collaborations, on the other hand, describe the structure of a service; the intention of the service is described by a *goal sequence diagram*. Such diagrams describe the desired sequence of goals of the elementary collaboration. Goal sequences are specified using UML Activity diagrams, as shown in Figure 6a. This goal sequence specifies how the goals of the elementary collaborations in Figure 5 should be ordered within the context of the composite collaboration: in this case a room and a seat must be reserved before the travel itself is completely reserved. I.e. the first two goals *RoomReserved* and *SeatReserved* are free to be achieved in parallel, or in sequence, while the third goal *TravelReserved* should not be achieved before both preceding goals are achieved.

[1] In some examples of ontological representation we will use natural language descriptions (like in the above). For the discussion about various forms of representation of ontology-driven artefacts (ODAs) please refer to deliverable D3.4 *Techniques for ontology-driven semantic interface artefacts*.


Hotel and Plane must be reserved, in any order
rh
rp
Room Reserved
Seat Reserved
pt
Travel Reserved
The two semantic interfaces are in sequence: the hotel will first be reserved, then the plane.
iResHotel
iHotel
Hotel
Bos
iTravel
iPlanTravel
retary
iResPlane
iPlane
Plane

a) Goal sequence b) Semantic interface dependencies

Figure 6: Goal sequences and semantic interface dependencies

Finally, interface dependencies are introduced to ensure that elementary collaborations are composed in a composite collaboration without causing errors. Interface dependencies specify dependencies between semantic interfaces bound to a service role, as shown in Figure 6b.

The distinction between goal sequences and semantic interface dependencies can be summarised as follows: goal sequences specify the intention of the service, by describing how elementary collaborations can achieve something useful. Semantic interface dependencies, on the contrary, specify temporal dependencies between semantic interfaces of a service role, and are used to verify that a composition of service roles is error-free.

3.3.3 Ontology driven artefacts

There are different kinds of ontology-driven artefacts, such as Device ODA, Service ODA or Elementary collaboration goal ODA, which represent different SIMS entities; see D3.4 *Techniques for ontology-driven semantic interface artefacts* for details about the ODA types. For instance, as we have seen in the previous section and illustrated in Figure 5, a composite collaboration can be annotated with an *ontology-driven artefact* (ODA). A *service ODA* provides developers and end-users information about a service using terms that belong to the application area, rather than being described in UML terms.

3.4 Validation of collaborative services

Once a service has been modelled, it can be validated in order to ensure it has no flaws. The following section discusses flaws, while the next section concentrates on the validation steps.

3.4.1 Checking for errors and usefulness

A concern when designing services and service components is that they are error-free and useful. We sometimes refer to error-freeness and usefulness as respectively safety and liveness properties.

Examples of errors are:

- messages sent by one component are not expected by the receiving component, as illustrated in Figure 7a;
- an interaction ends in a deadlock (see Figure 7b, where the three service roles wait for each other);

- a behaviour terminates in an improper way, for instance a component sends a signal to a component which has terminated its behaviour.

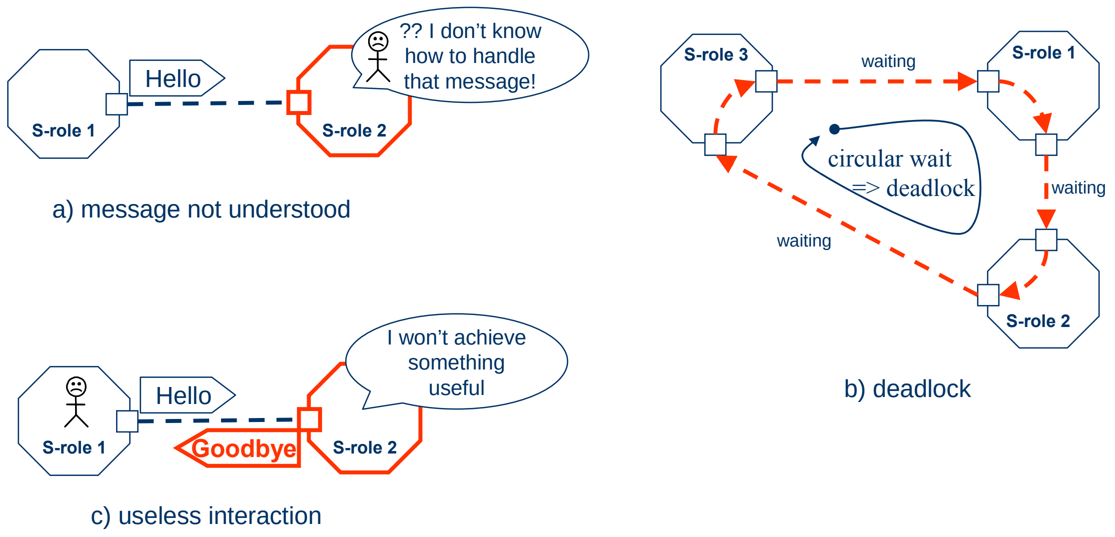


Figure 7: Checking for errors and usefulness

Even though components are error-free when they interact with other components, in some cases they might not be very useful to each other. Such a case is illustrated in Figure 7c, where *S-role2* doesn't achieve anything useful, and seems to terminate the service as soon as it started. In SIMS, we relate usefulness to the achievement of *service goals*.

3.4.2 Validation steps

The validation of the service starts by checking each element alone (i.e. elementary collaboration, service role, semantic interface, etc…). Then the composition of those elements is checked. The different checks performed when validating a service are illustrated in Figure 8.

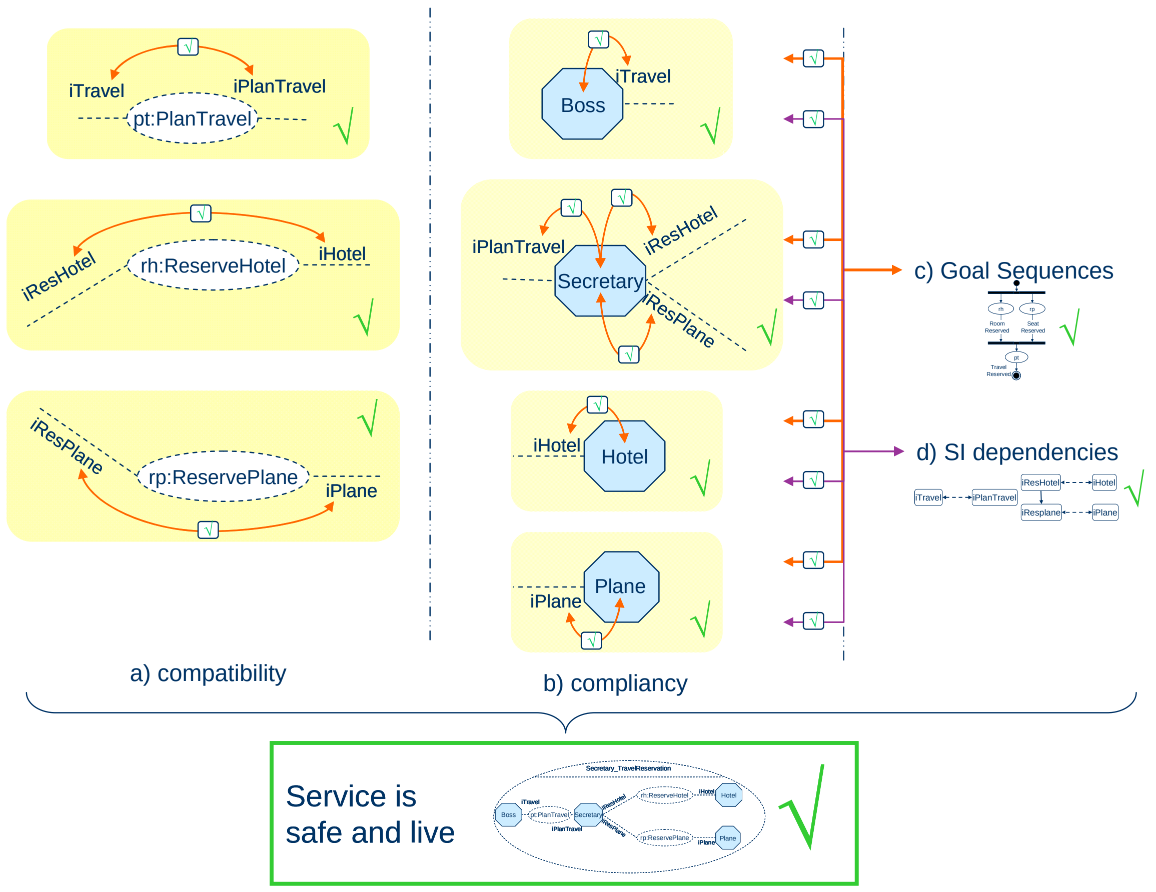


Figure 8: Validation steps

Firstly, one checks that semantic interfaces and goal sequences are *well-formed*. This means they abide some design rules, which allows for a better specification, and avoids unnecessary complexity in the validation algorithms.

After this has been established, one should check that:

a) the semantic interfaces of an elementary collaboration are compatible. This relation makes sure every message sent by one side is received by the other;

b) the service role is compliant with its semantic interfaces. This relation makes sure the service role behaves according to its semantic interfaces (i.e. every message sent or received by the service role is done according to its semantic interface);

c) the service role is consistent with the goal sequence, which means the service role sequences the service goals in the proper order. This ensures liveness properties for the service role and the composed service (i.e. the service role is useful when executing in the service);

d) the service role is consistent with the semantic interface dependencies. This check verifies that sequences of roles that interact are compatible. This ensures that the composition of service roles is safe (i.e. no error will occur).

3.4.3 Validating application-specific aspects described with the ontology

While the UML models allow us to check if service specifications are error-free and can achieve goals, the *ontological descriptions* (or, more, specifically *ontology-driven artefacts*) allow us to specify other properties such as the features of the service, or device capabilities. Once ontology-driven artefacts are specified for various entities (e.g., components, goals), it is then possible to detect ontological inconsistencies, for instance between services and components, service and devices, components and devices, etc.

For instance, suppose a developer wants to validate of the consistency between a service and a device. Assume the device is defined with a Device ODA “A cellular phone, supplied with GRPS, Bluetooth, Java, Symbian OS and HSCSD connectivity”. The service is defined with a Service ODA “An instant messaging service, which requires a device supplied with Java and the Symbian OS”. The validation tools will check if the service and device descriptions match, which is true in this example. This kind of checking is not done (and not possible) by using the UML-part of the service models only.

Ontology-based techniques are also used during the validation process described in point a) of section 3.4.2. The compatibility check will verify, at some point in the algorithm, if the two semantic interfaces achieve the same elementary collaboration goals. By means of ontology-based techniques, the two sets of goals are checked for being identical (identical in the ontological meaning: this is called the *exact match*, ref. D3.4 for details).

3.5 The service marketplace

Generally speaking, software services can be developed without SIMS. But for collaborative services based on truly heterogeneous components SIMS can be of great impact.

For more than a decade we have the paradigm of component-based heterogeneous software. While it seemed promising in the beginning, it failed to fulfil some of the expectations:

- Due to the heterogeneity, components often have incompatible interfaces; the component designer does not know with which other components it will be composed, and could not design the interfaces accordingly.
- There are few marketplaces in which domain-oriented components can be traded.

In SIMS we addressed the first challenge by introducing semantic interfaces and the accompanying validation techniques described in the sections above. The second is addressed by the mechanisms of the service marketplace.

In this section we present the vision of the service marketplace. Here we describe a service “ecosystem”, defining the roles of various stakeholders. The implementation of such a marketplace lay outside the scope of the SIMS project; an example of future planned work on the service marketplace is that of the Celtic project *Servery*. However, the SIMS project has investigated what mechanisms should be found in a service marketplace, because we consider that there lies a great potential in exploiting the concepts of SIMS in such a marketplace:

1. SIMS separates between service specifications (service structures with semantic interfaces) and service components;
2. SIMS validates the compliancy of service components to the specifications;
3. SIMS enriches service specifications and service components with standardized terms for a domain using ontological techniques; the full potential of this is taken out in settings with many components and specifications, for instance in a largely populated marketplace;
4. SIMS middleware provides support for End users and devices learning new service behaviour, what we for marketing reasons have called *peer-triggered self-update*[2]; this can introduce new business opportunities for 3rd Party software providers in a well-functioning service marketplace.

3.5.1 The vision of a service marketplace

SIMS envisions the existence of a well-functioning service marketplace.

[2] “Design driven composition” is the term used in D2.2 and other deliverables

The service marketplace is an ecosystem where service authors, component providers and service providers meet to publish and find information related to services and service components for the benefit of themselves and of the end users.

Key elements of a marketplace are the existence of a repository of service specifications[3] and a repository of service components[4] that comply with service specifications, see Figure 9.

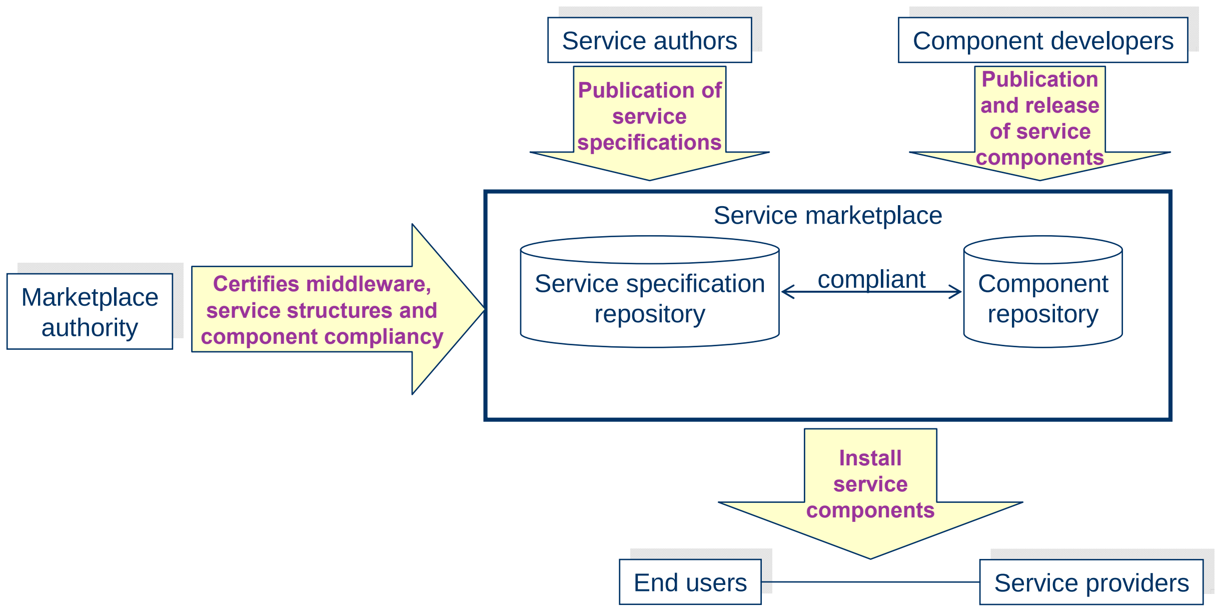

Service authors
Component developers
Publication of service specifications
Publication and release of service components
Service marketplace
Service specification repository
compliant
Component repository
Marketplace authority
Certifies middleware, service structures and component compliancy
Install service components
End users
Service providers

Figure 9: Players in the service marketplace

The key players in a marketplace are first and foremost the various stakeholders:

- *Service authors* that provide new service specifications, possibly based on existing specifications;
- *Component developers* that develop service components that comply with service specifications;
- *Service providers* that provide support for specified services for the benefit of their end users; and
- *End users* that install and use service components in order to collaborate in specified services.

An additional key element for the well-functioning of a marketplace is the existence of a governing body which we have chosen to call the *Marketplace Authority*; this organisation acts as a trusted authority to ensure the integrity of the marketplace for the benefit of all players.

3.5.2 Deployment of service components via a marketplace

In the term *deployment* we include the publication, release and installation of service components:

- *Publication* of component specifications: Component developers use the service marketplace to make the released service components public, so that potential Service providers and End users can retrieve information about them;
- *Release* of implementations of components: stores installable code in the repository from where they can be downloaded;

[3] By service specifications we mean descriptions of service structures (composite collaborations composed of elementary collaborations); this includes the behaviour of semantic interfaces with goals, interface dependencies and goal sequences. The internal behaviour of service roles and/or service components is not part of the service specification.
[4] Service components are described by component types, and come with an implementation (installable code).

- *Installation*: downloading service components to the devices where they can be instantiated and run.

Related to this is the publication of service specifications which the components fulfil.

Publication, release and installation using the service marketplace are shown in Figure 10 below, where flows of service and component specifications (dashed arrows) and component implementations (solid arrows) are depicted[5].

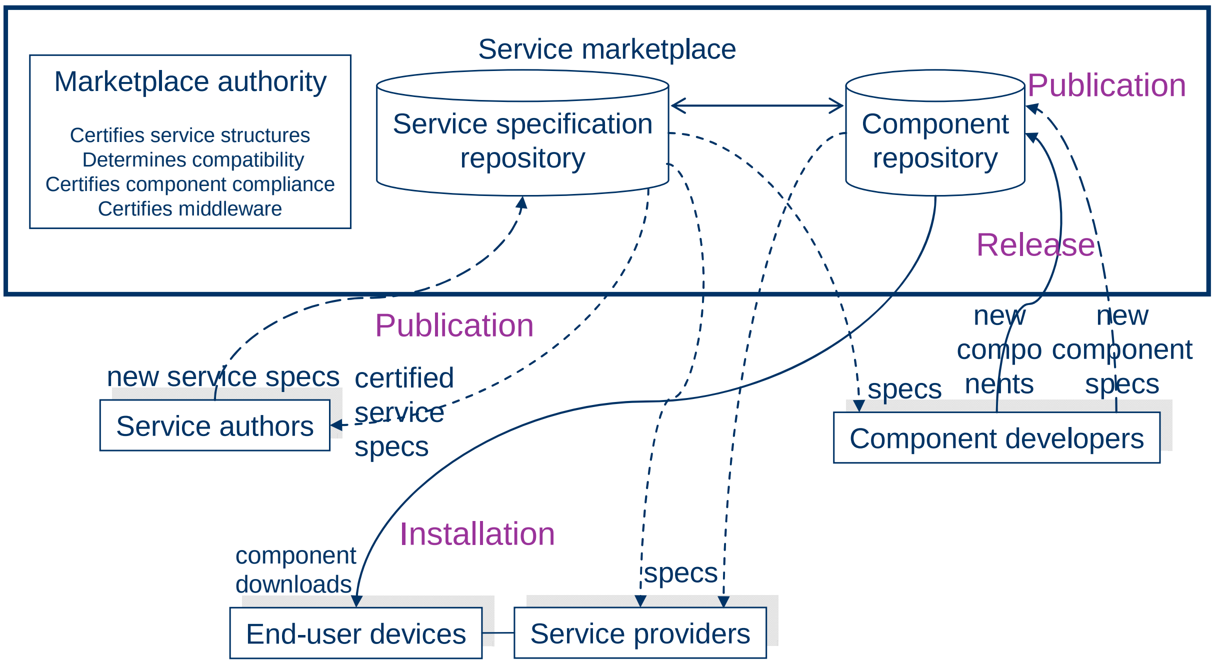


Figure 10: Publication, release and installation via a marketplace

Each of these types of activities is discussed in the subsections of section 3.5.5 and in D5.5.

3.5.3 Elements of a business model

Before going deeper into the technical solutions envisioned, a few words should be said concerning the viability of a service marketplace. It should be sustainable in a market economy where the individual stakeholders participate on their own free will. The following elements are relevant to the business model for the service marketplace and stakeholders described above:

- The existence of a centralized repository of service specifications means that Service authors have a one stop place to publish their service specifications. They can use pre-published service specifications in their creative work, and can decide what parts to share in a marketplace, and what to keep confidential for business reasons.
- Service authors who are also Service providers and/or Component developers can publish the parts where they desire the marketplace to provide competitive contributions, such as components that operate on new terminal types and that are compatible to service specifications that they already use in their business.
- A central repository of service components means that Component developers have a one stop place to publish their components. Openness and fairness in competition is ensured by the Marketplace Authority.[6]

[5] The figure shows that components may be installed from the marketplace or from component developers; flexibility in storing of component implementations is discussed later.

- The existence of a centralized repository of service specifications and service components enables the collection of usage data from Service providers (on the behalf of End users). Usage can include component installation and/or service instantiation (initiation of service sessions involving a component), thus monitoring both the use of services as well as service components. Usage data can also contain reports of quality problems, so that components can be attributed with such data. Privacy issues arise in this context; such data should be protected from un-authorized access.
- If only Service providers that are members can gain access to the published service components, then this will motivate their membership. Service providers could pay for membership in the Marketplace Authority, or waive their fee by collecting and supplying usage data.
- Usage data[7] from Service providers can be the basis of price calculations, and flexible pricing strategies can be implemented. For instance, free installation up to a certain number of End users; charging for the use of services, and combinations of these. Bulk negotiations can take place between large Service providers and large component providers; however, transparency is desired.
- End users have access to "latest and greatest" services, and can learn of new service opportunities from their communicating peers – what we have coined *peer-triggered self-update*.

As mentioned, SIMS did not implement a marketplace, but the principles upon which the project is based can contribute to the creation of a sustainable marketplace for the benefit of all stakeholders.

3.5.4 Marketplaces are domain specific

We do not envision a single, all-encompassing marketplace. Rather there can be many marketplaces: one for each domain – or even several "competing" marketplaces within a domain; some can be driven by serious players with rigorous quality checks and membership schemes, others can be "open source" style marketplaces with fewer tasks performed by the Marketplace authority and no membership fees.

Some marketplaces can be closed, e.g. internal to a single company or open only to partners with tight economic and organizational commitments; open marketplaces may involve independent actors with vested economic interests regulated by a governing body, while other again may be open source "best effort" marketplaces without any restrictions.

The economic models of marketplaces will depend on the openness and also on the mechanisms supported by the middleware that run the services and service components. In the case of an open marketplace with certified middleware support, flexible charging mechanisms can be put into place, such as payment based on service use (payment per service session), component installation or a combination. A variety of payment models can be supported.

SIMS provides technology that enables the technical functions of service marketplaces, such as compatibility validation algorithms and tools, ontology mechanisms, proof-of-concept ontologies, prototype design tools and prototype middleware support. These are outlined in the following sections.

[6] The Marketplace Authority does not rule out trade secrets being shared between players outside a marketplace; e.g. Component developers may exchange private components that fulfil both published and private service specifications without breeching the rules of the marketplace.
[7] The availability of usage data to the Marketplace Authority depends on where a component for download lies: if it is downloaded via the Marketplace Authority, e.g. from a central repository, then the Authority can gain usage info directly. Differently, if the Marketplace Authority only publishes a directory and does not support downloads of components, then this saves the Authority from solving licensing issues; this can be delegated to negotiations between the Component developers and the Service provider or End user.

3.5.5 Defining common terms in a service marketplace using ontologies

The proper working of a service marketplace requires standardisation of the terminology of the domain. This is where ontology techniques come in, see Figure 11.

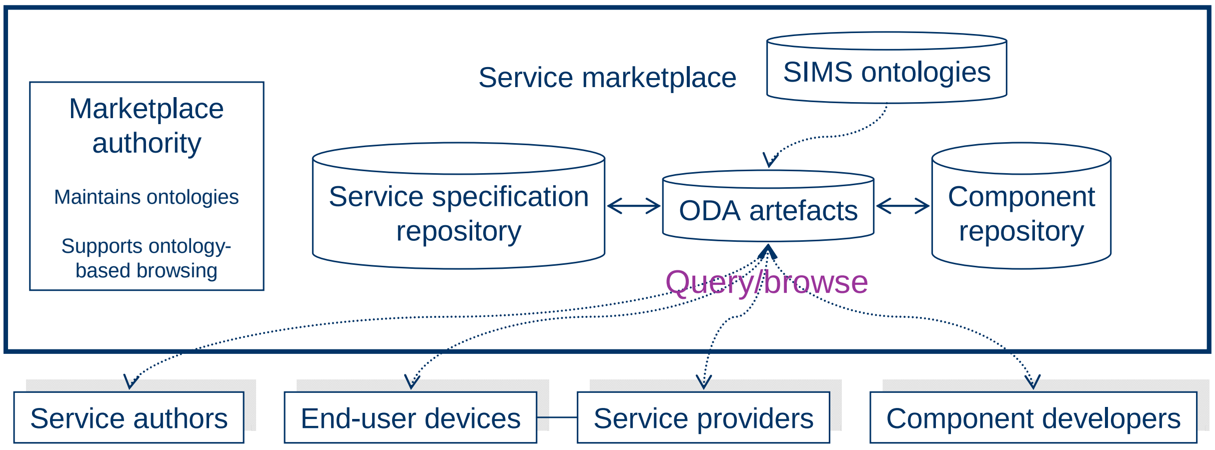


Figure 11: Use of Ontology derived artefacts in a marketplace

Here the use and maintenance of SIMS ontology and ontology driven-artefacts (ODAs) is indicated by dotted arrows[8]. The basic terminology is defined in the SIMS ontology, while the services and components are described as ODAs in the ODA artefacts repository. Every single ODA refers to a single service or a component specification. The ontology use cases described in D3.4, such as "Querying for services," "Subscription for services," or "Find components which realise a specific service" make use of information published in a Service marketplace.

3.5.6 Integrating a marketplace with design & validation tools and middleware

In this section we show the information flow between users of the SIMS design and validation tools, the service marketplace and installations of the SIMS middleware. The presentation is structured according to the principal activities of the stakeholders: service authoring, component development, service provisioning and installation. The involvement of the Marketplace Authority in each is discussed.

3.5.6.1 Service authoring

Service authoring is about creating new service specifications, and publishing them in a service marketplace. The service specification repository plays a key role in this picture, see Figure 12.

[8] The ontology is maintained by the Marketplace authority; Service authors may ask for changes to the ontology.

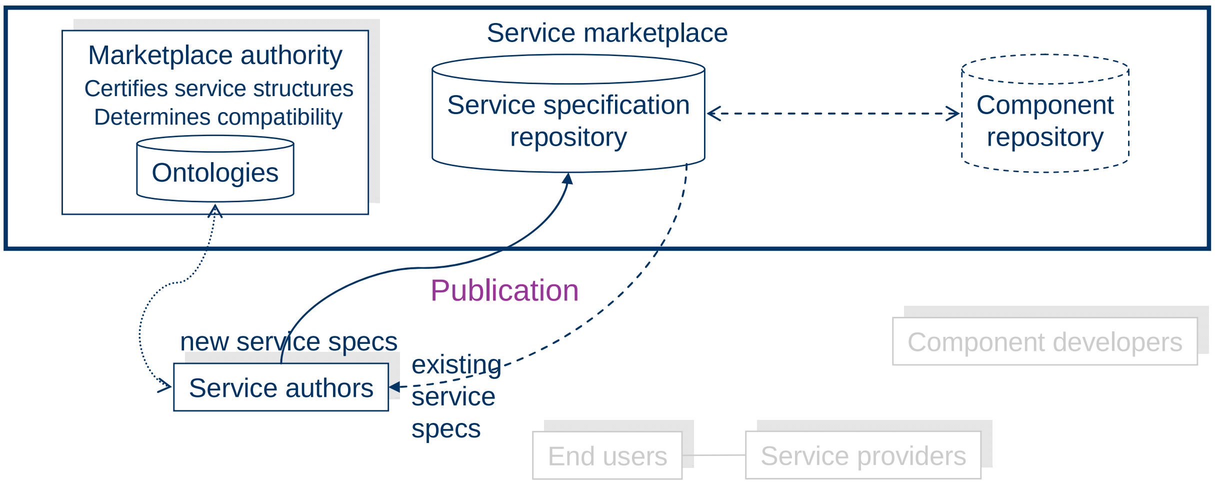


Figure 12: Publication of service specifications via a marketplace

New service specifications are developed by Service authors, possibly based on existing specifications found in the service marketplace. Service authors use design and validation tools in this work, as described below.

Specifications are submitted to the Marketplace Authority for publication. These specifications are certified by the Marketplace Authority before being made available in the marketplace. Once available, they may be used by Component developers, Service providers and other Service authors.

The development work on service specifications is carried out using the design and validation tools specified in SIMS. The use of tools by Service authors for authoring, and the use of tools by the Marketplace Authority for certification, are both depicted in Figure 13.

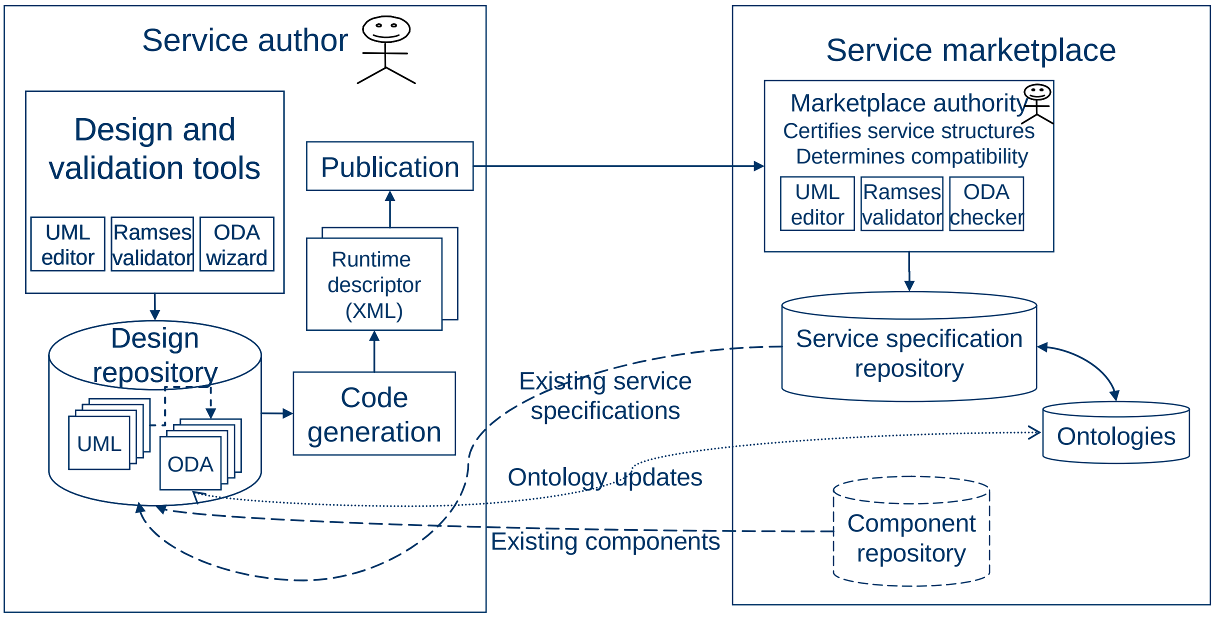


Figure 13: Use of tools for service authoring and publication

- **Service authors** use the SIMS tools on their own internal repository of service components and service specifications in their service design work. In addition they can also use pre-existing service specifications found in the service specification repository, as well as appropriate service components from the marketplace (as indicated in Figure 13 by dashed arrows)[9]. Enriching service specifications with Ontology Derived Artefacts is part of this work, and implies that Service authors have access to the ontologies maintained by the Marketplace Authority. See D2.3 for a scenario that exemplifies this.

- The **Marketplace Authority** uses the SIMS tools to certify the submitted service specifications. They certify that the specifications are consistent and well-formed; this includes checking that:
  - submitted specifications do not exist from before;
  - new elementary collaborations have semantic interfaces that are dual;
  - semantic interfaces contain goals;
  - goal expressions using Ontology Derived Artefacts (ODAs) are present and consistent with the ontologies maintained by the Marketplace Authority;
  - composite collaborations use published elementary collaborations, and that
  - goal sequences are consistent and achievable.
- In addition the Marketplace Authority searches for compatibility between submitted semantic interfaces and pre-existing semantic interfaces, and populates the service specification repository accordingly.

Submitting new service specifications may entail requests for alterations to the ontologies. This is negotiated between Service authors and the Marketplace Authority. We envision a lower rate of change to the ontologies compared to the component repository; service components will come and go as a result of technology changes, while ontologies reflect the nature of a problem domain, and change very slowly[10].

3.5.6.2 Component development

Component development is about developing new service components and publishing and/or releasing them in the service marketplace, see Figure 14.

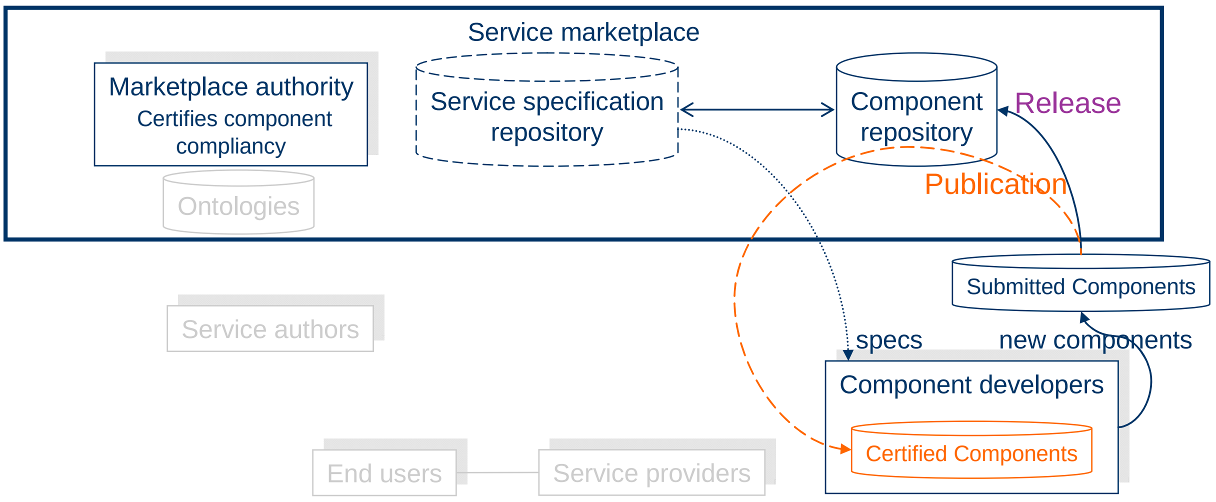


Figure 14: Publication and release of service components via a marketplace

[9] Recall that it is necessary for the Service author to have service components in order to run services, which may be needed in order to ascertain that the service specifications are correct; using validation tools alone is not enough. For instance certain forms of deadlocks are not found by the SIMS validation tools.
[10] Note that terms and relationships in an ontology cannot be removed without causing problems for artefacts that use them.

New service components are developed by Component developers based on existing service specifications found in the service marketplace, as indicated by a dotted arrow in Figure 14. Service authors use design and validation tools in this work, as described below. Component specifications are submitted to the Marketplace Authority for publication, and/or service components are submitted for release. These specifications and/or service components are certified by the Marketplace Authority before being made available in the marketplace. Once available, they may be used by Service providers and Service authors, and the service components can be installed by End users, see section 3.5.6.4 below.

Note that the release and publication of service components are intentionally separated in order to provide flexibility. For example, service components may be published in a marketplace and released on (installed from) one or several other sites. This is indicated in Figure 14 by the certified service components being available from the site of the Component developer, not just from the Component repository in the Service marketplace.

The development work on service specifications is carried out using the design and validation tools specified in SIMS, as well as other appropriate tools, as depicted in Figure 15 below.

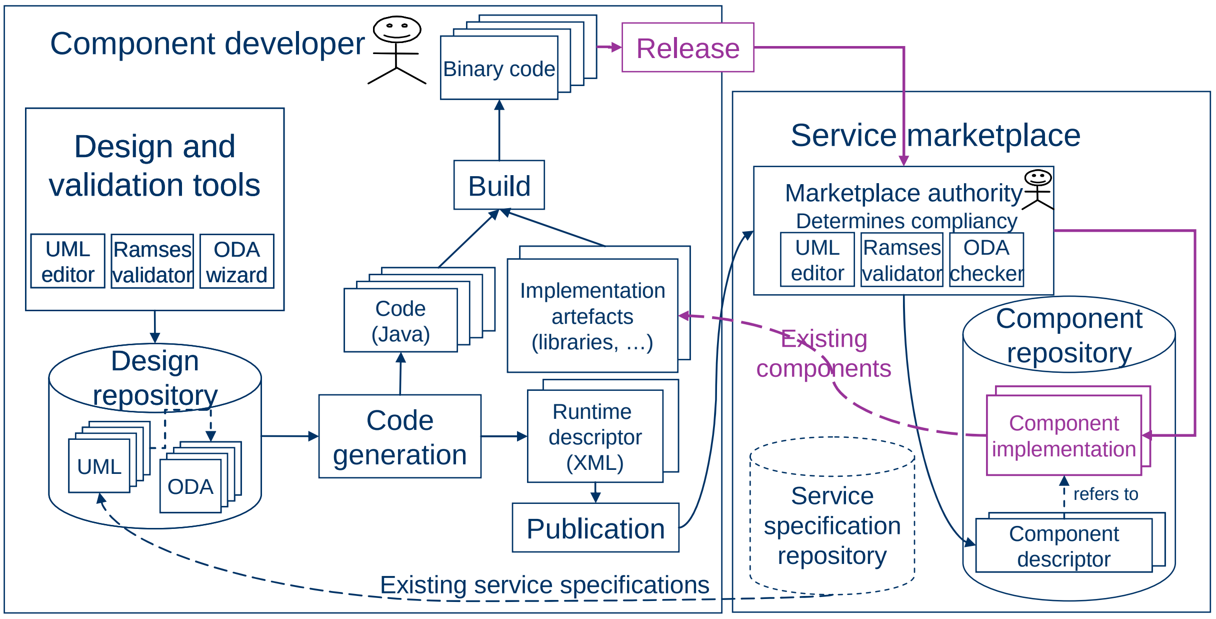


Figure 15: Use of tools for service component development

- Component developers use the SIMS tools on their own internal repository of service components and service specifications in their design work. In addition they can use pre-existing service specifications found in the service specification repository, as well as appropriate service components from the marketplace (as indicated in Figure 15 by dashed arrows). Service components are used by the Component developers to build running services, in order to ascertain that the design is correct. Component specifications (descriptors) are submitted to the Marketplace Authority for publication in the marketplace, as well as the released component implementations.
- The **Marketplace Authority** uses the SIMS tools to certify the submitted component specifications and the component types implementing them. They certify that they are compliant with the specifications; this includes:
  - checking that submitted components that exist from before are intentional replacements;
  - checking that new components are compliant with their specified semantic interfaces;
  - checking that goals and goal sequences are achievable, and
  - checking that components include adequate deployment information.

We envision a higher rate of change to the component repository compared to the service specification repository; components will come and go as a result of technology changes, while service specifications change slowly. A potentially large number of components may be submitted that comply with the same service specifications, e.g. implementations of the same service functionality on a number of devices and platforms.

More details concerning component release are described in D5.5.

3.5.6.3 Service provisioning

Service provisioning is about Service providers supporting the mechanisms required by their End users, and following the provisions of the service marketplace. For an Service provider to be granted access to the service marketplace there needs to be a certification process to ascertain that the Service provider can be trusted, see Figure 16.

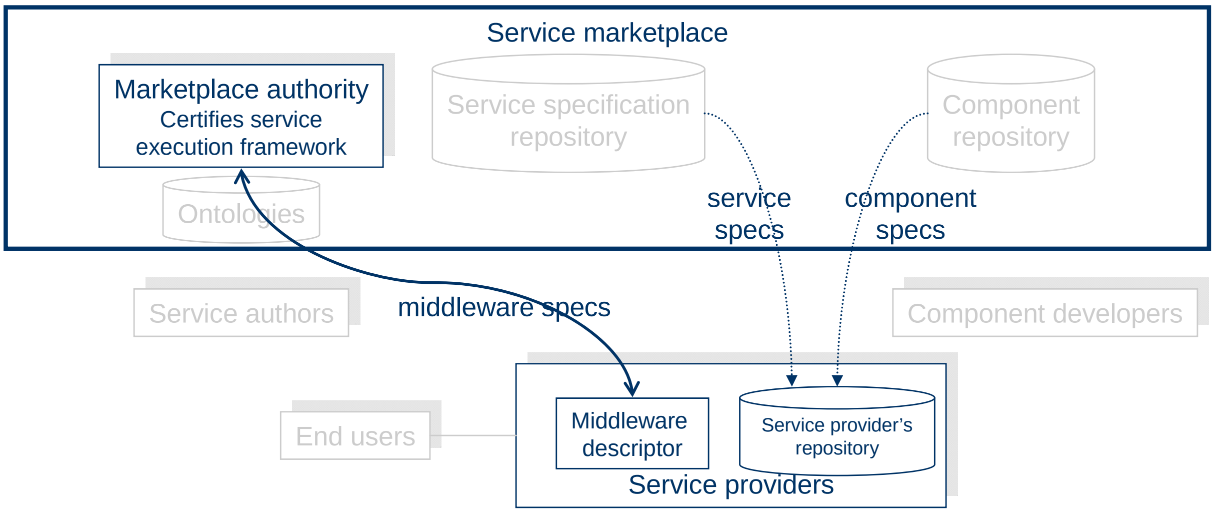


Figure 16: Certification of Service providers for a marketplace

Note that there can be a number of service providers operating in the service marketplace, each with its set of End users. Each service provider supports a service execution framework (the middleware support) that adheres to the rules of the service marketplace. One of the tasks of the Marketplace Authority is to certify the service execution framework offered by a Service provider. This includes:

- Certifying the component installation mechanisms so that correct information about component installation is available in the marketplace;
- Certifying the service discovery mechanisms across Service provider boundaries so that mechanisms work without disclosing sensitive information about End users;
- Certifying the usage data collection mechanisms so that correct information about component and service use is available in the marketplace, including reports of quality problems.

The technical details of the certification process are not described by SIMS; Figure 16 indicates that there exists a description of the middleware that can be certified. The Marketplace Authority could also provide a standard execution platform to Service providers, either as an offered or a required framework.

Certified Service providers gain access to the published specifications and the released components of the marketplace, as indicated by dotted arrows in Figure 16.

Figure 16 does not show details like the Service providers reporting Usage info to the Marketplace Authority. The Marketplace Authority can maintain a database for usage info such as component installation, service invocation, and error reports.

3.5.6.4 Component installation

Installation is concerned with making service components available for use by End users on End user devices, see Figure 17.

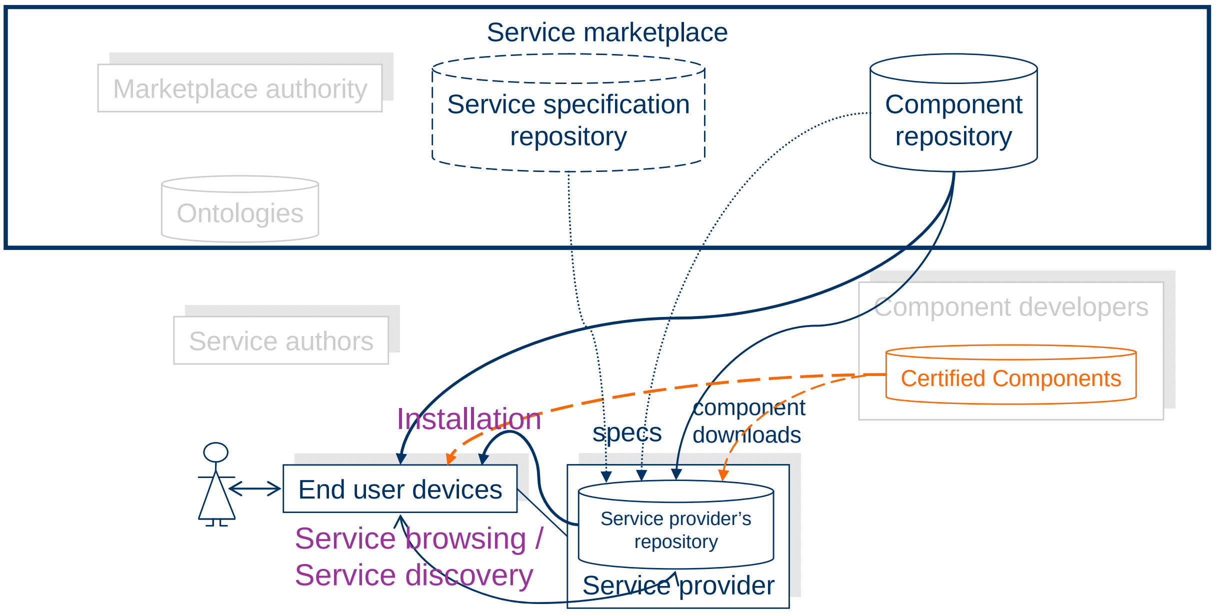


Figure 17: Component installation via a marketplace

The smallest entity that can be installed is the service component, which comes with its logic and its service roles. Components are installed on End user devices either due to pre-installation determined by the Service provider, due to service discovery initiated by the End user, or due to Service browsing.

The Marketplace Authority is indirectly involved in the installation cycle; as mentioned earlier, it certifies that the developed components are indeed compliant with the service specifications, so End users and Service providers can trust that installed components have the right quality.

The Service provider is also indirectly involved in the installation cycle in Service browsing and Service discovery. It exploits service specifications and component specifications obtained from the service marketplace, as indicated by dotted arrows in Figure 17.

It is an open issue if certified components available for installation are stored in the central component repository - indicated by solid arrows in Figure 17, and/or hosted by other sites, such as at the Component developers site - indicated by dashed arrows in Figure 17; in the latter case the component repository holds published component specifications and not released service components.

It is also an open question if service components are downloaded to the End user devices directly from the component repository, and/or installed via their Service provider. In the latter case the Service provider caches service components on behalf of the End users, according to marketplace agreements.

Settling these issues has implications on the business model, see D5.5 *Overall SIMS architecture*. More details concerning component installation are also described in D5.5.

3.6 SIMS support at runtime

One of the main goals of SIMS was to provide runtime support for *service discovery* and *service instantiation* with compatibility guarantees.

We distinguish between the middleware and service components:

- The middleware provides generic mechanisms for component deployment, component instantiation and communication between distributed components. It also provides mechanisms for service discovery and service composition.

- Services result from the interaction between distributed service components. Service components may play different service roles in a service. For example in response to a request during a call setup, a component may either play a "Callee" role or a "CallTransfer" role. The component logic decides which service role is instantiated. The decision may depend on the user context (e.g. the user may be busy) or on the availability of resources on a device.

The overall architecture of SIMS middleware is presented in Figure 18.

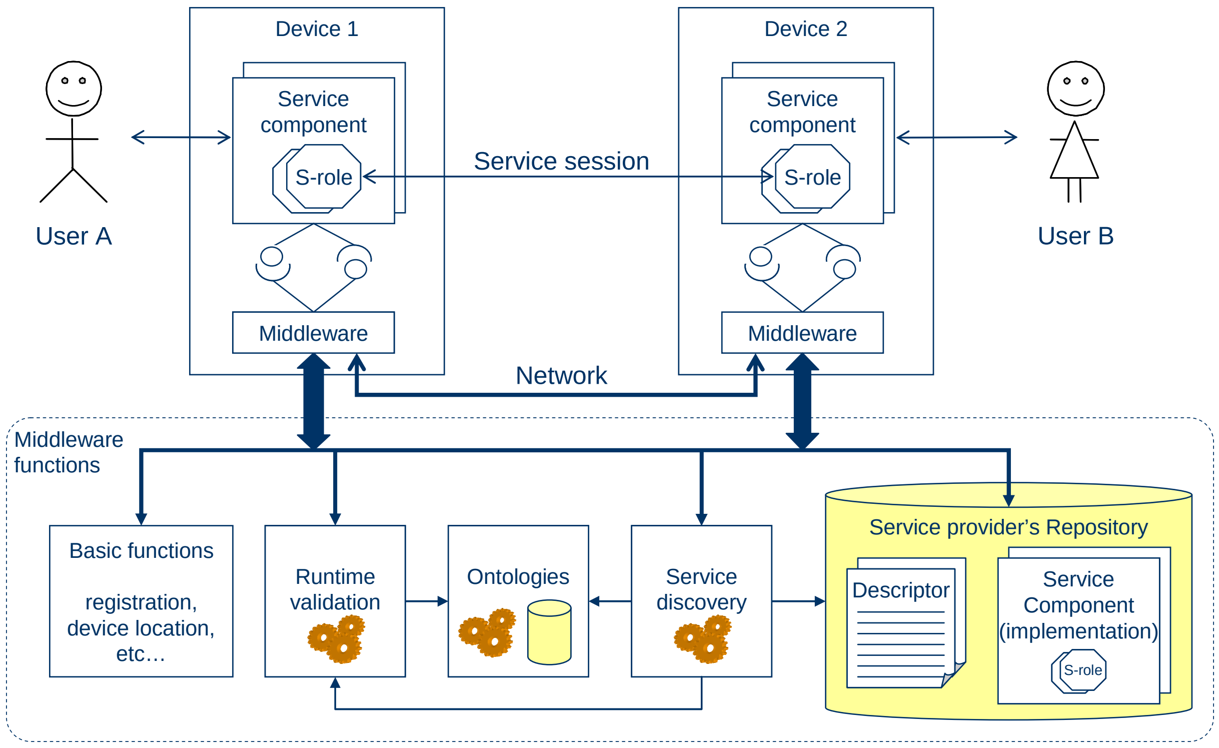


Figure 18: SIMS runtime architecture

The figure above emphasizes logical functions, and does not prescribe any particular placement of software or data. In particular it does not dictate a distributed or centralized solution to the middleware functions. That is why a bus-like notation is used. The service provider's repository of service components does not preclude components being sources elsewhere, compare with Figure 17.

Figure 18 shows two End users, each using a device[11]. Devices connect to each other through the middleware[12], and have access to the middleware functions presented at the bottom of the figure. Devices can contain several components, each of them playing different service roles as explained in D2.3. A component uses functions provided by the middleware (such as registration, service discovery and role request), and provides certain functions required by the middleware (such as up-bound role requests). Components interact with each other in service sessions established by middleware (role binding), ultimately making use of a communication network.

There may be one central middleware server or multiple distributed servers hosted by the service provider; however, this need not be known by the service components. Some middleware functions are initiated by the components themselves (e.g. discovery of compatible players), while some are initiated by the middleware (e.g. discovery of service opportunities).

[11] Devices do not need to have End users; some may act as servers without any End users directly connected.
[12] The SIMS approach and middleware allows devices to exchange data without using the middleware, e.g. for efficient media streaming; however, in SIMS all control of service sessions is provided by the middleware.

Figure 18 does not show the presence of multiple Service providers. The mechanisms for operating across Service provider boundaries are for future work, and are not detailed here.

In the following section we present the overall functions of the middleware.

3.6.1 Overview of middleware mechanisms

The different functions (or modules) provided by the middleware are as follow:

- Basic functions: functionality one usually finds in a middleware architecture, such as registration of devices/components, component/device/user location and addressing, etc… Those functions are detailed in D5.5;
- Runtime validation: used to check compatibility or subtype between two semantic interfaces, or soundness of a composition. Validation makes use of ontologies (i.e. check that two goals are compatible, for instance), and is used during service discovery, as well as by devices. The different kinds of validation are explained in D2.3;
- Ontologies: contains the Service provider’s copy of the ontology repository, as well as different reasoning functions associated to it. Note the devices do not have access to the ontology module, as they are used only during validation and service discovery. The different uses for ontologies are described in D2.3, while the ontological techniques developed at runtime are detailed in D3.4;
- Service discovery: provides functionality to discover compatible components, discover service opportunities, and perform role learning. It makes use of the repository (to get information about service components) and the validation module (in order to check if service components are compatible). The ontology helps the service discovery to filter its search among the specifications of service components; this filtering functionality is a way of identifying relevant service components among a large set of component. Service discovery is detailed in D5.5;
- The Service provider’s Repository, which stores service specifications and component specifications from the service marketplace, as mentioned in section 3.5.6.3. The repository is used by the devices (so they can discover and download new service components, or update old ones), and by service discovery (which accesses the specifications of the service components).

4 Who can benefit from SIMS

Below we list promising candidates for application areas that could benefit from SIMS in addition to the telecoms domain that lies at the heart of SIMS. Three areas are identified and discussed, and one was selected for pilot application development.

4.1 Mobile social networking

The concept of peer-triggered self-update is in fact already used in the Facebook application ecosystem. When a user wants to interact with another user with a new Facebook application, he will first send an "invitation to download the app" to the other user, before they can start interacting. The update process is fairly simple since the applications are Ajax/html based applications. However, an increasing number of applications are using more advanced components. For example the application "speed date" (http://apps.facebook.com/speeddatefb/index.php?page=speeddate) combines a chat, 2 way video link and search functions. A similar application deployed on a mobile version of Facebook (currently not available) would be an excellent implementation scenario for SIMS.

Similar functionality was implemented in one of the pilot services (to be specific: the service involving vehicle drivers and pick-up/delivery sites).

4.2 Social proximity applications

Social Proximity Applications (SPAs) are a promising new area that exploits everyday changes in the proximity of mobile users.

Software components and usage histories are exchanged between mobile users who are in proximity of each other. The Domino framework (http://www.equator.ac.uk/index.php/articles/c117/) applies this approach to a mobile strategy game in which players adapt and upgrade their game using components from other players, progressing through the game by sharing tools and history. Networks like www.aka-aki.com , http://brightkite.com , http://meetmoi.com , http://zyb.com/ also match users depending on their locations, profiles and what kind of interaction they are looking for. By integrating SIMS, these networks could improve their offer by enabling more interaction between users and easier spreading of applications.

4.3 Mobile collaborative software

Mobile Collaborative Software is another area of interest. Software start-ups and EU projects focusing on collaborative software could be surveyed. For example *inContext* (http://www.in-context.eu) is a FP6 project aimed at developing a novel scientific approach to the problem of enabling diverse individual knowledge workers in separate organisations to work in effective team collaboration with one another. The end result of the research has emphasised P2P collaboration software capabilities. Another example is Jive Software (http://www.jivesoftware.com/products).

5 Highlights of Achievements

The SIMS project has achieved the following key elements:

- Design-time methods and tools for collaborative service engineering that make it easier for developers to develop correct software and services.
- Runtime support for peer-triggered self-update of collaborative services that makes it easier for users to find and use useful services.
- Mechanisms of a marketplace for collaborative services that make it easier for service providers to establish a marketplace for service components.

Below is a description of these elements and what they include, followed by an account of the project's contribution to standards and Open Source software.

5.1 Methods and tools for designing collaborative services

Technical advances by SIMS in this area are closest to being mature, and include:

- A method for compositional service engineering exploiting new features of UML 2.
- Validation algorithms and tools at design time and runtime that can guarantee compatibility between collaborating service components.
- Methods and tools for enriching design models with semantic information (ontologies).

5.2 Peer-triggered self-update of collaborative services

Middleware support for peer-triggered self-update[13] is new. Technical advances by SIMS in this area include:

- Lightweight semantic browsing for services from mobile devices.
- Service role learning at runtime.
- Service composition at runtime with compatibility guarantees.

5.3 Marketplace for service components

The tools and techniques of SIMS can support a service marketplace. Technical advances by SIMS in this area are visionary more than technically feasible today, and include:

- A framework for a new business model, allowing composite services with components from multiple vendors.
- Links between services and components using semantic information (ontologies).
- A support for the deployment of new models for software services: for instance services can be charged by operators when their usage reaches a certain threshold.

Key elements of a marketplace are the existence of a repository of service specifications and a repository of service components that comply with service specifications, see Figure 9.

[13] The concept of *peer-triggered self-update* is chosen for marketing reasons. In the technical work this is referred to more precisely as *design-driven composition* and *role learning*.

5.4 Contributions to standards and Open Source

Below we list the SIMS contributions to standards and to Open Source software.

5.4.1 Standards for modelling language (UML)

SIMS has contributed to the UML Profile and Metamodel for Services (UPMS), *SoaML*. This was a response to an RFP from the OMG. It was SINTEF that undertook this responsibility in the project.

The contribution aimed to ensure that the SIMS purposes are covered by the emerging UPMS standard. Most important here are the SIMS modelling concepts, including the principles of semantic interfaces, composite collaborations and goals (in particular progress labels – called *Milestones* in SoaML).

Details of the standardization process and the results are found in D7.5 *Standardization result*.

5.4.2 Contributions to Open Source software and ontologies

The following tool components are or will be released as Open Source software. In addition the middleware component ActorFrame that was used by SIMS is being made Open Source by Tellu[14].

5.4.2.1 Ramses

Ramses is an Eclipse plug-in developed by NTNU. It is available under the Eclipse Public Plug-in Licence. Most algorithms that were developed are based on state machine transformation and validation. The algorithms were implemented with the constraint that very little should be changed in the code, so developers can then adapt the source code to suit their needs, with little effort.

Moreover, during the second phase of the project, extension points were introduced in the algorithms. This allowed us a smooth integration with ontologies. Most important, it allows third party developers to add their own extensions, in order to modify the algorithms without actually reading, understanding and modifying the source code of Ramses. Extension points also provide an easy parameterisation of the algorithms (for instance using Booleans to bypass some parts of an algorithm, or ignore an extension point in the algorithms).

5.4.2.2 Artefact Wizard

Artefact Wizard is an open-source, Eclipse-based tool for creating and manipulating of ontology-based descriptions (so-called *ontology-driven artefacts* or ODAs) developed by WUT. It is published under the Eclipse Public Licence.

Artefact Wizard comes in two versions (both published as open source). The first version is an integral part of the SIMS design time tools suite and is accessible from the SIMS project site.

The second version is a standalone tool. This version was prepared because of the Artefact Wizard's high adaptability to various kinds of artefacts, which can be used for annotating software modules with ODAs even outside SIMS. This version is available for download from the WUT's Mobile and Embedded Applications Group's site, see section chapter 8.

5.4.2.3 ODA Web Wizard

ODA Web Wizard is a lightweight semantic browsing tool, which enables searching for services from mobile devices. It has been developed by WUT and integrated with the SIMS middleware.

[14] www.tellu.no

The tool has been successfully tested in SIMS with the following mobile devices: Nokia N800, Nokia N810 and Fujitsu Siemens Pocket LOOX N560.

ODA Web Wizard can be also treated as a web-based tool for creation and manipulation of various ontology-based descriptions – even outside the SIMS environment.

The tool is available as open source and is accessible from the WUT's Mobile and Embedded Applications Group's site.

5.4.2.4 Ontology of telecommunication services

The ontology of telecommunication services for SIMS, made by WUT, is a contribution in defining the telecommunication service semantics.

The ontology has been validated within the SIMS proof-of-concepts applications. A number of examples of ontology-driven artefacts (ODAs) exploiting the ontology were produced, accompanying the ontology.

The ontology has a component-based structure, which makes the maintenance simpler. However, the most important reason for such separation of concepts was the potential for the ontology reusability (separate modules can be reused if needed since most of them are independent).

The ontology is available as open source, and is accessible from the WUT's Mobile and Embedded Applications Group's site. Complete documentation for the ontology is available for download there in form of the SIMS public deliverable D3.5 "A proof-of-concept ontology of telecommunication services".

6 Analysis of the results

In this chapter we present our own analysis of the results achieved in the project. We identify competitors in section 6.1, and provide a so-called SWOT-analysis in section 6.2. In section 6.3 we discuss recent developments in implementation platforms and how this relates to SIMS.

6.1 Identification of current and future competitors

6.1.1 Web-based applications and operating systems

Unlike the SIMS approach, web technologies are mainstream and can thus be viewed to be the main competitor. Web services technologies have been around for many years, and the newest generation of phones are so close to a mini-computer that many applications can be solved by being web based.

There are 4 main players trying to provide web-based operating systems / application environments:

- Microsoft, with Silverlight
- Sun, with JavaFX
- Google, with Google App Engine
- Adobe, with Adobe Integrated Runtime

However, only 2 of these platforms are currently ported to smartphones and they are not considered as major development environments for mobile devices, compared to the iPhone or Google Android SDKs.

Typical applications today are not implemented using peer interaction, but from an interaction between a central server and mobile devices (web based). The contribution of SIMS is marginal for information services that by nature are well supported by a client-server architecture.

However, some applications that by nature are between distributed components have also been implemented by centralized architectures. A typical example is MSN Messenger. Initially available as a native application in your desktop, it is now available as a web-based application (http://webmessenger.msn.com/).

However, in the case of complex interactions, collaborative services involving native applications might still be superior to web-based applications, particularly for mobile devices which have browsers with limited capabilities compared to standard desktops. Web services are inherently slow for implementing notification services, since the communication mechanisms they support are asymmetric. Here the SIMS approach has benefits, due to its symmetric message-based mechanisms.

6.1.2 Mainstream applications

Mainstream (market-leading) application vendors such as Skype do not need and do not want to be compatible with other applications since their business goal is to become the standard communication application in your device. The business model of these companies is to offer proprietary applications that will naturally spread once they become the dominant player.

6.2 SWOT[15] analysis of key results

Below we share our own analysis of the strengths, weaknesses, strengths and opportunities of the results from SIMS.

[15] Strengths, Weaknesses, Opportunities and Threats

6.2.1 Design and validation tools

6.2.1.1 Strengths

The validation functionality induces fewer errors since it is possible to check that there are no errors in the behaviour between components. It is therefore easier for developers to develop correct software services, and SIMS enables the design validation of applications in an easy way for those who are not experts.

Consequently, the approach saves time and money, since a developer does not waste time composing components that are incompatible. The validation of models reduces the need for testing code and helps to design services correctly right away. Validation tools are built in the design tools, therefore there is no need to know exactly the methodology behind it, and the developer just needs to follow the guidelines and the steps. The model enables validation without prior knowledge of the implementation of other components, it is sufficient to use the component specifications.

The model-driven approach allows easier and faster re-use of components. It enables developers to reuse an existing SIMS service. SIMS ensures interoperability between service components, and code can be generated from the models, saving time-to-market and reducing development costs while gaining quality. The Model Driven Approach (MDA) offers an easier way to develop applications in UML language. SIMS tools check and validate behaviour when most existing tools cannot even express behaviour.

6.2.1.2 Weaknesses

It takes time to learn the methods in order to use the design tool efficiently and start saving time. The key concepts are difficult to understand for developers and should be down-played in user guidelines.

Moreover, all components need to be specified in SIMS in order to interact, no ruse-engineering tools exist. Therefore it is needed to re-develop existing services with the SIMS approach to include them in the service marketplace.

6.2.1.3 Opportunities

Mobility is the trend in the industry. There are many different platforms, devices, operators and operating systems. The challenge is to develop applications that can work for all, to solve the problem of compatibility between heterogeneous devices with complex applications. Today, all services are exclusive to closed phone operators and there is no existing environment for developing services that could communicate with different networks/operators to software providers. SIMS can contribute by supporting component compatibility on the service level.

With the various APIs, modelling of applications is complicated, and it is difficult for software developers to ensure that their development is correct. They need to specify, implement and test all protocols manually. There is no validation technique valid for all components. Components that are connected together are developed by different companies with no idea of what is happening inside each other’s component. Today there are no integrated solutions for designing services.

6.2.1.4 Threats

The concept of semantic interfaces is foreign to users, and investment is needed to understand their use. Existing design tools with traditional interfaces are an easier alternative, although not allowing the semantic features or behavioural validation. Moreover many programmers prefer to code applications from scratch instead of using a model driven architecture. This may change as tool-support improves. Finally the fact that SIMS middleware is Java based can be an issue since it is not compatible with all mobile devices and would need to be ported to other environments. However, code generation tools imply that this is a minor issue.

6.2.2 Middleware

6.2.2.1 Strengths

The role learning functionalities of SIMS ensure compatibility in communication between mobile devices. The adaptive middleware enables heterogeneous devices to communicate.

The service discovery function can be useful to end-users since it enables them to select which services are available depending on what they want to achieve, their device, their OS and operator.

Also, the spreading of applications through peer triggered self update is a way to promote and distribute services with limited marketing costs. The best (i.e. most useful) services will naturally spread among a population of users.

6.2.2.2 Weaknesses

The risk of viral contamination and self propagation is high within peer-triggered self-update. It is therefore important to think of an anti-virus solution to protect users. Such a solution could itself be based on peer to peer interaction and update (similarly to the Bullguard antivirus solution: http://www.kazaa.com/us/picks/bullguard_lite.htm).

The fact that the SIMS clients must be installed on devices can be a challenge if it is not pre-installed by the device manufacturer. It would mean that the user must be informed of the existence of SIMS and then download the software, in order to be able to get and use applications. However, this is the same situation as our PCs: users have to browse or be tipped by others (e.g. to use Skype).

6.2.2.3 Opportunities

The current status of mobile technologies does not enable sufficient control over component instantiation, deployment and service discovery or distributed messaging.

Finding services and combining them is currently a problem. Today end-users need to search actively for services and then download them, hoping that they will be compatible with their friends mobile services. It is difficult for the end user to know about additional applications and to know if they will function with other devices they want to communicate with.

The available software solutions on the shelf are standard and cannot be adapted.

6.2.2.4 Threats

There is always an issue that component instantiation will not be initiated from mobile but from server to server, e.g. to ensure control over the device. Sites like www.actionengine.com provide automatic updates of mobile applications, even if they are not peer triggered. However, SIMS mechanisms for discovery and compatibility guarantees are still important, even if self-update is controlled by a server.

6.2.3 The service marketplace

6.2.3.1 Strengths

The SIMS concept would open the market of mobile applications and provide a way to develop and deploy services easily. All operators/devices/services can be used. The market would not only be reserved to market leaders.

For the Service Provider
The service providers can publish and maintain services, and attract and establish big communities. They can offer tailored services with different components, and more complex interactions can be offered without causing more need of support. SIMS offers to Service providers the possibility of new business models, an increased traffic and new revenues.

For the developer/software vendor
SIMS can open the market of services: many different developers could add their applications to the platform, more services can be created and this would open the access to many diversified applications of different origins. Services can be developed by any developer, put on the marketplace and then be distributed by self-update.

For the user
More quality: SIMS limits the number of errors by compatibility checking at runtime and validation in the design-tool: the quality of services is improved.

More communication: End-users can share a service component with another device, without needing to care about the brand, OS or Service Provider of this device. Services can be spread among users very easily, and the success/propagation of a service component depends on its popularity and not on the operator's choice.

More choice: End-users can decide which application they need or want, and do not need to depend on the operator's selection anymore. He can customise his range of applications and is not forced to keep on his phone applications that he does not need. He can even set some preferences and search for available applications.

Lower Cost: The reduction of testing and errors in the applications, and the increased volume of exchanges can cause a decrease in the price of applications.

6.2.3.2 Weaknesses

It would require a large effort to match all applications to all devices, service providers, and Operating systems. Also it is a challenge that all service components in an instance of a operational market place need to use SIMS middleware, otherwise they cannot communicate (ex: payment system by one provider, graphic interface by another provider etc…). Moreover, having SIMS middleware is a competitive advantage only if several operators use it. If only one operator uses it, the advantage is limited. Finally it is important to note that without an integrated antivirus, SIMS is not commercially applicable.

6.2.3.3 Opportunities

The current offer of service components is rigid and end-users cannot always choose which ones they want to download.
The service developers and software vendors are limited by the operators or device manufacturers who limit the access to their chosen services. It is not at all a free market.

6.2.3.4 Threats

In order for the marketplace to function, the SIMS middleware software must be installed on devices, and all service components must be developed for the SIMS marketplace. Existing service marketplaces can be a threat if the users do not experience the benefits of the SIMS marketplace.

6.3 SIMS' position w.r.t. recent developments in implementation platforms

SIMS has analysed its position relative recent developments in implementation platforms, such as the Apple iPhone and in particular the Google Android approach. This is detailed in D5.5 *Overall SIMS architecture*, which is available on the SIMS web site; here we include the conclusions only.

6.3.1 Apple iPhone

The iPhone, while being an end-user device with an appealing user interface, has inherent limitations; notably, it does not yet support deployment of service components. Services involving the iPhone must be realised using built-in software components, i.e. web services.

While there is a well-functioning marketplace for applications for the iPhone[16], component development can not reap the full benefits of SIMS due to the nature of these applications and the limitations of the platform.

6.3.2 Google Android

We have concluded that since SIMS and Android have different scope, they do not so much compete as complement each other.

With Android we expect to see more application code running on devices. Such components have a need for compatibility guarantees with other components, whether they are on other mobile devices with or without Android, or on network computers.

Android can be a promising implementation platform for SIMS components. It is however not yet a mature technology and provides only supports for service implementation, which is just a small portion for the whole service engineering process. Other limitations include lacking support of distribution, service and component deployment and instantiation as well as the fragmentation risks. SIMS can contribute to Android with methodology and tools for engineering better quality services and applications. Since SIMS middleware can execute on Android platform, SIMS technology can also contribute to a more complete tool chain and software development and deployment process, including software written for Android.

[16] See http://www.apple.com/iphone/appstore/

7 The pilot application

The chosen pilot services are a set of mobile collaborative services offering support for package delivery within the transport domain. This is a B2B application area combining information systems, positioning and telecoms technology, and where innovation makes economic and environmental sense. It involves many role players, and is not dominated by any single ICT player, and hence is a good candidate for the establishment of a service marketplace. It is also a domain in which Appear Networks does business, and has hence been chosen as the domain for the pilot service development.

A collection of services to support package delivery have been identified: customers request pick-up and delivery places and times to a transport company, who in turn distributes these to drivers. Communication services are used between the players: drivers check on e.g. parking conditions in connection with pick-up and delivery using chat-like sessions allowing media exchange (pictures), notifications are used to inform of delivery status, and incidents involving vehicles can be handled.

It is a scenario which scores best on the following criteria:

Prerequisites: Customers value

- Correctness over interfaces (protocol)
- Collaborative services (multiparty)

Desirable aspects:

- (Business) critical service (will motivate investment in special methods)
- If not correct then the price is high
- Big impact
- Multiple component developers[17]
- Protocols should not be trivial (so that validation by computers is needed)

Necessary aspects:

- Small example
- Simple to communicate
- Feasible for the project to produce in short time

A number of key SIMS benefits are demonstrated:

- Drivers may not necessarily be employed by the transport company, but could be independent contractors. In this case, they need to have the correct components in order to take part in the packet delivery. I.e. use service browsing to find the right components.
- The process of a customer ordering package pickup & delivery involves three parties (customer, transport company and vehicle/driver), and includes the option of the customer cancelling the order prior to the final committal. This entails conflicting initiatives, and thus demonstrates key SIMS validation issues.
- Pick-up and delivery places will not necessarily have the same chat services – some may have simple chat, others the advanced chat with the possibility of picture exchange (e.g. to show maps, pictures of parking conditions or the shape of the parcel); this is where peer-triggered self-update comes in to enable the pick-up / delivery places to interact with the vehicle, the service being initiated by the driver.

The pilot application is described in detail in D6.5 *Trial services*; the evaluation of the project's achievements are based on this application, see D6.6 *Evaluation of SIMS approach*.

[17] We assume components for pick-up / collection sites and for vehicles are written by different vendors.

8 Availability of results

The results of SIMS are available from the following web sites:

- The web site for the project, http://www.ist-sims.org/, contains all public deliverables and articles, as well as the project brochure and related links. The SIMS open source tools and the accompanying tool tutorial are available for the general public for experimentation use.
- WUT hosts a SIMS-related web site (http://meag.tele.pw.edu.pl/sims). The site makes available all ontologies developed, describes ontology-based techniques, and publishes ontology-based software developed by WUT within the SIMS project. The content can be reused outside SIMS, since the approach taken by WUT and applied in SIMS of using ontology-driven artefacts (ODAs) is general and domain-independent.

WUT has also prepared two movies introducing the ontology-based software developed by WUT within the project. The first movie introduces Artefact Wizard, the second movie the ODA Web Wizard. Both movies can be seen at WUT's SIMS-related web site (http://meag.tele.pw.edu.pl/sims).

9 Potential impact of the results

As we have mentioned above, if SoaML is taken up by large players, and modelling and validation of service behaviour becomes commonplace, then key SIMS technology can be taken into use by many actors in the service market. For this reason we are encouraging NEXOF-RA to adopt SoaML as a common platform.

The key benefits of the technology are its ability to enable easier and faster development of collaborative services, service interactions without errors, more service use and more revenues. The efficient component deployment methods are also an interesting feature.

Our current analysis seems to indicate that long term market exploitation is conditioned by:

- The existence of the service marketplace for collaborative services
- That at least one operator and one software vendor collaborate to implement the solution: a device equipped with SIMS components requires that the operator supports SIMS middleware.

These factors were analyzed and discussed in D7.8. Below we present our findings.

9.1 Potential impact for developers and software vendors

In order to attract developers, SIMS needs to be able to provide a measurable gain of time, money and quality, versus the time that needs to be spent on learning how to use the design tool.
Modelling of services:
Thanks to the SIMS Design Tool, developers and software vendors can develop semantic services easily thanks to model driven architecture. SIMS design tool offers the possibility to reuse and share components to design new services, which reduces time, cost and risk of error.

Testing and validation of services:
With SIMS, developers and software vendors can ensure that their development is correct with the validation tool. Several developers and/or software vendors will be able to collaborate and create services together using a common validation tool, gaining efficiency and lowering their costs.

Distribution of services:
Thanks to the SIMS technology, the volume of exchanged services will increase, it will be easier to distribute applications, reach users, and update existing services. SIMS will open the doors to a much greater market than with current static applications. With the service marketplace, developers and software vendors will be able to distributed more applications in an easier way, without the barrier of operators or device manufacturers.

9.2 Potential impact for operators

Quality of services
SIMS technology enables validation of services developed by different suppliers, which decreases the cost and time spent on checking compatibility and quality of codes. As a result, fewer errors will also decrease the time and costs linked to customer support, and improve the image of the service provider.

Distribution of services
With SIMS and the service marketplace, operators can generate revenue through either advertising, subscription, or charging each download of services. In this case the key success-factor is to involve trusted brands and clear payment standards. It is also possible to stimulate the distribution of applications by rewarding end-users that recommend services to other users.

9.3 Potential impact for end-users

Choice of applications
With SIMS, the end-users will have access to many more services and communicate with their contacts very easily, without needing to think of which device or operator they have.

Search of applications
If they wish to communicate with a friend, the phone will trigger an automatic update on their friend’s phone. If a component or an update is missing, it will be downloaded automatically from the Marketplace. The user will also be able to search for available services on the marketplace, by setting preferences and goal.

9.4 Potential impact for mobile collaborative communication

By integrating SIMS, mobile social networking and social proximity applications can improve their offer by enabling more interactions between users and easier spreading of their mobile applications. They could offer users to download the SIMS software to their device, which would enable them to use, share and update SIMS applications automatically.

SIMS will allow identifying which devices are around a user, how to interact with them, and what goals can be achieved with these devices (service discovery function).

We can imagine the development of context aware applications based on location, device, user preferences, and goal. For mobile social networking and social proximity platforms, using SIMS would be a way to spread their services among all users without needing to care about which device or operative system they are using. Examples of applications would be:

- Proximity network games with automatic updates of gaming modules depending on participants and location.
- SIMS could help Facebook or Linked-in members to connect with those who have common interests, exchange applications and add new features to their device.
- At music concerts or events, SIMS users could download and exchange applications including media, pictures, maps, and music. They could also vote for the next song to be played and see the statistics, chat with other people of the public.
- At sport matches, supporters could use betting applications linked to the game.
- Users could also use collaborative applications such as a mobile flee-market with a bidding functionality.
- To share music by streaming among other passengers on train and metros, and public places.
- Industrial applications: to add to mobile ERP systems a function of automatic update so that all employees have the right components and functionalities.
- To achieve flexibility with the employee’s role and position: employees want to achieve something but do not have the necessary software. They can get equipped and receive the relevant information by SIMS through reconfiguration of their device on the run, whenever it is needed.

Once SIMS middleware is integrated in a mobile device and an object such as a door, an elevator, or a computer, a user could connect to objects and control them from distance. This opens the doors to much revolutionary functionality like: intelligent houses, distance control of objects for disabled people, distance reparation of appliances etc.

The SIMS technology would then be used by manufacturers to enable communication with all mobile phones. It would then be possible to control objects from distance, with any device, operator or OS.

References

[1] Jacqueline Floch, Richard T. Sanders, and Rolv Bræk: "Compositional Service Engineering Using Semantic Interfaces", in *At Your Service: Service-Oriented Computing from an EU Perspective*, edited by Elisabetta Di Nitto, Anne-Marie Sassen, Paolo Traverso, and Arian Zwegers, MIT Press, ISBN 978-0-262-04253-6, pp 101-128 (To be published 2009)

[2] Michal Roj, Jarek Domaszewicz, Jacqueline Floch, Per H Meland: *Ontology-based Use Cases for Design-time and Runtime Composition of Mobile Services*, International Workshop on the Role of Services, Ontologies, and Context in Mobile Environments (RoSOC-M '08), April 27, 2008, Beijing, China

[3] Cyril Carrez, Jacqueline Floch, Richard Sanders: *Describing component collaboration using goal sequences*, in Proc. of 8th IFIP WG 6.1 International Conference, DAIS 2008, R. Meier and S. Terzis (ed), Oslo, Norway June 2008, Springer Verlag, Berlin Heidelberg, LNCS 5053, pp. 16-29

[4] Lotte Johansen, Cyril Carrez, Pawel Cieslak, Stefan Hänsgen: *The SIMS design and validation tools for Service Engineering*, Nordic Workshop on Model Driven Engineering (NW-MoDE-08), 20-22 August 2008, Reykjavik, Iceland

[5] Richard Torbjørn Sanders and Øystein Haugen: *Milestones: mythical signal in UML to analyse and monitor progress*, (to appear in) P. Mähönen, K. Pohl, and T. Priol (Eds.): ServiceWave 2008, Springer Verlag, Berlin Heidelberg, LNCS 5377, pp 110-121

[6] Shanshan Jiang, Jacqueline Floch, Richard Sanders, *Modeling and Validating Service Choreography with Semantic Interfaces and Goals*, (to appear in) 2008 IEEE International Symposium on Service-Oriented System Engineering (SOSE 2008), IEEE

[7] J. Domaszewicz, P. Cieślak, M. Rój, T. Rybicki, R. Sanders, J. Floch: *Projekt SIMS - Interfejsy Semantyczne w Usługach Mobilnych*, Krajowe Sympozjum Telekomunikacji i Teleinformatyki (National Telecommunications and Teleinformatics Symposium), KSTiT 2006, 13-16 September 2006, Bydgoszcz, Poland

[8] P. Cieślak, J. Domaszewicz, M. Rój, T. Rybicki: *Advances in SIMS: towards an ontology of telecommunication services*, Krajowe Sympozjum Telekomunikacji i Teleinformatyki (National Telecommunications and Teleinformatics Symposium), KSTiT 2007, Bydgoszcz, Poland, 12-14 September 2007; printed in „Przegląd Telekomunikacyjny" 8-9/2007, ISSN 1230-3496, pp. 752-758

[9] Michał Rój, Paweł Cieślak, Robert Dawidziuk, Jarosław Domaszewicz: *Wykorzystanie ontologii do projektowania i wzbogacenia funkcjonalności usług telekomunikacyjnych Projekt SIMS*, Krajowe Sympozjum Telekomunikacji i Teleinformatyki (National Telecommunications and Teleinformatics Symposium), KSTiT 2008, Bydgoszcz, Poland, 10-12 September 2008, published in Przegląd Telekomunikacyjny 8-9/2008, ISSN 1230-3496, pp. 954-957

[10] Paweł Cieslak, Michał Rój and Jarosław Domaszewicz: *Artefact Wizard: a tool enabling creation of ontology-driven artefacts*, submitted to CAISE'09

[11] Michał Rój, Robert Dawidziuk, Jarek Domaszewicz and Paweł Cieslak: *Mobile Semantic Explorer: a Facility for Finding Ontology-annotated Items*, submitted to ACM WWW2009

[12] Xavier Aubry: *EU Research Project SIMS Brings Ease of Use to Mobile Users*, Appear Networks Press release, 17 September 2007

[13] Xavier Aubry: *User Story: Workforce Application SIMS*, Leaflet, 11 September 2008