# Ontology-based Use Cases for Design-time and Runtime Composition of Mobile Services

Michał Rój[1], Per Håkon Meland[2], Jacqueline Floch[2], Jarosław Domaszewicz[1]
[1]*Institite of Telecommunications, Warsaw University of Technology*
[2]*SINTEF ICT, System Development and Security*
*mroj@tele.pw.edu.pl, per.h.meland@sintef.no,*
*jacqueline.floch@sintef.no, domaszew@tele.pw.edu.pl*

## Abstract

*This paper presents application of ontology-based modelling and reasoning related to the different phases of the lifecycle of mobile services. Ontology-based descriptions complement traditional design-time and runtime models allowing more complex reasoning. We present use cases for ontologies that may be applied at design time, deployment time and/or runtime. Some important characteristics of our approach are: 1. ontological descriptions define complex artefacts that are built from simpler ones defined in an ontology; 2. a single ontology can be used for specifying various artefacts and for reasoning on various aspects at different phases of the service lifecycle; 3. an artefact can be used for various purposes. This paper provides examples of ontological descriptions along with use cases, and discusses the applicability of the approach.*

## 1. Introduction

One of the core challenges of service engineering is to find practical ways to model services and service features independently of each other, such that services may be composed into well functioning systems that satisfy their requirements. Service composition in general involves discovery, reuse and static composition at design time as well as dynamic discovery, deployment and binding at runtime. The lack of machine-readable semantics currently requires human intervention for automated service discovery and composition, thus hampering ease-of-use and ease-of-composition. Ontology-based modelling tries to solve this problem by adding significance to the traditional modelling languages, and thus enabling more complex reasoning during discovery and composition.

The SIMS project[1] introduces *semantic interfaces* to specify the collaborative behaviour of service components and the goals that can be achieved through collaborative behaviour, and to guarantee compatibility in static and dynamic component compositions. SIMS addresses semantics at two levels: 1. UML is used to specify the semantic interface behaviour of service components, and the progress that might be achieved in a collaborative behaviour [1], 2. Ontologies are used to define extra-functional properties of services, service components and other service entities relevant for discovery and composition. For example, we use ontologies to specify *collaboration goals,* i.e. the desirable outcome achieved through a collaborative behaviour. Even though most service entities have representations in both UML and ontology universes, these representations do not overlap but complement each other. A main motivation for using UML and not a pure ontology-based approach is that UML is widely used by software developers. In that way we are also able exploit existing validation techniques. While the detailed behaviour descriptions in UML allow us to validate the safety and liveness properties of service collaborations, the ontological descriptions allow us to reason on other properties such as service intention and required device capabilities. Ontological descriptions are also exploited to provide developers and end-users with additional information about services. Modelling and validation using UML are not discussed in this paper. We rather concentrate on the ontological approach by introducing artefacts we have found useful for reasoning during service discovery and composition, and discuss use cases for design-time and runtime.

Unlike the Service Oriented Architecture paradigm (SOA), where services are normally understood as

[1] Semantic Interfaces for Mobile Services (SIMS), http://www.ist-sims.org/

capabilities provided by a service provider to a service consumer, our work considers collaborative services that entail collaborations between several autonomous entities that may behave in a proactive manner and may take initiatives towards each other. This is typical for telecom services, but also for a large class of services such as attentive services, context-aware services, notification services and ambient intelligence. Also, we do not restrict to describing external service properties, but also propose techniques for engineering services.

## 2. Semantic Interfaces for Mobile Services: approach overview

The core research of SIMS is to provide new means to specify services, to develop well-formed components that realize these services and compose services with compatibility guarantee. Semantic interfaces combined with an ontology are instrumental to a goal driven development process, and enable automated service discovery, selection and composition mechanisms at runtime.

### 2.1 Fundamental concepts

A central concept of SIMS is the principle of a **service collaboration**. A service is defined as a collaboration between distributed **service components** and delivers functionality to its environment. Service components are software entities that may partake in multiple services. To ease component design and validation, we distinguish between the service roles that a component plays in different services. In our approach we specify services using UML2.0 collaborations [1]. We distinguish between elementary and composite collaborations, the former defines a simple interaction between the interfaces of two service roles, the latter a structure of interacting service roles. Services are complex structures modelled by composite collaborations (Figure 1).

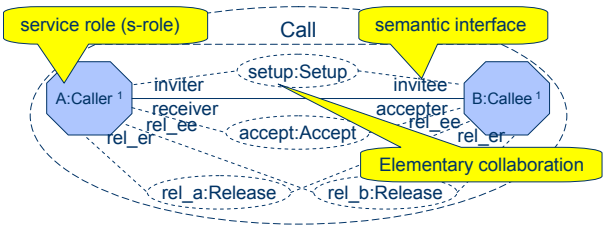


**Figure 1. Composite collaboration with two roles**

Service roles are characterized by **semantic interfaces** and interface dependency graphs that are used to validate interactions between service roles and to compose them [2]. A semantic interface represents a partial behaviour of the service role in an interaction towards another service role. Semantic interfaces are specified by state machines (Figure 2) with semantics of message passing allowing validation of safety properties (i.e. avoiding occurrence of bad behaviour, such as deadlocks). In addition to the behavioural semantics we also specify ontological artefacts that define extra-functional properties of interfaces (see Section 3).

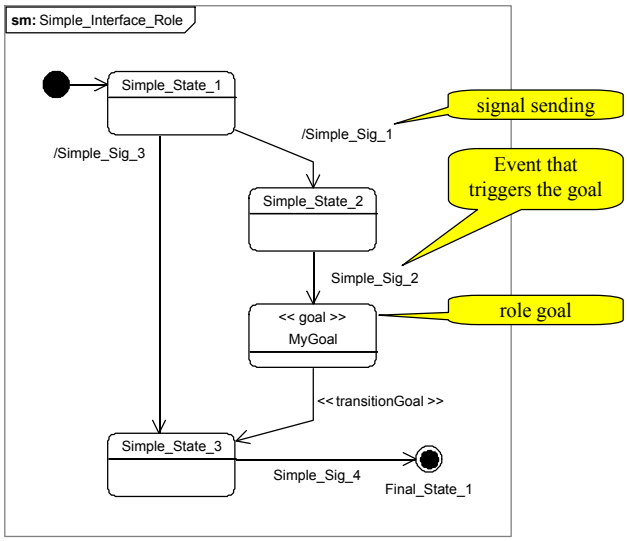


**Figure 2. The state machine of a semantic interface**

Beyond safety properties we express liveness properties (i.e. ensuring that some desired behaviour may occur) using so-called **service goals**. Service goals describe that something desirable may occur [3]; as such they do not describe a behaviour, but rather an abstract concept of the desired behaviour outcome. We distinguish between **role goals** that are associated to events in the behaviour of semantic interfaces or service roles, and **collaboration goals** that express what might be achieved thorough an elementary collaboration. A service specification should contain a **collaboration goal sequence**, which specifies the dependencies between the goals of the elementary collaborations that form the composite service collaboration.

## 2.2 The SIMS service lifecycle

A generic lifecycle of a mobile service is shown in Figure 3.

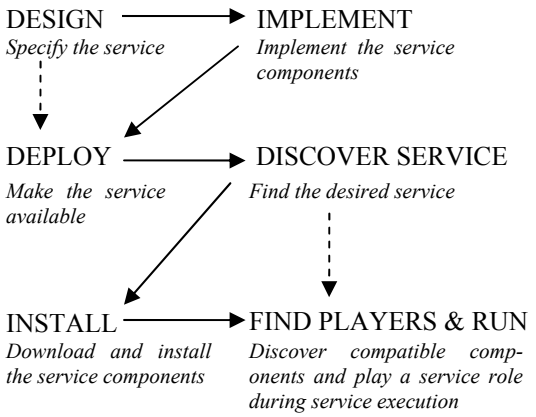


**Figure 3. The SIMS service lifecycle**

A service designer specifies a service using collaborations, semantic interfaces and goals. In addition, the service designer produces ontological descriptions for the designed entities. Service components realising the specified behaviour are developed, unless the service designer is able to find and reuse existing components (upper dotted arrow). Different parties can implement service components, provided the service specification is followed.

A service provider deploys service specifications and components so they are available for discovery. Services can be discovered through various means, and may eventually result in the end user downloading components to her device. Alternatively, the user already has a component that is able to play the desired service role (lower dotted arrow).

At runtime, service sessions are usually initiated by end users that decide to initiate a service. Discovery of compatible service components is performed previous to session creation allowing service components to find service component instances with which they can successfully achieve service goals. Depending on the type of service, component may restrict the search to particular contexts.

This lifecycle benefits from support provided by ontologies; the next sections focus on this.

# 3. Describing entities with ontology

In SIMS, an ontology spans two layers: the higher is called *SIMS ontology of telecommunication services* (in short: *SIMS ontology*) and the lower is a set of so-called *ontology-driven artefacts* (*ODAs*). The SIMS ontology contains a number of concepts of the telecommunication domain relating to services, actions, activities and their attributes. It has to be available prior the service design phase.

Ontology-driven artefacts (ODAs) are used to define SIMS-specific entities, and created using concepts, relations and properties found in the SIMS ontology. In addition to a "vocabulary," the ontology enables different types of ODAs to be constructed by the service designer in the service design phase[2].

Both layers are implemented using the Web Ontology Languages (OWL) [4]. From a technical point of view, an ODA is an OWL class constructed by using classes from the SIMS ontology and tying them together with relations using so-called class constructors provided by the underlying ontology language (OWL)[3].

## 3.1 SIMS ontology sample concepts

The SIMS ontology itself is not discussed in this paper. We rather concentrate on providing a "taste" of it by showing a selection of concepts in Figure 4.

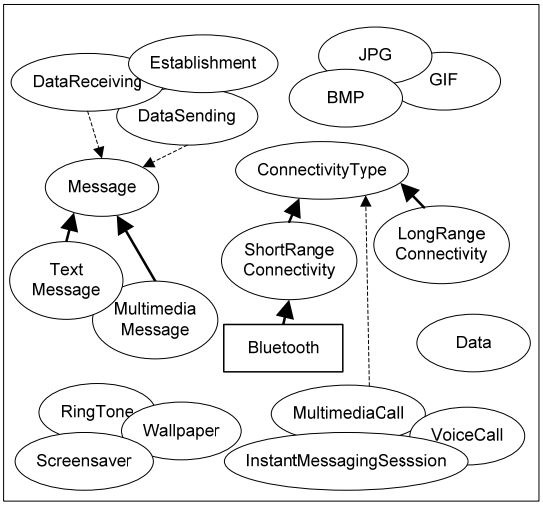


**Figure 4. SIMS ontology: sample concepts**

The ontology contains a number of concepts (ovals in the figure). Some concepts are organized in concept-subconcept hierarchies (linked with solid arrows), e.g. the *ConnectivityType* concept has two subconcepts: *ShortRangeConnectivity* and *LongRangeConnectivity*. In addition, the ontology can contain individuals (instances of concepts), shown as

[2] Some types of ODAs, such as user preferences ODAs, can be built or generated in later phases.

[3] The most prominent class constructors in OWL are existential and universal restrictions.

rectangles in the figure (*Bluetooth* is the only individual in the above example; the type of this individual is the *ShortRangeConnectivity* concept). Dotted arrows represent restrictions on properties (which are unnamed in the figure). Example properties are *refersToCommunication* (used to define a specific entity as referring a specific type of communication), *hasParticipant* (used to specify what kinds of participants are involved) or *hasNetworkConnectivity* (used to specify the network technology involved, such as wireless, fixed, etc.).

## 3.2 Types of ODAs in SIMS

We apply different kinds of ODAs to formally express semantic properties of the concepts mentioned above. The following sections present five different kinds of ODAs, one per section, exemplified with a so-called Manchester OWL syntax [5], which has a more concise syntax than "pure" OWL. Other ODA types exist but not mentioned in this paper.

**3.2.1 Service ODA.** The service ODA describes the functionality that can be obtained by the user of the service. It is specified by providing a general type of the service (e.g. voice communication, a multimedia conference, etc.) and its attributes (such as number of participants, media involved, and types of devices required). The Service ODA, which is mainly aimed for end-users, allow them searching and subscribing for services based on their attributes. It can be also easier to understand the service added value with this ODA.

The following is an example of a Service ODA (this is a conference with at least 2 participants with video or audio involved):

```
MultimediaCall
that containsStream some  VideoStream
  and containsStream some  AudioStream
  and hasParticipant some CommunicationParticipant
  and hasParticipant min 2
```

**3.2.2 Goal ODA.** Service goals (ref. Section 2.1) are expressed with Goal ODAs, e.g. role goals are connected to states in the state machines of the semantic interfaces (Figure 2). Goal ODAs are built of the concept describing the action that might be achieved (e.g. "establishment", "detachment", "initiation") and additional attributes (e.g., specifying what kind of connection should be established).

Goal ODAs can be used to find out if two semantic interfaces have goals that are semantically close, or thus to filter what semantic interfaces may be validated for interaction. The following goal example describes a multimedia call with a video/avi mime type:

```
Establishment
that refersToCommunication some MultimediaCall
  and refersToData some
    (StructuredData that hasMimeType
      value video/avi)
```

**3.2.3 Device ODA.** This ODA describes the capabilities of devices (such as ability to display pictures) and represents physical end-user devices (such as mobile telephones). The ODA contains information about the type of the device, and its hardware and software attributes. The type of the device can indicate if it is a smart phone, PDA, or PC/laptop. Hardware attributes capture the display or audio capabilities, memory, connectivity, etc. while software attributes would typically indicate if a browser is available, the multimedia codecs, etc.

Device ODAs can be helpful for the service designers: when designing a service and its components, it may be useful to find out what types of devices the service components can run on. Finally, Service ODA can be taken into account along with the Device and Component ODAs, in order to look for components that can realize or partially realize a specific service (e.g. for a given service type, one can find components suitable for different classes of devices).

The following definition describes a simple GSM mobile phone:

```
Device
that isDescribedByDevComponent some
(HardwarePlatform
  that containElements some (Keyboard
    that hasTextInputCapable value "true"^^boolean)
     and containElements some (Display
      that hasColourCapable value "false"^^boolean)
     and hasVoiceInputCapable value "true"^^boolean
)
and isDescribedByDevComponent some (SoftwarePlat
form
     that acceptMime value text/plain)
and isDescribedByDevComponent some (NetworkCha
racteristics
    that supportNetworkBearers value GSM_CSD_MSI
SDN)
```

**3.2.4 Component ODA.** The Component ODA gives an abstract description of the service component (ref. Section 2.1), describing the how the component

participates in the service, its features and its limitations.

Since the Component ODA is built using the same ontology as the service ODA, it is possible to find services (Service ODAs) which can be accessed through a given component, and conversely it is possible to find what components (Component ODAs) can realize a specific service (described by a Service ODA). Also, with this ODA, one can check what devices will handle a specified component, by comparing Component ODAs with Device ODAs. It is also possible to provide the user with information about what goals can be achieved by a component and to compare new components with components already installed by the user thus allowing for service role learning.

The following Component ODA example describes a component to be installed on a mobile phone, providing instant messaging exchange functionality over Bluetooth. It requires a proper device class to work properly (a Bluetooth-enabled phone), with the possibility of achieving a specific goal (message sending/receiving) and implements the functionality of the instant messaging service). Its OWL definition is as follows:

```
Component
that requiresDevice some (UserTerminal
        that isDescribedBy some (HardwarePlatform
          that containElements some (Bluetooth
                that supportBluetoothProfiles
value GenericObjectExchangeBluetoothProfile))
        and isDescribedBy some (SoftwarePlatform
             that acceptMime value text/plain))
and achievesGoal some (MessageSending
       or MessageReceiving)
and canRealizeService some InstantMessaging
```

**3.2.5 User preferences ODA.** The User Preferences ODA describes the user wishes about the service provided, expressed in terms of selected properties of the service or service goals. It represents a class of services the user wants to be involved in. The User Preferences ODA can be matched with other types of ODA, e.g., the Goal or Service ODAs, and be taken into consideration before the user accesses the service.

The following example expresses that a user prefers free instant messaging services:

```
UserPreferences
that hasPreferenceGoal some (Establishment
        that refersToCommunication some
(FreeService and InstantMessaging))
```

## 4. Ontology-related use cases

This section describes what we consider to be the most promising use cases for ODAs during design time creation (DT) and run-time discovery of mobile services (RT). At DT the main stakeholder is the service designer, while at RT the stakeholders are the service provider and the end user of the service.

Three of the following use cases are used in both DT and RT: (1) finding services with specific features, (2) finding components which realize a specific service, (3) finding compatible components. However the latter is only discussed for RT. In addition, a DT-specific use case is described (4) checking if the sequence of service goals is correct. Other useful use cases, but not described here[4], are: (5) subscribing for services with specific features, (6) finding non-dual semantic interfaces for subtype validation, (7) checking if the device can be used in a service, (8) complying with user preferences in service execution, (9) checking consistency between elementary collaboration goals, (10) checking consistency between devices and components.

Most of the above use cases have been implemented and tested. For matching ODA artefacts, subsumption (class-subclass) detection is used. For example, to find services with specific features (use case (1)), the Service ODA class (which covers only selected features) is created and matched with all available Service ODAs.

### 4.1 Find services with specific features

At design time, a service designer wants to create a service, but before that, she should check if such a service already exists or there might be something that she could build upon (this service could be created by others within her organisation or by outsiders who have publicly released their Service ODAs). She expresses the relevant service features by building a simple service ODA covering her needs and a search returns a set of services (if any) that will be subject for closer inspection. The search itself can be made on various levels of abstraction (more general / more specific) by exploiting the concept relations in the ontology.

At runtime, a user wishes to achieve some functionality and is looking for a service that can support this. He can specify the service features he is interested in, and a service repository presents available service candidates. He can then browse these

[4] See the publicly available reports at http:///ist-sims.org/ to have these explained.

services and download components for the most promising one.

An example is as follows. Assume that there is a service as specified in Section 3.2.1 that is defined in some DT/RT repository. A user/developer searches for services able to carry video which are not limited by a number of participants. She builds an ODA which can look as follows:

```
Communication
that containsStream some VideoStream
```

After the query is done, the service specified in Section 3.2.1 is returned as a match. In that case, information about the limitation on the number of participants will also be given to the user/developer.

## 4.2 Find components realizing a specific service

By comparing Component ODAs and Service ODAs it is possible to find all services related to a given component and vice versa. At design time, the service designer can find the components that that play service roles specified for the service or compatible roles, and thus can participate to the service. At runtime, components can be found, and consequently downloaded, based on desirable service features. A user can also identify that components installed on his device can be used in services they were not originally downloaded for.

For example, the service designer wants to know if there are any components implementing the instant messaging service, she makes a very simple artefact consisting of one concept only:

```
InstantMessaging
```

Assuming that the repository of components contains the Component ODA as defined in Section 3.2.4, this component will be returned to the designer.

## 4.3 Check if the sequence of goals is in the right order

This use case allows a service designer to check of two or more goals used in a goal sequence are properly ordered. By defining sequence rule relations in the telecom relation, it is possible to detect that the order of two or more elementary collaboration goals are not consistent.

Consider the following example; a service architect creates a conference service, where, among others, the achievement of two goals is desirable:

- Invitation to a multimedia conference (goal A)
- Establishment of a multimedia conference (goal B)

Assume that both goals are defined for elementary collaborations and their semantics defined by ODAs. Goal B may be defined as specified in Section 3.2.2, and goal A may be defined in a similar way. If the developer specifies a goal sequence where goal A is achieved before goal B, reasoning on the ODAs should make it possible to detect a wrong ordering and inform the developer about this sequence inconsistency. Knowledge that *establishment* of a conference takes place before *invitation* of participants is expressed in the SIMS ontology.

This use case should benefit service designers by assisting them during service creation and to assure that incorrect services will not be created.

## 4.4 Find compatible components

The Component ODAs provide a way to limit the search space for compatible components. We say that one component is *compatible* to another component if the former can be put in place of the latter without loosing (partial) functionality. It means that the latter can provide a wider functionality than the former but never more limited.

We search for compatible components if we want (1) to upgrade an existing to a newer version (e.g., a more stable one), (2) to extend the existing component with new features.

The example for this use case is as follows: Assume that a user has a Bluetooth instant messenger component installed as described in Section 3.2.4. The user enjoys the component very much and would like to have a similar component (in terms of functionality), which can run on a wider range of networks than Bluetooth. Then he makes a query: “find all components compatible to my current component.” A list of detected components is displayed to the user. One of them supports Bluetooth and other kinds of connectivity (such as GPRS). Then the user decides to download and install the component.

Notice that the compatibility between components cannot be ensured only by making ontology-based comparisons. Theoretically, incompatible components can provide the very same functionality (and hence, be described with identical ODAs). The full component compatibility can be checked using other UML-based validation techniques developed in SIMS. In these

terms, the two technologies based on ontology- and UML-reasoning complement each other very well.

## 5. Case study: A virtual meeting place

The following case study describes the creation and use of a single but compound service where we take advantage of the use cases described in the previous section. The service itself is not revolutionary, you can find similar functionality already offered different mobile device manufacturers and mobile operators, but notice *how* the SIMS technology solves design time and run-time issues.

In a virtual meeting place you would like people sharing a common interest or a common task (e.g. a company or a charitable institution) to interact with each other seamlessly regardless of geographic boundaries and across multiple means of communication. Actors of the meeting place service are (among others) *participants* (everyone who participates in the service) and a *controller* (the one who can create a *conference* within the meeting place, *invite* others and other things related to the conference). Figure 5 shows the composite collaboration diagram of how such a service might look like. The hexagons are service roles (played by service components), while the ovals are elementary collaborations connecting two semantic interfaces.

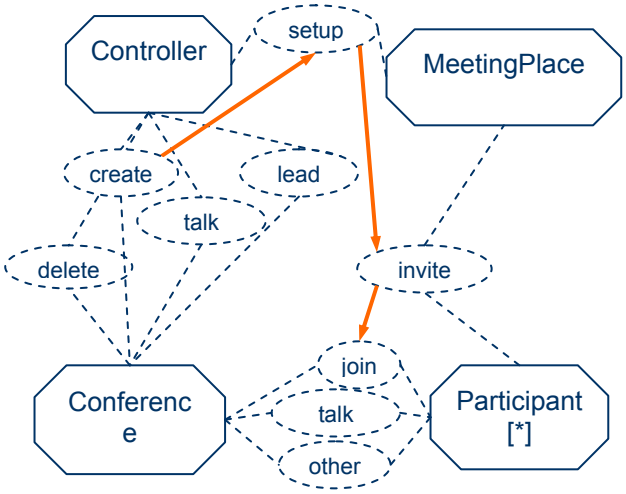


**Figure 5. The design of a virtual meeting place**

At design time the service designer looks for similar services as described by the use case in section 4.1. Ontological concepts related to *conference* would be natural search criteria. A set of matching existing services can then be used to find components she can reuse in her new meeting place service, as described by the use case in section 4.2. Let's say a component able to play the conference role is found and reused. The conference role will have a set of semantic interfaces which can be used to create the required duals (corresponding interfaces) for the other roles that will have to be played by newly implemented components. The use case in section 4.3 helps the service designer to define the right goal sequences (show with arrows in Figure 5).

Now, let's jump to runtime. Three users are participating within the same meeting place, and would now like to setup a common voice conference. Two of them have components realizing the voice communication (*talk*), while the third one has a component that supports voice and video communication. As described in the use case in section 4.4, these components could be found compatible as long as the voice data is the only thing transferred.

The virtual meeting place is a service being realized to demonstrate the SIMS approach's practical applicability and usefulness. Both existing and new components for mobile devices are used, and a centralized middleware has been developed for run-time reasoning.

## 6. Related work

Application of ontologies in SIMS is an original merge of use cases at design-time and runtime. When we separate the use cases into concrete phases we can find similarities with existing approaches.

At runtime, similar approaches have been developed in the area of Semantic Web Services (SWS). Two most prominent technologies are Web Service Modeling Ontology (WSMO) [6] and OWL-S [7]. Both WSMO and OWL-S use ontologies to describe functionality of *services*. Common to the SIMS and SWS approaches, a domain ontology is used to build artefact to represent specific entities (e.g., *goals*, *preconditions*, *assumptions* in WSMO, *service type*, *effects* in OWL-S). The same vocabulary is used for building the "advertisements" (descriptions of what is provided) and the "requests" (descriptions of what is desired) to be matched at runtime. The artefacts defined in SWS are different from those in SIMS. This results mainly from different understanding and definition for services. SWS addresses *Web services* that may be defined as "a computational entity which is able to achieve users' goals by invocation" [6], while SIMS address collaborative services. Another difference is that service providers offering Web services are responsible for their creation and management, and

services that are discovered are ready to be consumed. In SIMS, a service results from a collaboration between components possibly developed by different stakeholders. The concept of service discovery is extended and discovery might require downloading and instantiation of a service components.

Benefits from applying ontologies at design time have already been presented in the literature. Deridder and Wouters [8] point that using precise terminology (from an ontology) to annotate software entities addresses the problem of language ambiguity. This is of interest in the case of complex services and large service systems, where component development usually involves different developers. Since the SIMS ontology-based use cases cover most of the entity types the services are built of, annotation can be widely applied. Happel and others [9] present how software artefacts (especially in software libraries) can be easily reused if properly annotated with ontologies. The authors exploit ontology-based queries, which, when ontologies are involved, are a very flexible means to acquire relevant information. In a similar way the SIMS use cases also support reusing (e.g., services, components, elementary collaborations) and querying (e.g., available services). Using ontologies to assist in software engineering is addressed by the W3C initiative, "Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering"[5]. Several uses described in the document (such as "Software Lifecycle Support") are covered by the SIMS approach.

## 7. Summary and conclusions

This paper shows a novel way of exploiting ontologies, which can be summarized by the following points: (1) ontology use in many phases of the service lifecycle, (2) the use of so-called ontology-driven artefacts, (3) ontology techniques combined with other techniques within a single framework. In contrast to approaches where ontologies are used for a single purpose such as to discover a web service or to facilitate communication between designers and developers, in SIMS a single ontology finds its use in many phases and for different purposes. Specifically the ontology-driven artefacts that are created in the service design phase are exploited at design-time, deployment and runtime phases.

Selected use cases are currently being validated by the SIMS demo applications. The authors will examine which use cases are most useful for different kinds of users (service designers, component developers, and end users of services).

## 8. Acknowledgements

Our work is funded by the European Community under the Sixth Framework Programme, contract FP6-IST-027610 SIMS. Contributions from Cyril Carrez, Richard Sanders and Robert Dawidziuk are gratefully acknowledged.

## 9. References

[1] R. Sanders, H. Castejón, F. Kraemer, and R. Bræk, "Using UML 2.0 Collaborations for Compositional Service Specification," presented at 8th International Conference of Model Driven Engineering Languages and Systems, 2005.

[2] J. Floch and R. Bræk, "A Compositional Approach to Service Validation," presented at SDL 2005: Model Driven Systems Design: 12th International SDL Forum, Grimstad, Norway, 2005.

[3] B. Alpern and F. Schneider, "Recognizing safety and liveness," Cornell University, Computer Science Department TR86-727, 1986.

[4] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein, "OWL Web Ontology Language Reference," vol. 2005, 2005.

[5] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang, "The Manchester OWL Syntax," presented at OWL Experiences and Directions Workshop (OWLED'06) at the ISWC'06, Athens, Georgia, USA, 2006.

[6] D. Roman, U. Keller, H. Lausen, J. d. Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel, "Web Service Modeling Ontology," *Applied Ontology*, vol. 1, pp. 77-106, 2005.

[7] OWL Services Coalition, "OWL-S: Semantic Markup for Web Services, version 1.0," 2004.

[8] D. Deridder and B. Wouters, "The use of ontologies as a backbone for software engineering tools," *Proceedings of the Fourth Australian Knowledge Acquisition Workshop AKAW99*, pp. 187-200, 1999.

[9] H.-J. Happel and S. Seedorf, "Applications of Ontologies in Software Engineering," presented at 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), Athens, GA, USA, 2006.

[5] http://www.w3.org/2001/sw/BestPractices/SE/ODA/