Semantics-Aware Services for the Mobile Computing Environment

Nikolaos Georgantas, Sonia Ben Mokhtar, Ferda Tartanoglu, Valérie Issarny

Semantics-Aware Services for the Mobile Computing Environment

Nikolaos Georgantas, Sonia Ben Mokhtar, Ferda Tartanoglu, and Valérie Issarny

INRIA, UR Rocquencourt, Domaine de Voluceau, B.P. 105
78153 Le Chesnay Cedex, France
{Firstname.Lastname}@inria.fr
http://www-rocq.inria.fr/arles

Abstract. Today's wireless networks and devices support the dynamic composition of mobile distributed systems according to networked services and resources. This has in particular led to the introduction of a number of computing paradigms, among which the Service-Oriented Architecture (SOA) seems to best serve these objectives. However, common SOA solutions restrict considerably the openness of dynamic mobile systems in that they assume a specific middleware infrastructure, over which composed system components have been pre-developed to integrate. On the other hand, the Semantic Web introduces a promising approach towards the integration of heterogeneous components; current semantics-based approaches are, however, restricted to application-level interoperability. Combining the elegant properties of software architecture modeling with the semantic reasoning power of the Semantic Web paradigm, this paper introduces abstract semantic modeling of mobile services that allows both machine reasoning about service composability and enhanced interoperability at both middleware and application level.

1 Introduction

Mobile distributed systems cover a broad spectrum of software systems, by considering all the forms of mobility, i.e., personal, computer, and computational [9]. In this paper, we focus on the mobility of devices, as enabled by today's wireless devices. Then, most specifics of mobile distributed systems compared to their stationary counterpart follow from the features of the wireless infrastructure. Mobile software systems must in particular cope with the network's dynamics and quality of service (QoS) management; this is particularly challenging due to resource constraints of the wireless devices and varying bandwidth. A general approach to the management of the network's dynamics, following advances in wireless networks, lies in the automatic configuration and reconfiguration of networked devices and services. This is in particular supported by discovery protocols that provide proactive mechanisms for dynamically discovering, selecting and accessing reachable services and resources that meet a given specification [23]. This leads to building systems, in which (wireless) nodes advertise and consume networked resources according to their specific situation and requirements. This further leads to the design of mobile distributed systems as

R. de Lemos et al. (Eds.): Architecting Dependable Systems III, LNCS 3549, pp. 1 – 35, 2005.

systems of systems, whose component systems are autonomous and hosted by networked nodes, either wireless or stationary. The systems' configuration then evolves and adapts according to the network connectivity of component systems.

Despite the above dynamics, composition of systems shall ensure the correctness of the system's behavior with respect to target functional and non-functional properties. With respect to the former, the composition must enforce selection of the appropriate component systems and coordination protocols that conform to the specification of the component systems. More specifically, coordination protocols shall be agreed upon by the component systems, i.e., the communication protocols to be followed and their behavior need to be understood and adhered to by all the composed parties, although the protocols implemented by the resulting composite system cannot be fixed at design time. With respect to the latter, it is mandatory to account for the quality of service delivered by component systems and their integration. Specifically, the dynamic composition of mobile distributed systems must both minimize resource consumption on mobile nodes and satisfy the users' requirements with respect to perceived QoS [11].

The dynamic composition of mobile distributed systems from component systems poses further the challenge of interoperability. The composed systems may be implemented and deployed on different software and hardware platforms and assume different network infrastructures. Many of the network interoperability aspects can be addressed by reliance on the ubiquitous Internet's network and transport protocols. However, at middleware and application level, the interoperability problem remains, concerning further both functional and non-functional properties. Considering the large number of players and technologies involved in realizing current mobile distributed systems, solutions to interoperability based on reaching agreements and enforcing compliance with interoperability standards cannot scale. Instead, component systems shall adapt at runtime their functional and non-functional behavior in order to be composed and interoperate with other component systems. Moreover, supporting composition and interoperation requires the definition of behavioral conformance relations to reason on the correctness of dynamically composed systems with respect to both functional and non-functional properties.

Various software technologies and development models have been proposed over the last 30 years for easing the development and deployment of distributed systems (e.g., middleware for distributed objects). However, the generalization of the Internet and the diversification of connected devices have led to the definition of a new computing paradigm: the *Service-Oriented Architecture (SOA)* [29], which allows developing software as services delivered and consumed on demand. The benefit of this approach lies in the looser coupling of the software components making up an application, hence the increased ability to making systems evolve as, e.g., application-level requirements change or the networked environment changes. The SOA approach appears to be a convenient architectural style enabling dynamic integration of application components deployed on the diverse devices of today's wireless networks. This paper provides an overview of SOA principles together with that of the most popular existing software technology complying with the SOA architectural style, which is the *Web Services Architecture*[1]. The Web Services paradigm has been successfully employed in elaborating mobile distributed systems [5]. However, the SOA paradigm

[1] http://www.w3.org/TR/ws-arch/

alone cannot meet the interoperability requirements for mobile distributed systems. Drawbacks include: (i) support of a specific core middleware platform to ensure integration at the communication level; (ii) interaction between services based on syntactic description, for which common understanding is hardly achievable in an open environment.

A promising approach towards addressing the interoperability issue relies on semantic modeling of information and functionality, that is, enriching them with machine-interpretable semantics. This concept originally emerged as the vehicle towards the *Semantic Web*[2][2]. The semantic representation of Web pages' content aims at enabling machines to understand and process this content, and to help users by supporting richer discovery, data integration, navigation, and automation of tasks. Semantic modeling is based on the use of ontologies and ontology languages that support formal description and reasoning on ontologies; the *Ontology Web Language (OWL)*[3] is a recent proposition by W3C. A natural evolution to this has been the combination of the Semantic Web and Web Services into *Semantic Web Services* [16]. This effort aims at the semantic specification of Web Services towards automating Web services discovery, invocation, composition and execution monitoring.

The Semantic Web and Semantic Web Services paradigms address application-level interoperability in terms of information and functionality [3,17]. However, interoperability requirements of mobile distributed systems are wider, concerning functional and non-functional interoperability that spans both middleware and application level; conformance relations enabling reasoning on interoperability are further required. In our previous work [6], building on software architecture principles, we elaborated base modeling of mobile software components, which integrates key features of the mobile environment and allows for reasoning on the correctness of dynamically composed systems with respect to both functional and non-functional properties. Building on this work as well as on SOA and Semantic Web principles, we introduce in this paper semantic modeling of mobile services to enable interoperability and dynamic composition of services. Specifically, we introduce OWL-based ontologies to model the behavior of mobile services, which allows both machine reasoning about service composability and enhanced interoperability. We note that our focus is on the functional behavior of services; specification of the non-functional behavior of services and definition of related ontologies is part of our future work, still based on [6]. We further point out that our approach to interoperability is generic, thus, it may as well apply to non-mobile systems. Nevertheless, the requirement for dynamic composition and interoperability is particularly evident in mobile systems, due to their high dynamics and, principally, heterogeneity. Specialization to mobile systems will get clearer when our solution will further address non-functional properties. Our work described in this paper is part of the effort of the IST Amigo[4] project, which elaborates a generic framework for integration of the mobile communications, personal computing, consumer electronics and home automation domains in the networked home environment.

[2] http://www.w3.org/2001/sw/
[3] http://www.w3.org/TR/owl-semantics/
[4] http://www.extra.research.philips.com/euprojects/amigo/

In the following, Section 2 provides an overview of the Service-Oriented Architecture paradigm, integrating the Web Services, Semantic Web and Semantic Web Services paradigms. Section 3 introduces our semantic modeling of mobile services. Based on this modeling, Section 4 presents our approach towards semantics-based interoperability. We discuss related work in Section 5 and conclude in Section 6.

2 Service-Oriented Architecture

Service-oriented computing aims at the development of highly autonomous, loosely coupled systems that are able to communicate, compose and evolve in an open, dynamic and heterogeneous environment. Enforcing autonomy with a high capability of adaptability to the changing environment where devices and resources move, components appear, disappear and evolve, and dealing with increasing requirements on quality of service guarantees raise a number of challenges, motivating the definition of new architectural principles, as surveyed below for the service-oriented architectural style. Key properties for service-orientation are further discussed in Section 2.2. Section 2.3, then, presents software technologies enabling service-orientation, focusing on the Web Services Architecture. Finally, an overview of Semantic Web standards and the Semantic Web Services is presented in Section 2.4.

2.1 Service-Oriented Architectural Style

A service-oriented system comprises autonomous software systems that interact with each other through well-defined interfaces. We distinguish *service requesters* that initiate interactions by sending service request messages and *service providers* that are the software systems delivering the service. An interaction is thus defined by the sum of all the communications (service requests and responses) between a service requester and a service provider, actually realizing some, possibly complex, interaction protocol.

Communications between service requesters and providers are realized by exchanging messages, formulated in a common structure processable by both interacting partners. The unique assumption on these interactions is that the service requester follows the terms of a *service contract* specified by the service provider for delivering the service with a certain guarantee on the quality of service. The service requester does not make any assumption on the way the service is actually implemented. In particular, neither the service name nor the message structure implies any specific implementation of the service instance. Indeed, the service implementation may actually be realized either by a simple software function or by a complex distributed system involving as well third party systems. Similarly, the service provider should not make any assumption about the implementation of the service requester side. The only visible behavior for interacting parties is the protocol implemented by the exchange of messages between them.

A *service-oriented architecture* is then defined as a *collection* of service requesters and providers, interacting with each other according to agreed *contracts*. Main characteristics of the service-oriented architecture are its support for the deployment and the

interaction of *loosely coupled* software systems, which evolve in a *dynamic* and *open* environment and can be composed with other services. Service requesters usually locate service providers dynamically during their execution using some service discovery protocol.

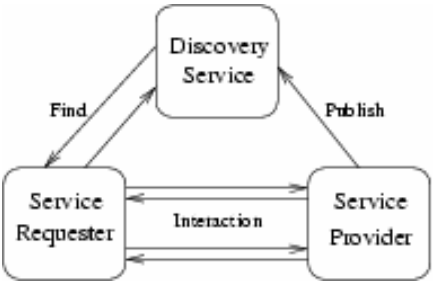

Fig. 1. Service-oriented architecture

A typical service-oriented architecture involving a service requester and a service provider is abstractly depicted in Figure 1. Localization of the service provider by the service requester is realized by querying a *discovery service*. Interactions are then as follows:

- The service provider deploys a service and publishes its description (the service contract) towards the discovery service.
- The service requester sends a query to the discovery service for locating a service satisfying its needs, which are defined with an abstract service contract, i.e., a service description that is not bound to any specific service instance.
- The discovery service returns to the service requester descriptions of available services, including their functional and non-functional interfaces. The requester then processes the description to get the messaging behavior supported by the service, that is, whether interactions should follow request-response, solicit-request, one-way messaging or even more complex interaction protocol, the structure of messages, as well as the concrete binding information such as the service's endpoint address.
- The service requester initiates interactions by sending a request message to the service.
- Interactions between the service requester and the service provider continue by exchanging messages following the agreed interaction protocol.

Note that the discovery service may be centralized (as depicted in Figure 1) or distributed (e.g., supported by all the service hosts), and may further adhere to either a passive (led by service provider) or active (led by service requester) discovery model. It is also important to note that the behavior of the interaction protocol between the service requester and provider may correspond to traditional communication protocols offered by middleware core brokers, but may as well realize a complex interaction protocol involving enhanced middleware-related services (e.g., replication, security, and transaction management) for the sake of quality of service.

The various refinements of the service-oriented software architectural style then lead to interoperability issue at the SOA level, possibly requiring interacting parties to compute and agree on the fly about a common discovery and communication protocol.

2.2 Key Properties of Service-Orientation

As previously stated, the benefit of service orientation for software system architectures lies in the looser coupling of the software components making up an application, hence the increased ability to making systems evolve as, e.g., application-level requirements change and the networked environment changes. Specifically, key properties of SOA with respect to openness include loose coupling, dynamicity and composability, as discussed below.

In a service-oriented architecture, services are provided by autonomously developed and deployed applications. In a dynamic and open system, designing tightly coupled services would compromise the services' respective autonomy, as they cannot evolve independently. Furthermore, failures would be more frequent in case of unavailability or failure of any of the composed applications. Instead, the service-oriented architecture focuses on loosely coupled services. Loosely coupled services depend neither on the implementation of another service (a requester or a third party constituent), nor on the communication infrastructure. To achieve interoperability among diversely designed and deployed systems, services expose a contract describing basically what the service provides, how a service requester should interact with the provider to get the service and the provided quality of service guarantees. Interactions between systems are done by message exchanges. This allows in particular defining asynchronous interactions as well as more complex message exchange patterns by grouping and ordering several one-way messages (e.g., RPC-like messaging by associating a request message with a response message). Moreover, the message structure should be independent of any programming language and communication protocol. A service requester willing to engage in an interaction with a service provider must be able – based solely on this contract – to decide if it can implement the requested interactions. The service contract comprises the functional interface and non-functional attributes describing the service, which is abstractly specified using a common declarative language processable by both parties. The service definition language should be standardized for increased interoperability among software systems that are autonomously developed and deployed. Indeed, the service definition language should not rely on any programming language used for implementing services, and the service being abstractly specified should be as independent as possible from the underlying implementation of the service. The service definition then describes functionalities offered by means of message exchanges, by providing the structure of each message and, optionally, ordering constraints that may be imposed on interactions involving multiple messages exchanges. Non-functional attributes may complement the functional interface by describing the provided support for QoS. Several non-functional properties may be here defined, such as security, availability, dependability, performance etc.

In a distributed open system, the system components and the environment evolve continuously and independently of each other. New services appear, existing services disappear permanently or get unavailable temporarily, services change their interfaces etc. Moreover, service requesters' functional or non-functional requirements may change over time depending on the context (i.e., both user-centric and computer-centric context). Adaptation to these changes is thus a key feature of the service-oriented architecture, which is supported thanks to service discovery and dynamic binding. To cope with the highly dynamic and unpredictable nature of service availability, services to be integrated in an application are defined using abstract service descriptions. Service requesters locate available services conforming to abstract service descriptions using a service discovery protocol, in general by querying a service registry. On the other hand, service providers make available their offered services by publishing them using the service discovery protocol. The published service descriptions contain the functional and non-functional interfaces of services, and provide as well concrete binding information for accessing the service such as the service's URI and the underlying communication protocol that should be used. Service discovery and integration of available concrete services are done either at runtime, or before the execution of interactions. Each interaction initiated by a service requester in a service-oriented architecture may thus involve different services or service providers, as long as the contract between the service provider and the service requester is implementable by both parties, i.e., the service description complies with the requirements of the service requester, which can in turn implement supported interactions of the service provider.

An advantage of describing services through well-defined interfaces is the possibility to compose them in order to build new services based on their interfaces, irrespective of technical details regarding their underlying platform and their implementation. A service built using service composition is called a composite service, and can in its turn, be part of a larger composition. The composition process is a complex task requiring integrating and coordinating diversely implemented services in a heterogeneous environment. It further requires dealing with the composition of QoS properties of individual services in order to provide a certain degree of QoS at the level of the composite service.

2.3 Software Technologies Enabling Service-Orientation

Compared to existing software technologies in the area of distributed computing, concepts introduced with the service-oriented architectural style are not new and can be implemented using various technologies. However, none of existing computing models or technologies do provide a complete solution. Furthermore, they often make assumptions that are not fully compatible with service-oriented computing concepts. A standardized model is also crucial to achieve the vision of service-oriented computing for providing full interoperability among autonomous components.

Object-oriented computing promotes the distinction between the implementation of a class and its public interface. However, there is a tight coupling between the interface of a class and its implementation. On the other hand, object-orientation

tends to build fine-grained classes with strong dependencies between them. Component-based systems do provide means for building composite systems out of independent building blocks that hide their implementation. However, component-based system integration is not appropriate for building dynamic systems, because of the strong dependencies upon available components at design time and of the interoperability issues between diversely implemented systems on heterogeneous platforms. Furthermore, interaction of components is often done using specific communication protocols, which is not always implementable by all parties. Distributed computing models such as CORBA[5] tried to fill this gap by enforcing interoperability by providing implementation-independent interfaces, standard communication protocols and a dynamic discovery service. However, strong assumptions made on related standards, like the specific communication protocol that is not easily implementable in all environments and the interface definition language that is tightly coupled with the type system of the service implementation, caused different vendors and developers to adopt custom and not standardized implementation decisions. While CORBA is widely used within single administrated domains, it failed to be adopted in the large scale.

The Web Services Architecture appears as the most compliant architecture to SOA principles, essentially due to its support for machine-readable, platform-neutral description languages using XML (eXtensible Markup Language), message-based communication that supports both synchronous and asynchronous invocations, and its adaptation to standard Internet transport protocols. According to the working definition of the W3C, a Web service is a software system identified by a URI, whose public interfaces and concrete details on how to interact with are described using XML-based languages. Using standardized specifications for defining Web services enforces interoperability among diversely implemented and deployed systems. In particular, Web service descriptions may be published and discovered by other software systems by querying common Web service registries. Systems may then interact in a manner prescribed by the service description, using XML-based messages conveyed by standard Internet transport protocols like HTTP. Web services can be implemented using any programming language and executed on heterogeneous platforms, as long as they provide the above features. This allows Web services owned by distinct entities to interoperate through message exchange. By providing standardized platform-neutral interface description languages, message-oriented communications using standard Internet protocols, and service discovery support, Web Services enable building service-oriented systems on the Internet. Although the definition of the overall Web Services Architecture is still incomplete, the base standards have already emerged from standardization consortiums such as W3C and Oasis[6], which define a core middleware for Web Services, partly building upon results from object-based and component-based middleware technologies. These standards relate to the specification of Web services and of supporting interaction protocols, referred to as conversation, choreography[7] or orchestration (see Figure 2).

[5] OMG Common Object Request Broker Architecture. http://www.corba.org.
[6] http://www.oasis-open.org/
[7] http://www.w3.org/2002/ws/chor

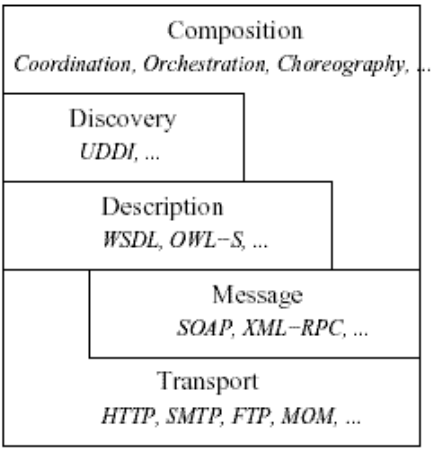

Fig. 2. Web Services Architecture

There is no single implementation of the Web Services Architecture. As Web Services refer to a group of related, emerging technologies aiming at turning the pervasive Web into a collection of computational resources, each with well-defined interfaces for their invocation, a number of implementation of these technologies are being introduced. Furthermore, Web Services are designed to be language and platform-independent, which leads to the implementation of a number of software tools and libraries easing the integration of popular software platforms into the Web Services Architecture and/or easing the development and enabling deployment of Web services in various environments. The interested reader is referred to Web sites keeping track of relevant implementations for an exhaustive list, and in particular the Xmethods site at http://www.xmethods.com/.

2.4 Semantic Modeling of Services

The World Wide Web contains a huge amount of information, created by multiple organizations, communities and individuals, with different purposes in mind. Web users specify URI addresses and follow links to browse this information. Such a simple access method explains the popularity of the Web. However, this comes at a price, as it is very easy to get lost when looking for information. The root of the problem is that today's Web is mainly syntactic. Documents structures are well defined but their content is not machine-processable. The Semantic Web specifically aims at overcoming this constraint. The "Semantic Web" expression, attributed to Tim Berners-Lee, envisages the future Web as a large data exchange space between humans and machines, allowing an efficient exploitation of huge amounts of data and various services. The semantic representation of Web pages' content will allow machines to understand and process this content, and to help users by supporting richer discovery, data integration, navigation, and automation of tasks.

To achieve the Semantic Web objectives, many Web standards are being used, and new ones are being defined. These standards may be organized in layers representing the Semantic Web structure, as shown in Figure 3. The *Unicode* and *URI* layers are the basic layers of the Semantic Web; they enforce the use of international characters, and provide means for object identification. The layer constituted of *XML*, *XML namespace* and *XML schema* allows a uniform structure representation for documents. By using *RDF* and *RDF Schema*, it is further possible to link Web resources with predefined vocabularies. The *ontology* layer is then based on RDF (Resource Description Framework) and RDF Schema, and allows the definition of more complex vocabularies, and relations between different concepts of these vocabularies. Finally, the *logic* and *proof* layers allow the definition of formal rules and the reasoning based on these rules.

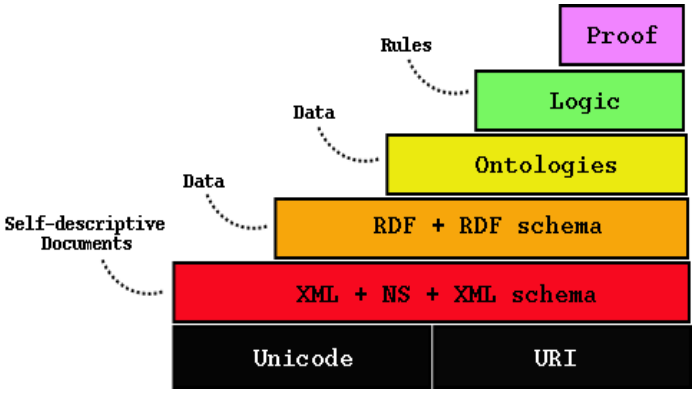

Fig. 3. Semantic Web structure

Specifically, RDF is a simple language allowing the semantic description of Web resources. This semantic description is specified as a triple in RDF. Such a triple is constituted of a subject, a predicate and an object. The subject is a link to the described resource. The predicate describes an aspect, a characteristic, an attribute, or a specific relation used to describe the resource. The object is an instance of a specific predicate used to describe a specific resource. Each piece of information in a triple is represented by a URI. The use of URIs ensures that the concepts that are used are not just structures stored in documents, but references to unique definitions accessible everywhere via the Web. For example, if one wants to access several databases storing persons' names and their addresses, and gets a list of the persons living in a specific district by using the postal code of the district, it is necessary to know for each database what are the fields representing the names and the postal codes. RDF allows specifying that: “(the field 5 of the database A)(is of type)(postal code)”, by using URIs for each term. RDF Schema is then a standard describing how to use RDF to define vocabularies, by adding to RDF the ability to define hierarchies, in terms of classes and properties. In RDF Schema, a class is a set of resources having similar characteristics, and the properties are relations that link the subject resources to the object ones.

In its origin, the term ontology is a philosophic term that means "the science of being". This term has been reused in computer science to express knowledge representation and the definition of categories. Ontologies describe structured vocabularies, containing useful concepts for a community that wants to organize and exchange information in a non-ambiguous manner. Thus, an ontology is a structured and coherent representation of concepts, classes, and relations between these concepts and classes pertaining to a vision of the world of a specific community. One of the most common goals in developing ontologies is for "sharing common understanding of the structure of information among people or software agents". According to the description given in [24], an ontology is a formal explicit description of concepts in a domain of discourse (classes, sometimes called concepts), properties of each concept describing various features and attributes of the concept (slots, sometimes called roles or properties), and restrictions on slots (facets, sometimes called role restrictions). An ontology together with a set of individual instances of classes constitutes a knowledge base.

One of the most widely used languages for specifying ontologies is the DAML+OIL language. DAML+OIL is the result of the fusion of two languages: DAML (Darpa Agent Markup Language) by the DARPA organization and OIL (Ontology Inference Layer) by European projects. Based on the DAML+OIL specification, the W3C has recently proposed the *Ontology Web Language (OWL)*, which has been used in introducing *Semantic Web Services*, as surveyed below. OWL is a one of the W3C recommendations related to the Semantic Web. More expressive than RDF Schema, it adds more vocabulary for describing properties and classes (such as disjointness, cardinality, equivalence). There are three sublanguages of OWL: OWL Lite, OWL DL and OWL Full. OWL Lite is the simplest one; it supports the basic classification hierarchy and simple constraints. OWL DL is named so, due to its correspondence with Description Logics[8]; it provides the maximum of OWL expressiveness, while guaranteeing completeness and decidability. OWL Full also provides the maximum of OWL expressiveness, but without computational guarantees. Thus, due to its syntactic freedom, reasoning support on OWL Full ontologies is less predictable compared to OWL DL.

OWL-S[9] (previously named DAML-S) is an OWL-based Web service ontology to describe Web services properties and capabilities, resulting from the work of many industrial and research organisms such as BBN Technologies, CMU, Nokia, Stanford University, SRI International and Yale University. OWL-S specifies a model for Web services semantic description, by separating the description of a Web services' capabilities from its external behavior and from its access details. Figure 4 abstractly depicts the model used in OWL-S. In this figure, we can see that a service description is composed of three parts: the service profile, the process model and the service grounding. The service profile describes the capabilities of the service, the process model describes the external behavior of the service, and the service grounding describes how to use the service.

[8] A field of research concerning logics that form the formal foundation of OWL.
[9] http://www.daml.org/services/

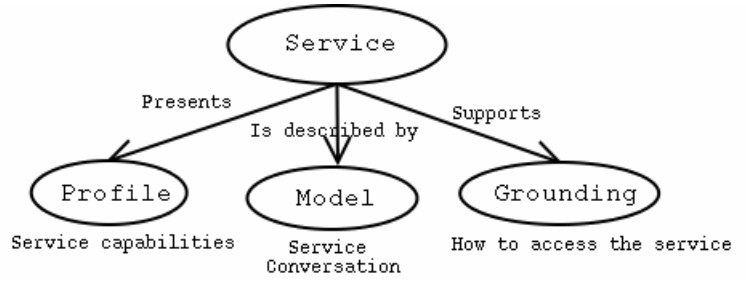

Fig. 4. OWL-S model

The service profile gives a high level description of a service and its provider. It is generally used for service publication and discovery. The service profile is composed of three parts:

- An informal description of the service oriented towards a human user; it contains information about the origin of the service, the name of the service, as well as a textual description of the service.
- A description of the services' capabilities, in terms of Inputs, Outputs, Preconditions and Effects (IOPE). The inputs and outputs are those exchanged by the service; they represent the information transformation produced by the execution of a service. The pre-conditions are those necessary to the execution of the service and the effects are those caused by the execution of the service; in combination, they represent the state change produced to the world by the execution of a service. Preconditions and effects are represented as logical formulas in an appropriate language.
- A set of attributes describing complementary information about the service, such as the service type, category, etc.

The process model is a representation of the external behavior – termed conversation – of the service as a process; it introduces a self-contained notation for describing process workflows. This description contains a specification of a set of sub-processes that are coordinated by a set of control constructs, such as a *sequence* or a *parallel* construct; these sub-processes are atomic or composite. The atomic processes correspond to WSDL operations. The composite processes are decomposable into other atomic or composite processes by using a control construct.

The service grounding specifies the information that is necessary for service invocation, such as the protocol, message formats, serialization, transport and addressing information. It is a mapping between the abstract description of the service and the concrete information necessary to communicate with the service. The OWL-S service grounding is based on WSDL. Thus, it introduces a mapping between high-level OWL classes and low-level WSDL abstract types that are defined by XML Schema.

3 Modeling Services for Mobile Computing

As already discussed, interoperability requirements of mobile distributed systems concern functional and non-functional interoperability that spans both middleware and application level; conformance relations enabling reasoning on interoperability are further required. As inferred from the survey of the previous section, the Service-Oriented Architecture with Web Services as its main representative, semantically enhanced by Semantic Web principles into Semantic Web Services, only partially address the interoperability requirements of mobile distributed systems.

On the other hand, mobile services may be conveniently modeled using concepts from the software architecture field: architectural components abstract mobile services and connectors abstract interaction protocols above the wireless network. Based on these concepts, we have addressed in [6] the composition of mobile distributed systems at both middleware and application level by modeling functional and non-functional properties of services and introducing conformance relations for reasoning on composability. Building on this work, we introduce in this section semantic modeling of mobile services so as to offer enhanced support to the interoperability requirements of mobile distributed systems. We focus on the functional behavior of services; semantic modeling of the non-functional behavior of services is part of our future work. Specifically, we introduce OWL-based ontologies to model mobile components and wireless connectors constituting mobile services. The reasoning capacity of OWL enables conformance relations for checking composability and interoperability methods for composing partially conforming services, as further detailed in Section 4. In our modeling, we have adopted some existing results from the OWL-S community [20]. Nevertheless, our approach is wider and treats in a comprehensive way the interoperability requirements of mobile distributed systems. In Section 5, we point out the enhanced features of our approach, comparing with OWL-S approaches.

In order to illustrate the exploitation of our model, we consider the example of an *e-shopping* service selling a specific type of goods, provided by a *vendor* component, which is normally stationary, hosted by some server. Mobile *customer* components hosted by wireless devices may access the vendor component over the wireless Internet to purchase goods on behalf of a human client.

3.1 Mobile Services

In traditional software architecture modeling, a service specifies the *operations* that it provides to and requires from the environment. The dynamic composition of the mobile service with peer networked services further requires enriching the service's functional specification so as to ensure adherence to the coordination protocols to be satisfied for ensuring correct service delivery despite the dynamics of the networks, i.e., the interaction protocols that must be atomic. The specification of coordination protocols among mobile services relates to the one of conversation or choreography in the context of Web Services. Such a specification also relates to the one of interaction protocols associated with component ports to ensure conformance with connector roles, as, e.g., supported by the Wright architecture description language [25].

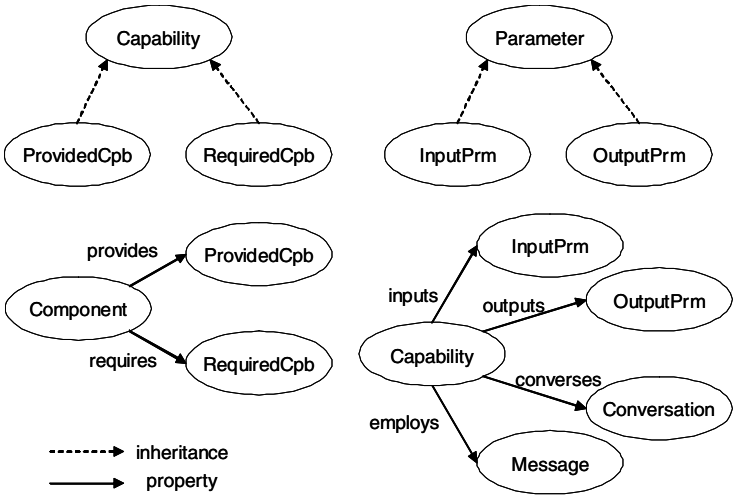

Fig. 5. Basic elements of the mobile service ontology

Building on the above fundamentals, we introduce a *mobile service* ontology to model the functional behavior of mobile services. The basic elements of this ontology are depicted in Figure 5. `Component` is the central class of the ontology representing the component realizing a mobile service. We introduce the notion of `Capability` for a component, which is a high-level functionality provided or required by the component, thus, refined as `ProvidedCpb` and `RequiredCpb`. A capability specifies a number of inputs and outputs, modeled as classes `InputPrm` and `OutputPrm`, which are derived from the parent class `Parameter`. We associate capabilities to distinct conversations supported by a component. Thus, `Capability` is related to `Conversation`, which contains the specification of the related conversation. `Capability` is further related to a set of messages employed in the related conversation; class `Message` is used to represent such messages. Conversations are specified as processes in the π-calculus [4], as introduced in [6].

We model communication between service components as exchange of one-way messages. This is most generic and assumes no specific interaction model, such as RPC or event-based, which is realized by the underlying connector. For example, in the case of RPC, communication between two peer components is based on the execution of operations that are provided by one peer and invoked by the other peer. Such an operation may be represented as the exchange of two messages, the first being the invocation of the operation and the second being the return of the result. Hence, we enrich our ontology to represent messages in a detailed manner, as depicted in Figure 6. Class `Message` is related to class `Parameter`, which represents all parameters carried by the message; members of the same class are the inputs and outputs of a capability, as defined above. As capability is a high-level functionality of the component, the inputs and outputs of a capability are a subset of all parameters of the messages employed within this capability. `Parameter` is associated to classes `PrmType`, `PrmValue` and `PrmPosition`; the latter denotes the position of the

parameter within the message. This representation of messages is most generic. A special parameter commonly carried by a message is an identifier of its function, i.e., what the message does. In the case of RPC, for example, this identifier is the name of the operation. We represent this identifier with the derived class `MsgFunction`.

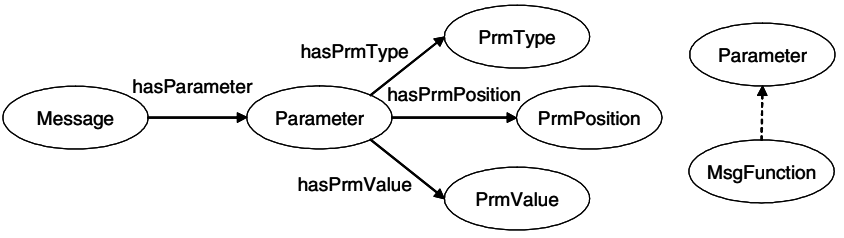

Fig. 6. Message modeling in the mobile service ontology

Based on the introduced mobile service ontology, a mobile service specification is as follows. For simplicity and space economy, we use – instead of the OWL notation – a simplified notation, only listing related OWL classes and their properties. Classes and instances of classes – termed *individuals* in OWL – are denoted by their first letter in uppercase, while properties are written in lowercase.

```
Component
   provides ProvidedCpb
   requires RequiredCpb
ProvidedCpb or RequiredCpb
   inputs InputPrm
   outputs OutputPrm
   converses Conversation
   employs Message
Message
   hasParameter MsgFunction
   hasParameter Parameter
MsgFunction or Parameter
   hasPrmType PrmType
   hasPrmPosition PrmPosition
   hasPrmValue PrmValue
```

Example. We now employ the elaborated mobile service ontology to model the vendor component involved in the e-shopping service of the example introduced above. We refine the mobile service ontology to produce the vendor ontology. Each class of the mobile service ontology is instantiated; the produced individuals constitute the vendor ontology. We assume that the vendor component supports the operations *browse()*, *book()* and *buy()*, which shall be realized as synchronous two-way interactions. From these operations we derive the messages supported by the vendor component, which we define as individuals of the class `Message`. For example, operation *browse()* produces the following listed messages, where parameters (`MsgFunction`

and Parameter individuals) of the messages are also specified. In our simplified notation, we use braces to denote that a class or individual is associated through a property to more than one other classes or individuals.

```
Message BrowseReq
   hasParameter BrowseReqFunc
   hasParameter ArticleInfo
Message BrowseRes
   hasParameter BrowseResFunc
   hasParameter {ArticleInfo, ArticleId, Ack}
```

BrowseReq is the input request message and BrowseRes is the output response message of the synchronous two-way interaction. The other two operations produce the following messages, where MsgFunction parameters have been omitted:

```
Message BookReq
   hasParameter ArticleId
Message BookRes
   hasParameter {ReservationId, Ack}
```

```
Message BuyReq
   hasParameter {ReservationId, CreditCardInfo, Ship-
pingInfo}
Message BuyRes
   hasParameter {ReceiptId, Ack}
```

Operation *browse()* allows browsing for an article by providing – possibly incomplete – information on the article; if this article is available, complete information is returned, along with the article identifier and a positive acknowledgement. Operation *book()* allows booking an article; a reservation identifier is returned. Operation *buy()* allows buying an article by providing credit card information and shipping information; a receipt identifier is returned. The vendor component supports further the operations *register_for_availability()* and *notify_of_availability()*, which shall be grouped in an asynchronous two-way interaction. These operations are encoded as follows:

```
Message RegisterForAvailabilityIn
   hasParameter {ArticleInfo, ReplyAddress}
```

```
Message NotifyOfAvailabilityOut
   hasParameter {ArticleInfo, SourceAddress}
```

The suffixes *in* and *out* have been added to these message names just to make clear the direction of the messages. The first operation or message allows registering for a specific article. When this article becomes available, a notification is sent back to the registered entity by means of the second operation or message. The vendor component and a peer customer component take care of correlating the two operations by including appropriate identifiers in the operations. Furthermore, we specify syntactic characteristics of the produced messages. For example, for message BrowseReq:

```
MsgFunction BrowseReqFunc
   hasPrmType string
   hasPrmPosition 1
   hasPrmValue "browse_req"
Parameter ArticleInfo
   hasPrmType some complex type
   hasPrmPosition 2
```

The supported messages are incorporated into the following specified two capabilities (`ProvidedCpb` individuals) provided by the vendor component. We specify the inputs (`InputPrm` individuals) and outputs (`OutputPrm` individuals) of these capabilities, as well as the associated conversations (`Conversation` individuals) described in the π-calculus. In the conversation specifications the following listed notation is used. For simplicity, we omit message parameters in the conversation specifications.

P, Q ::=		Processes
	P.Q	Sequence
	P|Q	Parallel composition
	P+Q	Choice
	!P	Replication
	v(x)	Input communication
	v[X]	Output communication

```
Component Vendor
   provides {Buy, Available}
ProvidedCpb Buy
   inputs {ArticleInfo, CreditCardInfo, ShippingInfo}
   outputs {ArticleInfo, ReceiptId, Ack}
   converses "
        BrowseReq().BrowseRes[].
        (
          !(BrowseReq().BrowseRes[]) +
          !(BrowseReq().BrowseRes[]).BookReq().BookRes[].BuyReq().BuyRes[]
        ) "
ProvidedCpb Available
   inputs ArticleInfo
   outputs ArticleInfo
   converses "RegisterForAvailabilityIn().NotifyOfAvailabilityOut[]"
```

An entity using capability `Buy` may either browse for articles several times, or browse several times and then book and buy an article. The inputs and outputs of `Buy` are a subset of all the parameters involved in the three included operations. A number of intermediate parameters, such as `ArticleId` and `ReservationId`, are further involved in the conversation; these are not visible at the level of capability `Buy`. An entity using capability `Available` registers and gets notified asynchronously of a newly available article.

It is clear from the example that most of the introduced classes of our ontology represent a semantic value that expresses the meaning of the specific class. For example,

giving the value `Buy` to `ProvidedCpb`, we define the semantics of the specific capability provided by the vendor component, as long as we can understand the meaning of `Buy`. The only classes that do not represent a semantic – according to the above definition – value are `Conversation`, which is a *string* listing the π-calculus description of the related conversation; `PrmPosition`, which is an *integer* denoting the position of the related parameter within the message; and `PrmValue`, which is the actual value of the parameter. Incorporating these non-semantic elements into our ontology allows an integrated modeling of mobile services with minimum resorting to external formal syntactic notations, as the π-calculus.

3.2 Wireless Connectors

In the mobile environment, connectors specify the interaction protocols that are implemented over the wireless network. This characterizes message exchanges over the transport layer to realize the higher-level protocol offered by the middleware, on top of which the mobile component executes. In addition, the dynamic composition of mobile services leads to the dynamic instantiation of connectors. Hence, the specification of wireless connectors is integrated with the one of mobile services (actually specifying the behavior of connector roles), given that the connectors associated with two interacting mobile services must compose.

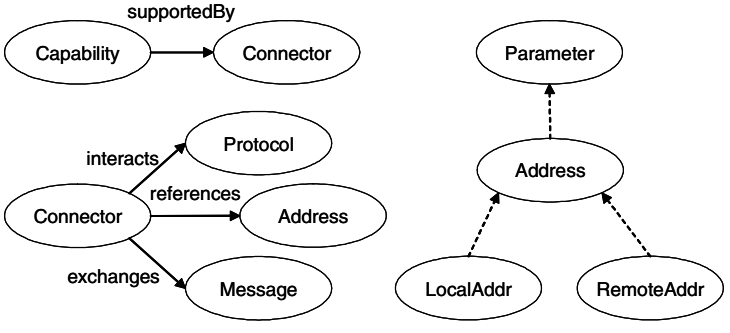

Fig. 7. Connector modeling in the mobile service ontology

To integrate connectors in the so far elaborated mobile service model, we extend the mobile service ontology with a number of new classes, as depicted in Figure 7. `Capability` is related to class `Connector`, which represents the specific connector used for a capability; we assume that a capability relies on a single connector, which is a reasonable assumption. A connector realizes a specific interaction protocol; this is captured in the relation of `Connector` to class `Protocol`, which contains the specification of the related interaction protocol. Interaction protocols are specified as processes in the π-calculus.

An interaction protocol realizes a specific interaction model for the overlying component, such as RPC or event-based. This interaction model is implicitly specified in the π-calculus description of the interaction protocol. Nevertheless, the interaction

model may additionally be semantically represented by class `Connector`. As there is a large variety of connectors and associated interaction models [21], there is no meaning in enriching the generic mobile service ontology with a taxonomy of connectors. Class `Connector` may be associated to external ontologies on a case by case basis to represent the interaction model supported by a specific connector.

Furthermore, a connector supports an addressing scheme for identifying itself as well as its associated component over a network realized by the underlying transport layer. A number of different approaches are allowed here, depending on the addressing functionality already supported by the transport layer and on the multiplexing capability of the connector, i.e., its capability to support multiple components. The latter further relates to a connector acting as a container for components, e.g., a Web server being a container for Web applications. Thus, considering the Web Services example, we may distinguish the following addressing levels:

- The TCP/IP transport layer supports IP or name addressing of host machines.
- A Web Services SOAP/HTTP connector binds to a specific TCP port; in this case the transport layer specifies an addressing scheme for the overlying connectors.
- The SOAP/HTTP connector supports addressing of multiple Web service components, treating Web services as Web resources; thus, incorporating the underlying IP address & port number addressing scheme, the SOAP/HTTP connector supports URI addressing.

To be most generic, we enable a connector addressing scheme without assuming any connector addressing pre-specified by the transport layer. This scheme shall incorporate the established transport layer addressing. Moreover, this scheme shall integrate component identifiers for distinguishing among multiple components supported by a single connector, when this is the case. The introduced generic scheme is represented by the relation of `Connector` to class `Address`. Thus, `Address` represents a reference of a mobile service component accessible through a specific connector and underlying transport layer. `Address` is a subclass of `Parameter`.

`Connector` is further related to a set of messages exchanged in the related interaction protocol, which are members of the class `Message`. This is the same generic class used for component-level messages, as it also applies very well to connector-level messages. Communication between connectors can naturally be modeled as exchange of one-way messages; this takes place on top of the underlying transport layer. To enable component addressing, connector-level messages integrate addressing information. To be most generic, we enable connector-level messages to carry complete addressing information, assuming no addressing information added by the transport layer; certainly, this scheme may easily be adapted according to the addressing capabilities of the transport layer. We introduce two subclasses of `Address`, named `LocalAddr` and `RemoteAddr`, which represent the local address and remote address information included in a connector-level message exchanged between two peer connectors. Remote address information is used to route the message to its destination, while local address information identifies the sender and may be used to route back a possible response message.

According to the distinction introduced in the previous section, only `Protocol` does not represent a semantic value among the new classes of our ontology. Based on

the extended mobile service ontology, a mobile service specification is extended as follows to integrate connectors:

```
ProvidedCpb or RequiredCpb
   supportedBy Connector
Connector
   interacts Protocol
   references Address
   exchanges Message
Message
   hasParameter LocalAddr
   hasParameter RemoteAddr
```

Example. We now complete the modeling of the vendor component based on the extended mobile service ontology. As specified in the previous section, the vendor component relies on two connectors, one supporting synchronous two-way interactions and one supporting asynchronous two-way interactions. By properly instantiating class `Connector` and its associated classes, we can model the two required connectors, thus completing the vendor ontology. We define two individuals of `Connector`:

```
Connector VConn1
  interacts "vreq(vreq_prm).vres[VRES_PRM]"
  references VAddr
  exchanges {VReq, VRes}
Connector VConn2
  interacts "vreq(vreq_prm)", "vres[VRES_PRM]"
  references VAddr
  exchanges {VReq, VRes}
Address VAddr
   hasPrmType URL
   hasPrmValue "http://www.e-shopping.com:8080/vendor"
```

Both connectors exchange a request and a response message. For connector `VConn1`, the emission of the response message is synchronous, following the reception of the request message; while for connector `VConn2`, the emission of the response message is asynchronous, not coupled with the reception of the request message. Both connectors enable addressing the vendor component with a URL address following the scheme *http://<host>:<port>/<path>*. Each connector supports a specific capability of the vendor component:

```
ProvidedCpb Buy
   supportedBy VConn1
ProvidedCpb Available
   supportedBy VConn2
```

Furthermore, we specify the characteristics of messages `VReq` and `VRes`. For example, for message `VReq`, which is input by the vendor component:

```
Message VReq
   hasParameter VReqFunc
   hasParameter {VLocalAddr, VRemoteAddr}
   hasParameter VReqPrm
MsgFunction VReqFunc
   hasPrmType byte
   hasPrmPosition 1
   hasPrmValue 7Ah
RemoteAddr VRemoteAddr
   hasPrmType URL
   hasPrmPosition 3
   hasPrmValue "http://www.e-shopping.com:8080/vendor"
LocalAddr VLocalAddr
   hasPrmType URL
   hasPrmPosition 2
Parameter VReqPrm
   hasPrmType hex
   hasPrmPosition 4
```

`PrmValue` for `VLocalAddr` will be determined by the peer connector – supporting a customer component – sending the request message. `PrmType` *hex* of `VReqPrm` determines the encoding of the component-level message (e.g., an invocation of a remote operation) carried by the connector-level request message. This further corresponds to the serialization of remote method invocations performed by a middleware platform.

4 Semantics-Based Interoperability

Given the above functional specification of mobile services and related wireless connectors, functional integration and composition of mobile services in a way that ensures correctness of the mobile distributed system may be addressed in terms of conformance of respective functional specifications. Conformance shall be checked both at component and at connector level; for two services to compose, conformance shall be verified at both levels. To this end, we introduce a conformance relation for each level. To allow for the composition of heterogeneous mobile services, our conformance relations enable identifying partial conformance between components and between connectors. Then, we employ appropriate interoperability methods at each level to ensure composition of heterogeneous components and connectors; for two services to compose, interoperability must be established at both levels.

Our conformance relations and interoperability methods exploit our ontology-based modeling of mobile services. As detailed in Section 3, the introduced mobile service ontology enables representing semantics of components and connectors. Nevertheless, to enable a common understanding of these semantics, their specification shall build upon possibly existing globally shared ontologies. Incorporating external

commonly shared ontologies serve two purposes: (i) these ontologies are used as common vocabulary for interpreting mobile services' semantics; and (ii) these ontologies may be used to extend the mobile service ontology to enable a more precise representation of mobile services' semantics. OWL targeting the semantic Web provides inherent support to the distribution of ontologies enabling the incremental refinement of ontologies based on other imported ontologies. Further, employing OWL to formally describe semantics allows for automated interpretation and reasoning on them, thus enabling conformance checking and interoperability.

In the following, we introduce our solution to interoperability at connector and at component level, specifying conformance relation and interoperability method for each level. We first address connector level, as this constitutes the base for service interoperability.

4.1 Interoperability at Connector Level

Based on our functional modeling of wireless connectors, a connector (`Connector`), realizes an interaction protocol (`Protocol`) specified as a process in the π-calculus, establishes an addressing scheme (`Address`) described by a complex data structure (`Parameter`), and employs a number of messages (`Message`) described as complex data structures (`Parameter`). These classes are complementary or may even overlap in specifying a connector. For example, we may associate class `Connector` to external ontologies representing some features not, partially or even fully specified by the other classes; in this way, we may, for example, represent with `Connector` the interaction model realized by the connector, such as RPC or event-based. This redundancy may be desirable in order to facilitate the conformance relation or the interoperability method described in the following.

Conformance relation. We introduce a conformance relation for connectors based on the above classes. As already discussed, we specify a connector by instantiating these classes into individuals specific to this connector. Two connectors may be composed if they (at least partially) conform to each other in terms of their corresponding individuals for all the above classes. The definition of partial conformance depends on the capacity to deploy an adequate interoperability method to compensate for the non-conforming part.

Conformance in terms of interaction protocols is checked over the associated π-calculus processes, as detailed in [6]; this implicitly includes the realized interaction models. For interaction models, conformance may alternatively be asserted by semantic reasoning on the related individuals of class `Connector`. In the same way, for addressing schemes, exchanged messages, parameters of messages and types of parameters, conformance may be asserted by semantic reasoning on the related individuals of classes `Address`, `Message`, `Parameter` and `PrmType`. Finally, to ensure syntactic conformance in exchanged messages, the specific values of `PrmPosition` and `PrmValue` shall be the same for the two connectors.

Interoperability method. To compose non-absolutely conforming connectors, an appropriate interoperability method shall be employed. We employ a *connector customizer* that serves as an intermediate for the message exchange between the two connectors. The customizer has access to the ontologies of the two connectors, and from there to the parent mobile service ontology and the possibly incorporated external ontologies. The customizer shall take all appropriate action to remedy the incompatibilities between the two connectors. For example, upon reception of a message, the customizer shall interpret it and perform all necessary conversions to make it comprehensible to the other peer. The connector customizer may be collocated with one of the two peers or be located on an intermediate network node, depending on architectural requirements; for example, for wireless ad hoc computing environments the former solution is more appropriate.

Example. We now concretize the above outlined conformance relation and interoperability method for the e-shopping service example. In Section 3, we specified the vendor ontology defining the vendor component and its associated connectors. The vendor component provides its services to customer components.

To enable a more precise representation of connector semantics for the vendor and customer components, we assume the existence of an external *remote operation connector* ontology, which defines a simple taxonomy of connectors supporting remote operation invocation. This ontology provides a common vocabulary for connectors of this type. This ontology is outlined in the following:

```
RemoteOperationConn
   hasLegs {OneWay, TwoWay}
   hasSynchronicity {Sync, Async}
   keepsState {State, NoState}
```

Class `RemoteOperationConn` is related to three other classes, which are defined above by enumeration of their individuals. Property `hasLegs` determines whether a connector supports one-way or two-way operations; `hasSynchronicity` determines whether a connector supports synchronous or asynchronous operations; finally, `keepsState` determines whether a connector maintains state during the realization of an operation, e.g., for correlating the request and response messages of an asynchronous operation. We additionally pose the restriction that each one of the three above properties has cardinality *exactly one*, which means that any `RemoteOperationConn` individual has exactly one value for each of the three properties.

We further refine the remote operation connector ontology to identify a number of allowed combinations of the above properties, which produces a number of feasible connector types specified by the ontology. Hence, the following subclasses of `RemoteOperationConn` are defined:

```
SyncConn
   hasLegs TwoWay
   hasSynchronicity Sync
   keepsState State
```

```
AsyncStateConn
   hasLegs TwoWay
   hasSynchronicity Async
   keepsState State

AsyncNoStateConn
   hasSynchronicity Async
   keepsState NoState
```

In the above definitions, properties in boldface are set to be a *necessary and sufficient condition* for identifying the associated connector class. For example, a synchronous connector has synchronicity `Sync`, and synchronicity `Sync` is sufficient to identify a synchronous connector. `NoState` for an asynchronous connector means that the communicating components take care of correlating the request and response messages of an asynchronous operation. In this case, it makes no difference whether an asynchronous connector is one-way or two-way. Thus, `hasLegs` is left undefined in `AsyncNoStateConn`; it may take any of the two values `OneWay` or `TwoWay`.

We now exploit the above ontology to specify interaction model semantics for the two connectors supporting communication between the vendor component and a specific customer component. To this end, the two connectors inherit from both the mobile service and remote operation connector ontologies. More specifically, the two connectors are represented by two classes that are subclasses of both `Connector` and `RemoteOperationConn`, which means that they inherit properties of both classes:

```
VendorConn
   hasLegs TwoWay
   hasSynchronicity Async
   keepsState NoState

CustomerConn
   hasLegs OneWay
```

These two connector classes are defined independently, each one by the designer of the related connector, and make part of the vendor and customer ontologies, correspondingly, which are normally local to the related components and connectors. Here, the two designers have opted not to reuse any of the specialized connector classes, pre-defined in the remote operation connector ontology; they have instead defined two new connector classes. We can see that class `VendorConn` represents the features required by the `Connector` individual `VConn2` defined in Section 3.2. Employing an OWL reasoning tool, an inference about conformance between `VendorConn` and `CustomerConn` may be drawn as follows:

`VendorConn` has both property values `Async` and `NoState`, which makes it necessarily an `AsyncNoStateConn`. `CustomerConn` must have exactly one value for each of the two undefined properties. Its synchronicity cannot be `Sync`, because this would make `CustomerConn` necessarily a `SyncConn`, which, however, is two-way, while `CustomerConn` is one-way. Thus, `CustomerConn` has property value `Async`. In the same way, its state property cannot be `State`, because

this together with Async would make it necessarily an AsyncStateConn, which also is two-way. Thus, CustomerConn has property value NoState. Property values Async and NoState make CustomerConn necessarily an Async-NoStateConn. Thus, VendorConn and CustomerConn belong to the same connector class within the remote operation connector ontology, which makes them conforming in terms of their supported interaction models.

In the above, interaction model conformance was asserted by comparing semantics co-represented by class Connector of the mobile service ontology (together with class RemoteOperationConn of the remote operation connector ontology). Conformance between VendorConn and CustomerConn shall be further checked in terms of all the other classes of the mobile service ontology. We instantiate Vendor-Conn and CustomerConn to define the rest of their characteristics according to this ontology:

```
VendorConn VConn2
(as specified in Section 3.2)

CustomerConn CConn2
  interacts "cout[COUT_PRM]", "cin(cin_prm)"
  references CAddr
  exchanges {COut, CIn}
Address CAddr
   hasPrmType URL
   hasPrmValue some URL
Message COut
   hasParameter COutFunc
   hasParameter {CLocalAddr, CRemoteAddr}
   hasParameter COutPrm
MsgFunction COutFunc
   hasPrmType word
   hasPrmPosition 1
   hasPrmValue 3FEDh
RemoteAddr CRemoteAddr
   hasPrmType URL
   hasPrmPosition 2
   hasPrmValue "http://www.e-shopping.com:8080/vendor"
LocalAddr CLocalAddr
   hasPrmType URL
   hasPrmPosition 3
Parameter COutPrm
   hasPrmType bin
   hasPrmPosition 4
```

Interaction protocol conformance for VConn2 and CConn2 is checked over the associated π-calculus processes, which are obviously complementary (see [6]); however, different names are used for messages VReq-COut, VRes-CIn and for message parameters VReqPrm-COutPrm, VResPrm-CInPrm. Semantic conformance

between corresponding messages and parameters is asserted by using external ontologies, as already done for semantic conformance between interaction models. In the same way, semantic conformance is asserted between addressing schemes (`VAddr`-`CAddr`).

Thus, the conformance relation applied to the current example requires: (i) semantic conformance between interaction models, addressing schemes, messages and message parameters; and (ii) workflow conformance between interaction protocols.

Nevertheless, there are still incompatibilities between `VConn2` and `CConn2` in terms of types of parameters (e.g., between `VReqPrm` and `COutPrm`), position of parameters within messages (e.g., between `VRemoteAddr` and `CRemoteAddr` within `VReq` and `COut`), and values of parameters (e.g., between `VReqFunc` and `COutFunc`). Further, referenced types such as *URL*, *byte*, *word*, *hex* and *bin* may not belong to the same type system. Thus, we employ a connector customizer which resolves these incompatibilities by (i) converting between types by accessing some external type ontology; if different type systems are used, external ontologies can help in converting between type systems; (ii) modifying position of parameters; and (iii) modifying values of parameters. This customizer exploits the semantic conformance established above to identify the semantically corresponding messages and message parameters of `VConn2` and `CConn2`.

A weaker conformance relation than the one applied to this example would require a more competent interoperability method, e.g. a connector customizer capable of resolving incompatibilities in addressing schemes or even in interaction models and workflows of interaction protocols. The feasibility of such cases depends on the nature of addressing schemes or interaction protocols and the degree of heterogeneity, and shall be treated on a case-by-case basis. Enabling automated, dynamic configuration or even generation of the appropriate interoperability method from some persistent registry of generic interoperability methods is then a challenging objective. Ontologies could then be used to represent generic interoperability methods and to guide the automated generation or configuration of these methods based on the concrete ontologies of the two incompatible connectors.

4.2 Interoperability at Component Level

Based on our functional modeling of mobile components, a component provides or requires a number of capabilities (`ProvidedCpb`, `RequiredCpb`). Each capability has a number of inputs (`InputPrm`) and outputs (`OutputPrm`) described as complex data structures (`Parameter`), realizes a conversation (`Conversation`) specified as a process in the π-calculus, and employs a number of messages (`Message`) described as complex data structures (`Parameter`). Based on the similarity of capability `Conversation` to connector `Protocol` and on the common use of `Message` by both capabilities and connectors, we could introduce a conformance relation and associated interoperability method for component capabilities similar to the ones elaborated for connectors. Nevertheless, considering the diversity of component capabilities and conversations, requiring workflow conformance between component conversations and semantic conformance for each single message and message parameter

– as for the two connectors in the example above – is too restrictive. Moreover, the introduced connector-level interoperability method, based on communication interworking, cannot deal with the high heterogeneity of components, e.g., it cannot resolve highly incompatible component conversations. Therefore, we introduce a more flexible, coarse-grained approach for component conformance and interoperability based on component capabilities.

Conformance relation. Our high-level conformance relation for components states that two components may be composed if they require and provide in a complementary way semantically conforming capabilities. We model a capability by instantiating classes `ProvidedCpb` or `RequiredCpb`, `InputPrm` and `OutputPrm` into individuals specific to this capability. Semantic conformance between two capabilities is asserted by reasoning on their corresponding individuals. As already detailed for connectors, these individuals shall as well inherit from external ontologies; this allows a rich representation of capabilities based on common vocabularies, which enable their interpretation and conformance checking.

Depending on the existence of external ontologies, capabilities may be directly provided with semantics (class `ProvidedCpb` or `RequiredCpb`). Alternatively, capabilities may be semantically characterized by the semantics of their inputs and outputs (classes `InputPrm` and `OutputPrm`). As discussed in [26] for Semantic Web Services capabilities, the latter approach requires a reduced set of ontologies, as inputs and outputs may be combined in many diverse ways to produce an indefinite number of capabilities. However, semantically characterizing a capability based only on its inputs and outputs may produce ambiguity and erroneous assertions, e.g., when checking conformance between capabilities. We opt for a hybrid approach, where, depending on the availability of related ontologies, both capability semantics and input/output semantics are used. As presented in Section 2.4, OWL-S identifies Web services by their inputs and outputs, enhanced by preconditions and effects. This enables a more precise representation of a service. We consider integrating preconditions and effects into our model as part of our future work.

Our conformance relation adopts the approach presented in [10] for matching Semantic Web services' capabilities, which identifies several degrees of matching: (i) *exact*; (ii) *plug in*, where the provided capability is more general than the requested one, thus it can be used; (iii) *subsume*, where the provided capability is more specific than the requested one, thus it may be used in combination with another Web service complementing the missing part; and (iv) *fail*. As we are assessing conformance between two peer components, we exclude case (iii). Our conformance relation requires that inputs of a required capability be a superset of inputs of the associated provided capability, while outputs of a required capability be a subset of outputs of the associated provided capability. This refers to both the number of equivalent inputs and outputs and to subsumption relations between mapped inputs and outputs. Equivalence and subsumption are asserted by semantic reasoning, where the degree of similarity may be measured as the distance between concepts in an ontology hierarchy. This approach ensures that a service is fed at least with all the needed input and produces at least all the required output.

Interoperability method. To compose the high-level-conforming components resulting from the introduced conformance relation, an appropriate interoperability method shall be employed. To this end, we intervene in the execution properties of the component requiring the specific capability. First, the component providing the specific capability is a normal component, the executable of which integrates the hard-coded implementation of the conversation and messages associated to the capability. Thus, this component exposes a normal specific functional interface. Regarding the component requiring the specific capability, its executable is built around this capability, which may be represented as a high-level local function call. This component integrates further an *execution engine* able to execute on the fly the specific conversation associated to this capability and supported by its peer component. Thus, this component comprises a specific part implementing the component logic that uses this capability, and a generic part constituting a generic interface capable of being composed with diverse peer interfaces. The execution engine shall be capable of**:**

- Executing the declarative descriptions of conversations; to this end, execution semantics of the π-calculus descriptions are employed;
- Parsing the incoming messages and synthesizing the outgoing messages of the conversation based on the syntactic information provided by classes `PrmType`, `PrmPosition` and `PrmValue`; access to an external type ontology may be necessary if the type system of the peer is different to the native type system;
- Associating the inputs and outputs of the required capability to their corresponding message parameters; this is based on semantic mapping with the inputs and outputs of the remote capability, which are directly associated to message parameters; conversion between different types or between different type systems may be required.

It is clear from the above that for components it is not necessary to provide messages and message parameters – at least parameters that are not capability inputs or outputs – with semantics.

The introduced component-level interoperability method shall be employed in combination with the connector-level interoperability method discussed above to ensure service interoperability. It is apparent from the above that the component-level method is more adaptive and can resolve higher heterogeneity than the connector-level one, which is appropriate for components, considering their diversity. On the other hand, the connector-level method permits lower heterogeneity, which is normal for connectors, which shall not be allowed to deviate significantly from the behavior expected by the overlying component. By locating the connector customizer on the side of the component requiring a specific capability, this component becomes capable of adapting itself at both component and connector level to the component providing the specific capability. Employing dynamic schemes for the instantiation of connectors as the one outlined in the previous section would make this adaptation totally dynamic and ad hoc.

Example. We will now complete the e-shopping service example by applying the introduced component-level conformance relation and interoperability method. In Section 4.1, we specified connector CConn2 within the customer ontology. We complete the customer ontology by defining capabilities for the customer component and a second connector. As discussed above, the customer component will be specified only at capability level. We assume that the customer component requires the capabilities Get and NewRelease, which also concern buying an article and registering for notification of new releases of articles.

```
Component Customer
   requires {Get, NewRelease}
RequiredCpb Get
   inputs {ArticleData, Address, PaymentData, Customer-
Profile}
   outputs {ArticleData, Ack}
RequiredCpb NewRelease
   inputs ArticleData
   outputs ArticleData
```

To assert conformance between the customer and the vendor component with respect to capabilities Get and Buy or NewRelease and Available, semantic matching shall be sought for the compared capabilities and their inputs and outputs.

We discuss the case of Get and Buy. We assume that there exists a *commerce* ontology specifying among other the class Purchase, as one of the activities included in commerce. Furthermore, we assume the existence of a specialized ontology describing the specific articles being sold by the vendor component and possibly sought by the customer component. Finally, a *payment information* ontology – describing payment methods, such as by credit card, by bank transfer, etc. – and a *location information* ontology are available. Having – independently – defined capabilities Get and Buy as direct or less direct descendants of class Purchase enables the assertion of their conformance. In the same way, ArticleData may be mapped to ArticleInfo if the vendor component sells what the customer component seeks to buy; the same for the couple Address-ShippingInfo. PaymentData can be found to be more general than CreditCardInfo in the payment information ontology. This means that the customer component is capable of managing as well other payment methods than by credit card, which is required by the vendor component. This is in accordance with our conformance relation. We may further see that Get additionally inputs CustomerProfile, which is not required by Buy, and Buy additionally outputs ReceiptId, which is not required by Get. This, too, is in accordance with our conformance relation.

To be able to use the remote capability Buy, the customer component shall have a connector (e.g., CConn1) conforming to VConn1. Then, the customer component will execute the declarative conversation associated to Buy in the way detailed above.

5 Related Work

In the last couple of years there has been extensive research towards semantic modeling of services. This research has mostly been focused on adding semantics to Web Services, which, as presented in Section 2.3, is the dominant paradigm for service-oriented architectures on the Web. Hence, there are a number of efforts towards Semantic Web Services. The most complete effort concerns OWL-S, which was outlined in Section 2.4. In this section, we compare our approach with OWL-S and discuss OWL-S-based and non-OWL-S-based efforts.

OWL-S defines an ontology for semantically describing Web Services in order to enable their automated discovery, invocation, composition and execution monitoring. From our standpoint, this may be regarded as enabling application-level interoperability. Our work has aimed at introducing semantic modeling of mobile services in order to deal with the interoperability requirements of mobile distributed systems. This has led us to elaborate a comprehensive modeling approach that spans both the application and middleware level. Furthermore, our modeling considers services from a software architecture point of view, where services are architecturally described in terms of components and connectors. This abstracts any reliance on a specific technology, as on Web Services in the OWL-S case. We compare further our approach with OWL-S in the following.

Our modeling of provided capabilities along with their inputs and outputs may be mapped to the OWL-S service profile. Both describe the high-level functionalities of services and may be used for discovering services, thus, for matching or conformance verification. We additionally explicitly model required capabilities for a component, which is done implicitly in OWL-S, e.g., for an agent contacting Web services. As further discussed in Section 4.2, OWL-S enhances the description of capabilities with preconditions and effects, which we consider integrating into our approach.

Our modeling of conversation and component-level messages may be mapped to the OWL-S process model. We have opted for a well-established process algebra, such as the π-calculus, which allows dealing with dynamic architectures [27] and provides well-established execution semantics. The OWL-S process model provides a declarative, not directly executable specification of the conversation supported by a service. One has to provide external execution semantics for executing a process model, which has been done, for example, in [22]. The OWL-S process model decomposes to atomic processes, which correspond to WSDL operations. Our modeling employs component-level messages, which make no assumption of the underlying connector. The types of the inputs and outputs of an OWL-S atomic process are made to correspond to WSDL types, which are XML Schema types. This restricts the employed type system to the XML Schema type system. Our approach enables using different type systems, and, further, heterogeneous type systems for the two peer components.

Our modeling of connectors may be mapped to the OWL-S grounding. The OWL-S grounding is restricted to the connector types specified by Web Services, which comprise an interaction model prescribed by WSDL on top of the SOAP messaging protocol, commonly over HTTP. As WSDL 2.0 has not yet been finalized, the current

version of OWL-S relies on WSDL 1.1, which supports only two-way synchronous operations and one-way operations. The WSDL 1.1 interaction model does not support, for example, two-way asynchronous interactions or event-based interactions, as has been indicated in [1]. WSDL 2.0 will allow greater flexibility in its interaction model. Nevertheless, our approach enables the use of any connector type, which is modeled by the connector-level part of our mobile service ontology; this allows any interaction model, interaction protocol and addressing scheme. Finally, our approach enables using different type systems for connectors and, further, heterogeneous type systems for the two peer connectors, while WSDL and SOAP rely on the XML Schema type system.

Work by Carnegie Mellon University described in [26] is the most complete effort up to now in the OWL-S community; the authors have realized an OWL-S based architecture for automated discovery and interaction between autonomous Web services [19]. Discovery is based on the matching algorithm detailed in [10], which has been adopted by several other efforts in the literature. The main features of this algorithm were discussed in Section 4.2; as stated there, our component-level conformance relation incorporates some of the principles of this work. However, this matching algorithm does not exploit the full OWL-S representation of services in terms of inputs, outputs, preconditions and effects; preconditions and effects are not employed. Interaction between autonomous Web services is based on an OWL-S (formerly DAML-S) virtual machine [28], which is capable of executing OWL-S process model descriptions. As mentioned above, execution is based on the execution semantics defined by the authors in [22]. The virtual machine integrates OWL reasoning functionality to be able to interpret and synthesize messages. Its implementation is based on the DAML-Jess-KB [14], an implementation of the DAML (a predecessor of OWL) axiomatic semantics that relies on the Jess theorem prover [13] and the Jena parser [15] to parse ontologies and assert them as new facts in the Jess Knowledge Base. Our component-level interoperability method employing an execution engine capable of executing the π-calculus descriptions of service conversations can certainly build upon tools and experience coming from this work. Nevertheless, as it realizes a more general conceptual model, our approach addresses also connector-level interoperability.

In the work presented in [18], the authors elaborate an ontology-based framework for the automatic composition of Web Services. They define an ontology for describing Web services and specify it using the DAML+OIL language (a predecessor of OWL). They further propose a composability model based on their service ontology, for comparing the syntactic and semantic features of Web services to determine whether two services are composable. They identify two sets of composability rules. Syntactic rules include: (i) *mode composability*, which compares operation modes as imposed by WSDL, that is, two-way synchronous operations and one-way operations; and (ii) *binding composability*, which compares the interaction protocols of communicating services, e.g., SOAP. Semantic rules include (i) *message composability*, which compares the number of message parameters, their data types, business roles, and units, where business roles and units represent semantics of parameters; (ii) *operation semantics composability*, which compares the semantics of service operations; (iii) *qualitative composability*, which compares quality of service properties of Web services; and (iv) *composition soundness*, which semantically assesses whether com-

bining Web services in a specific way is worthwhile. The introduced service ontology resembles our mobile service ontology, while it additionally represents quality of service features of services. However, what is lacking is representation of service conversations; actually, in this approach, services are implicitly considered to support elementary conversations comprising a single operation. These operations are employed into an external workflow to provide a composite service produced with a development time procedure. Additionally, there is no attempt to provide interoperability in case that the composability rules identify incompatibilities. Composability rules are actually used for matching existing services to requirements of the composite service. Same as the other approaches adding semantics to Web services, this approach treats only application-level composability.

6 Conclusion

Mobile distributed systems are characterized by a number of features, such as the highly dynamic character of the computing and networking environment due to the intense use of the wireless medium and the mobility of devices; the resource constraints of mobile devices; and the high heterogeneity of integrated technologies in terms of networks, devices and software infrastructures. To deal with high dynamics, mobile distributed systems tend to be dynamically composed according to the networking of mobile services. Nevertheless, such a composition must be addressed in a way that enforces correctness of the composite systems with respect to both functional and non-functional properties and deals with the interoperability issue resulting from the high heterogeneity of integrated components. The Semantic Web paradigm has emerged as a decisive factor towards interoperability, which up to then was being pursued based on agreements on common syntactic standards; such agreements cannot scale in the open, highly diverse mobile environment. Related efforts elaborating semantic approaches are addressing application-level interoperability in terms of information and functionality. However, interoperability requirements of mobile distributed systems are wider, concerning functional and non-functional interoperability that spans both middleware and application level.

Towards this goal, we have introduced semantic modeling of mobile services based on ontologies, addressing functional properties of mobile components and associated wireless connectors. We have further elaborated conformance relations over component and connector models so as to be able to reason on the correctness of the composition of peer mobile services with respect to offered functional properties. Our conformance relations enable identifying partial conformance between components and between connectors, thus reasoning on interoperability. Based on these conformance relations, we have further specified appropriate interoperability methods to realize composition and interoperation of heterogeneous mobile services. Nevertheless, our modeling needs to be complemented with specification of the non-functional behavior of services and definition of related ontologies. We plan to do this building on our previous work described in [6], which has identified key non-functional features of the mobile environment.

As discussed and demonstrated in this paper, ontologies enable a rich representation of services and a common understanding about their features. As discussed in the

OWL specification[10] and in [17], there are two advantages of ontologies over simple XML schemas. First, an ontology is a knowledge representation backed up by enhanced reasoning supported by the OWL axiomatic semantics. Second, OWL ontologies may benefit from the availability of generic tools that can reason about them. By contrast, if one built a system capable of reasoning about a specific industry-standard XML schema, this would inevitably be specific to the particular subject domain. Building a sound and useful reasoning system is not a simple effort, while constructing an ontology is much more manageable. The complex reasoning employed in the example of Section 4.1 to assert conformance between connector interaction models would not be easy to implement based simply on XML schemas.

OWL reasoning tools shall be employed by the introduced conformance relations and interoperability methods. A number of such tools already exist, such as the ones discussed in the previous section. Conformance verification needs to be integrated with the runtime system, i.e., the middleware, and be carried out online. Interoperability methods further involve processing and communication cost upon their functioning, but also upon their dynamic instantiation, as discussed in Section 4.1; they shall as well function with an acceptable runtime overhead. These requirements are even more challenging if we take into account the resource constraints of wireless devices. A number of techniques need to be combined in this context, including effective tools for checking conformance relations and lightweight interoperability mechanisms in the wireless environment, possibly exploiting the capabilities of resource-rich devices in the area so as to effectively distribute the load associated with the dynamic composition of mobile services. We are thus currently investigating base online tools and techniques to support open, dynamic system composition, while keeping the runtime overhead acceptable for wireless, resource-constrained devices.

In the spirit of the general principles identified for connector modeling and connector interoperability, we have already elaborated preliminary work towards middleware interoperability. Specifically, we have studied service discovery protocol interoperability in the open mobile environment [12]. This solution employs a mapping of several standard service discovery protocols (SLP [7], UPnP[11], Jini [8]) on semantic events. This approach includes dynamic instantiation of the appropriate connector customizer, as discussed in Section 4.1. This work is currently being extended to interoperability between standard middleware communication protocols (SOAP, RMI).

References

1. Curbera, F., Mukhi, N., Weerawarana, S. On the Emergence of a Web Services Component Model. In *Proceedings of the WCOP 2001 workshop at ECOOP 2001*, Budapest, Hungary, June 2001.
2. Tim Berners-Lee, James Hendler, Ora Lassila. The Semantic Web. *Scientific American*, May 2001

[10] http://www.w3.org/TR/owl-guide/
[11] http://www.upnp.org/

3. A. Tsounis, C. Anagnostopoulos, and S. Hadjiefthymiades. The Role of Semantic Web and Ontologies in Pervasive Computing Environments. In *Proceedings of Mobile and Ubiquitous Information Access Workshop*, *Mobile HCI '04*, Glasgow, UK, Sept. 2004.
4. R. Milner. *Communicating and Mobile Systems: The π-Calculus*. Cambridge University Press, 1999.
5. Valerie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Francoise Sailhan, Rafik Chibout, Nicole Levy, Angel Talamona. Developing Ambient Intelligence Systems: A Solution based on Web Services. In *Journal of Automated Software Engineering*, Vol. 12(1), January 2005.
6. V. Issarny, F. Tartanoglu, J. Liu, F. Sailhan. Software Architecture for Mobile Distributed Computing. In *Proceedings of 4th Working IEEE/IFIP Conference on Software Architecture* (WICSA'04). 12-15 June 2004. Oslo, Norway. To appear.
7. E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, version 2. IETF RFC 2608, June 1999.
8. Waldo, J. (1999). The Jini Architecture for Network-centric Computing. In *Communications of the ACM*, July 1999, pp. 76-82.
9. G.-C. Roman, G. Picco, and A. Murphy. Software Engineering for Mobility: A Roadmap. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE'22)*, 2000.
10. Paolucci, M. et al. Semantic Matching of Web Services Capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC 02)*, 2002.
11. Jinshan Liu, Valerie Issarny. QoS-aware Service Location in Mobile Ad Hoc Networks. In *Proceedings of the 5th IEEE International Conference on Mobile Data Management (MDM'2004)*, January 2004.
12. Yerom-David Bromberg, Valerie Issarny. Service Discovery Protocols Interoperability in the Mobile Environment. In *Proceedings of the International Workshop Software Engineering and Middleware (SEM).* September 2004.
13. E. Friedman-Hill. *Jess: Java Expert System Shell*.
14. J. Kopena and W. C. Regli. DAMLJessKB: A Tool for Reasoning with the Semantic Web. In *ISWC 2003*, 2003.
15. B. McBride. Jena: Implementing the RDF Model and Syntax Specification. In *Semantic Web Workshop*, *WWW 2001*, 2001.
16. S. McIlraith, D. Martin. Bringing Semantics to Web Services, *IEEE Intell. Syst.*, 18 (1) (2003), 90–93.
17. Declan O'Sullivan and David Lewis. Semantically Driven Service Interoperability for Pervasive Computing. In *Proceedings of the Third ACM International Workshop on Data Engineering for Wireless and Mobile Access*, *MobiDE 2003*, September 19, 2003, San Diego, California, USA.
18. B. Medjahed, A. Bouguettaya, and A. Elmagarmid. Composing Web Services on the Semantic Web. *The VLDB Journal*, 12(4):333-351, November 2003.
19. M. Paolucci and K. Sycara, Autonomous Semantic Web Services, *IEEE Internet Computing*, vol. 7, no. 5, September/October 2003.
20. D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, July 6-9, 2004, San Diego, California, USA.
21. N. Mehta, N. Medvidovic, and S. Phadke. Towards a taxonomy of software connectors. In *21st International Conference on Software Engineering*, November 1999.

22. Anupriya Ankolekar, Frank Huch and Katia Sycara. Concurrent Execution Semantics for DAML-S with Subtypes. In *Proceedings of The First International Semantic Web Conference (ISWC)*, 2002.
23. C. Bettstetter and C. Renner. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. In *Proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications*, 2000.
24. Natalya F. Noy and Deborah L. McGuinness. *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford University.
25. R. Allen and D. Garlan. A Formal Basis for Architectural Connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213-249, 1997.
26. Sycara, Katia; Paolucci, Massimo; Ankolekar, Anupriya; Srinivasan, Naveen. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, Volume 1, Issue 1, December 2003.
27. J. Magee and J. Kramer. Dynamic Structure in Software Architecture. In *Proceedings of the ACM SIGSOFT'96 Symposium on Foundations of Software Engineering*, pages 3-14, 1996.
28. Massimo Paolucci, Anupriya Ankolekar, Naveen Srinivasan and Katia Sycara. The DAML-S Virtual Machine. In *Proceedings of the Second International Semantic Web Conference (ISWC)*, 2003, Sandial Island, Fl, USA, October 2003, pp 290-305.
29. M. P. Papazoglou, D. Georgakopoulos (Eds.). Service-oriented computing. *Special section in Communications of the ACM,* Volume 46, Issue 10, October 2003.