Semantic and syntactic modeling of component-based services for context-aware pervasive systems using OWL-s

Davy Preuveneers and Yolande Berbers

Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium,
{davy.preuveneers, yolande.berbers}@cs.kuleuven.ac.be,
http://www.cs.kuleuven.ac.be

Abstract. In this paper, we present a service design methodology and specification as a basis for a pervasive context-aware service infrastructure. The service specification is based on the OWL-s specification, a standard proposed to add a semantic layer on top of WSDL web service descriptions. We have defined a set of OWL-s concepts that make it possible to express various pervasive service related properties, including service adaptation, relocation, personalization, deployment and runtime requirements. Though not straightforward to parse within strict resource boundaries, OWL-s provides an open, flexible specification language for expressing syntactic and semantic pervasive service characteristics and it increases service interoperability.

1 Introduction

The growing presence of mobile devices, such as laptops, PDAs and smartphones, along with advances in wireless network communication technologies, have created new opportunities for making the applications and services available on these hosting devices more intelligent and supportive to the user. This trend will lead us to a new computing future, often called *ubiquitous and pervasive computing*, where people are surrounded by and interacting with many small embedded networked devices. These devices and services will support users in a large variety of tasks, while allowing users to be mobile at the same time. Two important aspects to be considered within this ubiquitous and pervasive computing paradigm are (1) the highly dynamic environments in which services will operate and (2) the notion of user related information for supporting personalized assistance. While the first is the result of the high availability of hardware systems with different resource characteristics and the desire to be able to change the serving host of a running application without interrupting it, the second is the driving force behind services becoming more sensitive to user requirements and preferences while becoming less dependent on user attention. Both require applications and services to be more adaptive.

Although the design and implementation of reliable and robust applications is already a difficult task, it is clear that the development of adaptable pervasive

services will even further increase the complexity. The effort to cope successfully with this complexity requires a firm foundation, consisting of: (1) a sound general service design methodology to support discovery, adaptation, composition and distribution as important properties of services within ubiquitous computing environments, and (2) the adequate modeling of relevant information for characterizing the user-service interaction that drives the service personalization and adaptation.

The information being considered in the attempt to adapt and personalize a service is referred to as the *context* [1] of a user or device and its environment, and it often includes properties such as current location, time, user preferences and activities, available devices, services and resources in the neighborhood, etc. The use of this context information allows services to be adapted to a device's capabilities and to user preferences. However, this requires that the underlying context infrastructure be capable of acquiring and transforming context information and of reasoning on the basis of this information, thus showing the need for a uniform and interchangeable context representation. Additionaly, these so-called *context-aware* systems will need to support user mobility and enhanced service cooperation, including service discovery and composition. As discovery and composition require a different granularity of machine interpretable information, it is paramount to include both semantic and syntactic service information within the context specification as part of the information describing available services in the neighborhood. In this paper we show how OWL-s [2] can be used to provide a multi-functional service description.

In section 2 we describe our novel methodology for designing services to be deployed on context-aware mobile embedded devices. In section 3 we discuss how these services can be semantically and syntactically specified using OWL-s and we introduce new OWL-s concepts related to pervasive services for a formal concretization of our design methodology. Section 4 provides an overview of related work. We end with conclusions and future work in section 5.

2 Design methodology for pervasive services

In several computer science domains the concept of services refers to a computational entity that offers a particular functionality to a possibly networked environment. Typical examples of where this term is used are in the domains of web services, telematics, residential gateways and mobile services. Although web services target different users, all the services have in common the fact that they are deployed to offer users a certain functionality through the use of a well-defined interface, thereby providing a comfortable way for users to achieve their goals and perform their tasks.

However, the different services within the context of ubiquitous and pervasive computing have different characteristics and requirements compared to web services, for example, and they require an adequate design methodology to support these needs from the bottom up. These requirements will be presented in the following subsection. In subsection 2.2 we discuss how our component-based development approach is able to fulfill these requirements.

2.1 Functional requirements and characteristics of pervasive services

The following paragraphs give an overview of the requirements of pervasive services to be supported within our service design methodology.

R.1 User personalization
A user may have requirements or preferences regarding services he, or a device on his behalf, wants to execute. These requirements and preferences will result in constraints on which services are selected, how they are adapted and composed and how they behave at runtime. Ideally, vague preference descriptions in natural language would need to be transformed into formal machine-interpretable specifications.

R.2 Deployability on embedded systems
Service interaction will mainly occur between users and small, embedded systems. This means that the execution of services on these devices is subject to tight resource constraints, including limited memory, processing power and network bandwidth. Above all, it is very likely that these devices will have to share resources while serving several users concurrently, thus causing a highly dynamic fluctuation in currently available resources. Hence, it should be clear before deployment whether an available service is able to execute properly or whether optimizing adaptations are needed to better exploit the available resources.

R.3 User mobility
User mobility is the cornerstone of the future. However, while pervasive services are mobile and interact with fixed remote services, their network performance tends to vary unexpectedly and, in the worst case, the network connections can get lost. Also, a user may wish to run a service offline to increase the autonomy of his handheld device by disabling the highly power-consuming wireless network communication. For this reason, users should be able to select a remotely provided service and run it locally.

R.4 Service relocation
Related to the previous requirement, it is possible that users may want to make use of a service that cannot currently be deployed and run on their own device, either because the resource requirements are way beyond what the device is able to offer, or because the device is already running so many applications that the new application does not fit within the currently available resources. The user should then be able either to run the new service on a more powerful device in the neighborhood or to relocate (parts of) the already running services.

R.5 Adaptability
The offering of personalized services and the running of them on devices with different resource characteristics requires support for adapting the services. This means that services should not be monolithically designed, but rather that they should have a modular structure that makes it possible to replace parts of the service without changing the overall functionality of the service as requested by the user.

R.6 Flexibility
Flexibility is the result of having pervasive services with well-defined interfaces that support automatic service interaction and composition without looking into the inner workings of the service. A description of the interfaces and message formats for inter-service communication, however, is not enough in and of itself to decide whether a service will provide the user-requested functionality, since this description will not define the semantic meaning of a service. We will further elaborate on modeling the semantics of services in section 3.

2.2 A component-based design for pervasive services

In the following sections we will propose a component-based service design with support for service adaptation, interaction and composition within a mobile context. The proposed service design is based on the SEESCOA methodology [3], which means that it incorporates components and connectors as functional building blocks. However, the service specification includes not only functional aspects but also non-functional information, which makes service discovery and interaction possible. A general overview of the service specification is given in Figure 1.

Functional entities of a service
The following entities together form the business logic of a service and define its functional aspects:

- *Components:* Services are created through the assembling of components. As services need to be deployed on devices with varying characteristics, the use of components makes it possible to dynamically adapt services either by relocating components or by replacing them with others in order to optimize deployment on a specific device. Adaptable components support requirements *R.1*, *R.2* and *R.5*.
- *Connectors:* Connectors, which are linked to the component ports of a component, provide communication channels between components within a service.
- *Service Ports:* Service ports are the visible interfaces of a service to the external world. These service ports are mapped onto ports of internal components, and hence are communication gateways to other services or components, thus fulfilling requirement *R.6*. As other services and components can only communicate with a service through these ports, the service itself can be considered as a component. Hence a service imposes a hierarchy of components.
- *Service Control Interface:* The Service Control Interface is a standard dedicated interface for controlling a service. It allows the service to be (re)started, updated, relocated, stopped and uninstalled. By making this an obligatory interface, no knowledge about the other service ports is requiered for basic service management. The Service Control Interface is responsible for requirements *R.3* and *R.4*.

Non-functional entities of a service
Services have non-functional characteristics defining restrictions on how services

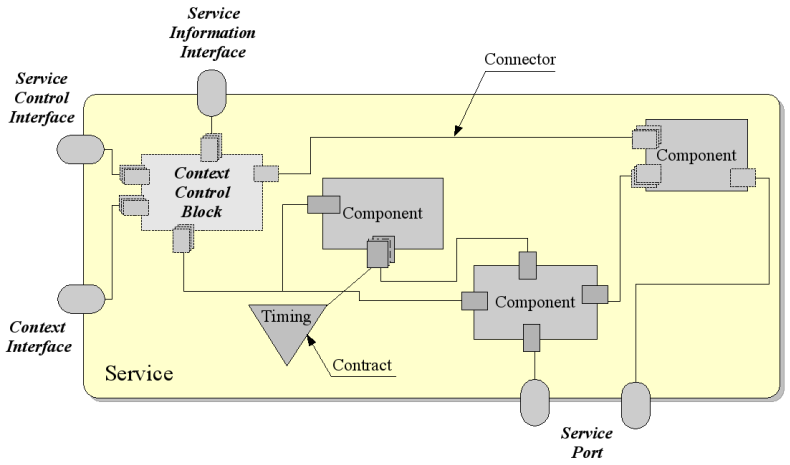

Fig. 1. Functional and non-functional entities within a component-based service

are discovered, adapted and composed. These characteristics also provide contextual information, both semantic and syntactic in nature, which is intended to be used by other services for better service cooperation.

- *Contracts:* Contracts [4] impose non-functional constraints on a component or a group of interacting components. Contracts can be used, for example,for guaranteeing memory or bandwidth constraints within a service internally. If, however, contracts involve component ports that are mapped onto the ports of a particular service, then they can be considered dependencies for connecting this service to other components or services. Contracts can be used to specify user requirements and to ensure optimal deployment on embedded devices, hence fulfilling requirements *R.1*, *R.2*, *R.4* and *R.5*.
- *Service Information Interface:* To fulfill requirement *R.6*, the Service Information Interface provides a static description of the semantics and syntax of a service and its service ports, and hence of how the service can be interfaced, so that other components or services can discover and use the service without any manual configuration or wiring. This information is expressed in OWL-s, as will be discussed in section 3.
- *Context Interface:* The Context Interface is responsible for the sending and receiving of the context information, which is available only at run-time when the service is active. Among other things, it allows the service to be notified of new resources, and to inform other services or devices about resources currently in use by this service. This context interface sustains requirement *R.5*.
- *Context Control Block:* The Context Control Block is not an interface to the outside world, but rather is responsible for the management and processing of context information. This block can be shared by several services being hosted on the same device and it acts as any normal component within

the service. Its inner working, which is beyond the scope of this paper, is described in [5].

Figure 1 shows that a service can be considered to be a component with a fixed set of predefined interfaces (*context interface, service control interface, service information interface*) and an optionally shared subcomponent for managing the service and making it context-aware (*context control block*).

The necessary infrastructure for this service model was developed on top of Draco [6], an extensible runtime system for components designed to be run on embedded devices. The base system is very small and lightweight with support for extensions such as component distribution, live updates, contract monitoring and resource management. This runtime environment with extensions provides a unique test platform for validating the proposed service concepts in an ambient intelligence context.

3 Semantic and syntactic service specification using OWL-s

Within the context of ubiquitous and pervasive computing, services should be able to work together automatically with other services in the neighborhood. To support cooperating mobile services, the services need to be able to discover and interact with each other. This implies the need for a uniform service specification containing both the functional and the non-functional aspects, as a service is more than just a collection of APIs. The non-functional part is responsible, among other things, for describing the semantics and syntax of a service. Syntactic descriptions of service interfaces are useful for the processing of service invocations, but are less relevant when high-level semantic information about a service is required during service discovery, when a service discovery protocol (SDP) such as UPnP, Jini, Salutation, SLP or Bluetooth SDP [7] is used.

Service ontologies provide a solution for this problem as they describe concepts and relationships between service concepts, and specify rules and constraints on certain service attributes. The OWL Services Coalition has recently described an ontology specifically for web services, namely OWL-s [2]. The idea behind this ontology is that web services alone offer poor support for the matchmaking in service discovery. The XML based Web Service Description Language (WSDL) [8], which is used to describe a web service interface, specifies the functionality and message formats of a service at a syntactic level and hides implementation details, thus increasing cross-platform interoperability. The interpretation of their meaning, however, is left to the user, a fact which reveals the lack of semantics within these service descriptions. The Semantic Web community, using the OWL-s ontology specification, addresses this problem by adding a semantic layer to achieve automatic discovery, composition, monitoring and execution of web services.

After giving a short overview of OWL-s, we will show how to use and extend OWL-s for our component-based service model to specify the typical pervasive service characteristics as described in section 2.

3.1 Overview of OWL-s

OWL-s is an OWL-based web service ontology used for expressing the semantic meaning of a web service, with a binding to WSDL for providing a syntactic description of the service. A service description in OWL-s has three parts, each of which is responsible for specific information relating to a service: a *Service Profile*, a *Service Model* and a *Service Grounding*.

The Service Profile is mainly used for automatic discovery. Apart from very general information on the organization that provides the service, such as contact information and a textual description, it provides a functional description of the service, specifying its inputs, outputs, preconditions and results, as well as an unbounded list of service parameters that can contain any type of information, such as for example a quality rating or classification.

The Service Model, on the other hand, describes the control flow and data flow within a service by specifying the service either as a simple process or as a composite process of several atomic processes combined by one or more control constructs specifying sequential or parallel execution, with or without *if-then-else* conditions or *repeat-until* loops, etc.

While the previous models provide an abstract description of a service, the Service Grounding provides a concrete description with details on how to access a service, specifying message and protocol formats, serialization, transport and addressing.

3.2 Extending OWL-s for pervasive services

As mentioned before, each service is described by a Service Profile, a Service Model and a Service Grounding specification. In this section we will define a set of OWL-s concepts to specify various requirements as described in section 2. The new concepts will be introduced briefly, after which we will illustrate their implementation by means of a short example.

Required Resources
Services on embedded systems are subject to tight resource constraints and are therefore not guaranteed to always run optimally on any given device, as already mentioned in requirement *R.2*. For this reason, minimal resource requirement specifications for a service to be deployed and executed are needed, such as requirements for minimum available memory, processing power, bandwidth, etc. Other requirements may include the presence of certain input or output facilities on the hosting device.

```
<profile:requiredResources>
  <resource:Resource rdf:ID="CommunicationService">
    <resource:memory>1 MiB</resource:memory>
    <resource:cpu>40 MIPS</resource:cpu>
    <resource:bandwidth>64 kbps</resource:bandwidth>
  </resource:Resource>
</profile:requiredResources>
<profile:requiredOutput rdf:resource="#Speaker"/>
```

Contracts
Contracts provide a way to define an agreement between two or more parties. For example, a memory contract between a service and the system ensures that the service will have the specified amount of memory at its disposal until the contract has been ended by the requester or after mutual agreement. Contracts can also be specified between internal components or component ports of a service. A service may provide several contracts in which it specifies and ensures different behaviour when other conditions are fulfilled, hence fulfilling requirements *R.2* and *R.6*.

Contracts can also be used to enforce user specific requirements or preferences as discussed in requirement *R.1*. While user requirements impose an obligation to the service, user preferences assume a best effort in trying to comply with user demands.

```
<process:hasContract>
  <expr:ContractCondition>
    <expr:expressionBody>
      $port:audio_out->bandwidth <= max_bandwidth
    </expr:expressionBody>
  </expr:ContractCondition>
</process:hasContract>
```

Runtime Adaptation
The flexibility of a service can be measured in terms of its ability to adapt itself to given circumstances. The easiest way to provide service adaptation in our service design methodology is to specify alternative components and optional components. Alternative components provide similar functionality, but require different runtime resources. Optional components, such as filters, provide added functionality to the service: they specify an internal wiring that will bypass their functions when they are disabled, just as if the component were not there. When a user or device has a choice between services with similar functionality, the most flexible one for adaptation will be selected.

```
<process:hasAlternative>Mpeg2Encoder</process:hasAlternative>
<process:hasAlternative>DivxEncoder</process:hasAlternative>
<process:hasOptional rdf:resource="#VideoRescaler">
  $port:video_in => $port:video_out
</process:hasOptional>
```

The new concepts, which were introduced by subclassing the Service Profile and Service Model, have resulted in a specialized service specification. The Service Grounding is a matching between the previous models and the component implementation. It is very similar to how it is done with the WSDL binding as a WSDL [8] service description also defines message types for communication. For example, a grocery list message is defined by elements declaring *product* as a string, and *quantity* as an integer. Instances of these message types are then later on defined as ingoing or outgoing messages of a service port, which location is addressed by its URL. This differs with our component specification as

we address a port by its name. The intercomponent communication itself is more straightforward and is not using a protocol such as SOAP or HTTP.

4 Related work

Some well-known service architectures include the .Net and the J2EE platforms. These architectures are more oriented to developing multi-tier enterprise applications, however, and less applicable than our service design methodology for services in a mobile computing environment. OSGi [9], on the other hand, is a service platform intended to provide both servers and embedded devices with the capability of managing the life cycle of the software components in the device without having to disrupt the device's operation. The difference between OSGi and our service design is that OSGi has no semantic or syntactic service descriptions that specify how a service can be adapted internally to meet user demands or resource requirements.

The Semantic Web has provided us with specification languages such as DAML+OIL [10] and OWL [11] for describing ontologies as the ideal candidate for knowledge representation and processing. DAML-s [12] was one of the first ontologies to add semantic meaning to web services, though it was later replaced by OWL-s as a more mature semantic service specification.

METEOR-S [13] is another proposal for enhancing web service descriptions and composition. METEOR-S focuses more on the communication aspects between services. The fact that semantics using DAML+OIL ontologies have been added to WSDL [8] and UDDI [14], makes this specification highly linked to the concept of web services and less suited for our component-based pervasive service methodology.

Similar recent proposals for semantic web service infrastructure support include the Web Services Modeling Framework [15], the Web Service Modeling Ontology [16] and the Web Services Modeling Execution [17] specifications.

5 Conclusions and future work

In this paper we have presented a novel service design methodology supporting discovery, adaptation, relocation, composition and deployment on resource limited devices from the bottom up. This methodology is an ideal candidate for developing applications and services within ubiquitous and pervasive computing environments.

As the success of this methodology largely depends on the specification of services, we have investigated how OWL-s can be used and extended to concretize this design methodology. The main advantage of using OWL-s is, in the first place, that it is an open and extendible specification for describing semantics and syntax, which accordingly supports better service interoperability between devices.

The focus of our future work will be on further enhancements to the processing of context information. Other research to be carried out includes determining

how general user requirements or preferences can be mapped to complex OWL-s rules and constraints.

References

1. Preuveneers, D., Van den Bergh, J., Wagelaar, D., Georges, A., Rigole, P., Clerckx, T., Berbers, Y., Coninx, K., Jonckers, V., De Bosschere, K.: Towards an extensible context ontology for Ambient Intelligence. In: Proceedings of the Second European Symposium on Ambient Intelligence, Springer (2004)
2. The OWL Services Coalition: OWL-S: Semantic Markup for Web Services, Release 1.1. http://www.daml.org/services/owl-s/1.1/index.html (2004)
3. Urting, D., Van Baelen, S., Holvoet, T., Berbers, Y.: Embedded Software Development: Components and Contracts. In: Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems. (2001) 685–690
4. Wils, A., Gorinsek, J., Van Baelen, S., Berbers, Y., De Vlaminck, K.: Flexible Component Contracts for Local Resource Awareness. http://www.cs.kuleuven.ac.be/ andrew/stuff/ecoop2003.pdf (2003)
5. Preuveneers, D., Berbers, Y.: A Component-based Approach for Managing Context Information. Technical Report CW397, Department of Computer Science, Katholieke Universiteit Leuven, Belgium (2004)
6. Vandewoude, Y.: Draco : An adaptive runtime environment for components . (http://www.cs.kuleuven.ac.be/~yvesv/Draco/index.html)
7. Preuveneers, D., Berbers, Y.: Suitability of existing service discovery protocols for mobile users in an ambient intelligence environment. In: Proceedings of the International Conference on Pervasive Computing and Communications, CSREA Press (2004) 760–764
8. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl (2001)
9. Open Services Gateway Initiative: OSGi Service Gateway Specification, Release 3.0 (2003)
10. Horrocks, I., van Harmelen, F., Patel-Schneider, P.: DAML+OIL, Darpa Agent Markup Language and Ontology Interference Layer (2001)
11. Smith, M.K., Welty, C., McGuinness, D.L.: OWL Web Ontology Language Guide (2003)
12. The DAML Services Coalition: DAML-S: Semantic Markup for Web Services. http://www.daml.org/services/daml-s/0.9/daml-s.html (2003)
13. Sivashanmugam, K., Verma, K., Sheth, A., Miller, J.: Adding Semantics to Web Services Standards. In: Proceedings of the 1st International Conference on Web Services (ICWS'03). (2003) 395–401
14. OASIS: The Universal Description, Discovery and Integration (UDDI) (2000)
15. Bussler, C., Fensel, D., Maedche, A.: Web Service Modeling Framework WSMF (2002)
16. The SDK WSMO working group: The Web Service Modeling Ontology. (http://www.wsmo.org/2004/d2/v1.0/)
17. Zaremba, M., Haller, A., Zaremba, M., Moran, M.: WSMX - Infrastructure for execution of semantic web services. In: Proceedings of the 2nd International Conference on Web Services (ICWS'04), Springer (2004)