Using OWL-S to annotate services with ancillary behaviour

Roxana Belecheanu, Mariusz Jacyno, Terry Payne

School of Electronics and Computer Science
University of Southampton
Highfield, Southampton, SO17 1BJ
{rab2, mj04r, trp}@ecs.soton.ac.uk

Abstract: This paper introduces the concept of services with ancillary behaviour and illustrates the use of OWL-S to semantically describe them. The OWL-S syntax used reflects the dynamic and core-function independent nature of ancillary behaviour. The approach is illustrated on the case of a ubiquitous computing system designed to offer care in the home of a cardiac patient. Here one of the challenges is to ensure service availability, team awareness and transaction atomicity. The concept of commitment is discussed as an example of ancillary behaviour that can achieve these requirements.

1 Introduction

Service oriented applications often require that services which implement certain core functions are accompanied by supporting functionality, like monitoring behaviour, commitment, authentication, encryption/decryption. This supporting functionality is usually relevant only to the context of the core function, but does not always play a direct role in the invocation of the core function. This paper introduces the concept of *ancillary behaviour* to describe functionality that is additional to the core service typically annotated in an OWL-S description, and which has the role of augmenting the service's capability, or enhancing the QoS achieved by the core service.

The ancillary functionality of a web service can be either mandatory or optional. Mandatory ancillary operations (e.g. authentication) *must* be executed in addition to the service's core functionality and the service cannot be invoked without the execution of these operations. Ancillary functionality contributes to the delivery of a service either by providing a *supporting role* (enable the core service) or an *enhancing role* (increase the value of the core service) [1].

A related but distinct concept that can also be introduced here is that of a *higher-order service*. A higher-order service is one which determines a particular aspect of the invocation of a core service without invoking the core service. For example, in order to find out the cost of the invocation of a service for certain input parameters, a costing service can be attached to the core service, whereby the invocation of the cost function would not require the invocation of the core service.

2 An application example

An application example for the concept of services with ancillary behaviour is an ubiquitous computing [4] system whereby a set of heterogeneous devices and their applications establish connections between each other in a dynamic manner (i.e. devices leaving and joining the system unexpectedly), in order to achieve certain tasks. Due to the high variability of the Quality of Service of wireless data communications (e.g. line rate, throughput, error rate) in such a system, there is also a need for dynamically discovering and composing the applications running on these connected devices. Abstracting devices (physical) functionality as services (logical) functionality and using OWL-S to semantically annotate these services can help achieve this flexibility. Thus building interoperable service descriptions similar to the Semantic Web Services is needed to accomplish tasks like service discovery, management, invocation and monitoring.

For example, in the case of a ubiquitous system designed to offer care in the home of a patient, there may be the issue of having a limited number of devices, each providing services that take a finite time to execute, and that may have to be shared by several contexts. Hence, no guarantees exist that any single service is available for invocation at any given time. Therefore, a service centric application must ensure:

1. *service availability*: services which are critical for the patient care (e.g. heart rate monitoring) must be available for execution at the desired time.
2. *team awareness*: services must be aware of the status of other services they depend on.
3. *atomic transactions*: due to high power consumption of message transmission, it is necessary to reduce the number of service requests, or to support a mechanism whereby both core and supporting activities can be requested via a single call (i.e. as one transaction).

3 Commitment based services

Commitment is an example of ancillary behaviour that can help achieve this type of service interactions. The concept is used to enforce that the provider of a commitment supporting service commits to perform an action for a requester. This has been formalised within the theory of local and social agent behaviour [2], using concepts from the Beliefs-Desires-Intentions model. Commitment has also been used to model coordination protocols for business transactions [3], e.g. in an “atomic transaction”, several services committed to one requester will succeed or fail as an atomic unit. A common point of the two interpretations of commitment is that a protocol (e.g. operations like *RequestCommitment*, *DischargeCommitment*, *ReleaseCommitment*, etc.) can be designed to allow services to check each other’s availability to perform a joint task, thus building *team awareness*. The difference between the two commitment models is that a committed service in agent theory will eventually perform the task, but may accept other requests in the meantime, while a committed service in a transactions model must be available to the requester immediately after commitment is granted (a reservation-like interaction).

4 Describing ancillary service behaviour in OWL-S

Service ancillary behaviour has two salient characteristics:

1. It is *dynamic*, i.e. it involves communication with other services of which instances can only be discovered at runtime; (hence it cannot be included in the static description of the service's process).
2. It is *core function independent*, i.e. it is common to a range of services with different core functions.

For these reasons, ancillary behaviour has to be abstracted and described separately from a given service's core functionality, thus facilitating several different services sharing the same ancillary behaviour. Ancillary behaviour description should be loosely linked to the core service specification. The resulting workflow for both the core and ancillary functionality can then be realized (through entailment) when an agent reasons about the service, with respect to a usage context.

To illustrate how core services can be augmented by ancillary services, we take the example of commitment as ancillary behaviour and an example of a core service from the healthcare setting. Suppose *HRMonitor_with_Commitment* is a heart rate monitoring service which supports commitment. To annotate it in OWL-S, we use the core service instance *GetHeartRate* and augment it by: i) defining a *CommitmentService* instance and ii) by linking the core service description with the ancillary service description. *CommitmentService* is the representation of a commitment protocol that ensures that the interaction with the *HRMonitor_with_Commitment* service occurs as one transaction. The resulting workflow for this service must therefore be the sequence of the processes: *RequestCommitment*, *GetHeartRate* and *DischargeCommitment*. Suppose that *GetHeartRate* is an atomic process:

```
<process:AtomicProcess rdf:ID="GetHeartRate">
 <hasOutput rdf:resource="ansConcepts#HeartRate"/>
</process:AtomicProcess>
```

To combine the core and ancillary workflows, we use the OWL-S *Simple Process* class as an unbound service abstraction that can be dynamically linked to the *GetHeartRate* service instance. The *CommitmentService* workflow can be written as:

```
<process:CompositeProcess rdf:ID="Commitment_Process">
 <process:composedOf>
  <process:Sequence>
   <process:Components>
    <process:AtomicProcess rdf:resource="#RequestCommitment"/>
    <process:SimpleProcess rdf:resource="#Core_Function"/>
    <process:AtomicProcess rdf:resource="#DischargeCommitment"/>
   </process:Components>
  </process:Sequence>
 </process:composedOf>
</process:CompositeProcess>,
```

whereby the Simple Process represents a step that must be replaced by a core process before execution, thus allowing the annotation of the commitment as a function that is common and can be attached to several core services. The *CommitmentService* thus acts as a wrapper around the core function of any service that supports commitment.

Additional annotations are then added to the core *GetHeartRate* service to link it with the ancillary commitment service definition: 1. the *GetHeartRate* process

realizes the abstract Simple Process, thus stating that the ancillary service definition is effectively unbound, until reasoned about in context with the *GetHeartRate* service:

```
<process:AtomicProcess rdf:ID="GetHeartRate">
 <process:realizes rdf:resource="ansAncillary.owl#Core_Function"/>
</process:AtomicProcess>
```

2. In order to distinguish between a committed and a non-committed service during discovery, (as typically a service is requested according to the functional parameters of the core service), *ANSAncillaryFunctionality* is defined as a *serviceParameter* type of property, in the profile of any ANS service:

```
<ANSServiceProfile rdf:ID="HRMonitor_with_Commitment_Profile">
 <profile:has_process rdf:resource="#GetHeartRate"/>
 <ansProfile:ANSAncillaryFunctionality>
  <ansProfile:ANSAncillary>
   <profile:sParameter rdf:resource="ansAncillary#Commitment_Profile"/>
  </ansProfile:ANSAncillary>
 </ansProfile:ANSAncillaryFunctionality>
</ANSServiceProfile>
```

5 Further work

Further work aims to address issues which result from using this type of annotation: first, discovering services with ancillary behaviour requires searching by core functionality parameters, as well as by non-functional properties (i.e. *sParameter*), to determine what ancillary services are also provided as part of the service description. Secondly, the runtime composition of the workflows describing the core and ancillary sub-processes into one executable workflow is necessary. The possibility to annotate services with more than one type of ancillary behaviour must also be addressed.

Acknowledgements

This research is funded by DTI (UK) as part of the ANS (Autonomous Networked System) project; other research partners are Imperial College and Lancaster Univ.

References

1. Baida, Z., Akkermans, H. and Bernaras, A., 2003., The configurable nature of real-world services: analysis and demonstration, ICEC-Workshop.
2. P. R. Cohen and H. J. Levesque, Intention is choice with commitment, Artificial Intelligence, 42 (1990), pp. 213-261
3. Papazoglou, M.P., 2003. “Web Services and Business Transactions”, WWW: Internet and Web Information Systems, 6, pp. 49-91.
4. Weiser, M, "The Computer for the Twenty-First Century," Scientific American, pp. 94-10, September 1991