OWL-S for Describing Artifacts

Rossella Rubino[a] Ambra Molesini[b] Enrico Denti[b]

[a] *CIRSFID - Alma Mater Studiorum - Università di Bologna, Via Galliera 3, 40121 Bologna, Italy*
rossella.rubino@unibo.it
[b] *DEIS - Alma Mater Studiorum - Università di Bologna, Viale Risorgimento 2, 40136 Bologna, Italy*
ambra.molesini@unibo.it, enrico.denti@unibo.it

Abstract

Artifacts for Multi-Agent Systems have been defined as runtime entities providing some kind of function or service that agents can fruitfully exploit to achieve their individual or social goals. In order to enable agents to autonomously discover the available services, select the proper artifact(s), and use the services they need, it is necessary to describe the services offered by artifacts in terms of *(i)* what each service does, *(ii)* how to access it, and *(iii)* how it works. In this paper we explore how OWL-S (Ontology Web Language for Services), a language specifically aimed for the description of Web Services, can be exploited also for describing the services offered by artifacts. In particular, we investigate how to represent the artifact features by means of OWL-S, provide an example, and discuss some preliminary results in terms of advantages and limitations of this approach.

1 Introduction

According to Activity Theory [24], most of the human activities are mediated by some kind of artifact – physical, such as blackboards and traffic lights, or cognitive, such as languages and norms. Analogously, in the context of Multi-Agent Systems (MASs), artifacts are both conceptual and runtime entities that mediate agent activities [27], providing some kind of function or service that agents can fruitfully exploit to achieve their individual or social objectives.

In particular, in the Agents&Artifacts (A&A) [21, 17] meta-model, recently proposed for MAS engineering, artifacts – along with agents – are adopted as the basic building blocks to engineer complex software systems. More precisely, agents are the basic abstractions to represent active, task-/goal-oriented components, designed to pro-actively carry on one or more activities towards the achievement of an objective, requiring different levels of skill and reasoning capabilities. On the other hand, artifacts are the basic abstractions to represent passive, function-oriented building blocks, which are constructed and used by agents, either individually or cooperatively, during their working activities. So agents can be used to model individual activities, while artifacts can be well suited for mediating the interaction between individual components and their environment (including the other components), and for embodying the portion of the environment that is explicitly designed to support agents activities [17].

Many sorts of artifacts can populate a MAS, providing agents with a number of different services, embodying a variety of diverse models, technologies and tools, and addressing a wide range of application issues. In particular, artifacts are used to mediate between individual agents and the MAS (individual artifacts), to build up agent societies (social artifacts), and to mediate between a MAS and an external resource (resource artifacts) [20].

In this paper we explore the use of the OWL-S [3] language for describing the services offered by artifacts, so that agents can autonomously discover, select and use artifacts offering particular services into which they are interested. OWL-S is a OWL-based Web Service ontology, which

supplies Web Service providers with a core set of markup language constructs for describing the properties and capabilities of their Web Services in unambiguous, computer-interpretable form. OWL-S markup of Web services will facilitate the automation of Web Service tasks, including automated Web Service discovery, execution, composition and inter-operation. OWL-S seems a good candidate for describing services provided by artifacts because it should allow to express in a high-level way these services so as to support agents on the one hand in choosing the proper artifact, on the other, agents should be able to understand how the artifact works and know how to access it. As a consequence the use of OWL-S for describing artifacts' services could improve the openness of MASs and the mobility of agents: agents could access to artifacts in a standard way in each place of the net.

So, the paper is structured as follows. Section 2 introduces the artifact features, while Section 3.1 briefly presents the fundamental semantic web languages, with special regard to OWL-S. Next, Section 3.2 introduces how OWL-S can be used for describing artifacts, and Section 3.3 provides some details about artifact representation in OWL-S; a discussion about the pros and cons of this approach is developed in Section 3.4. Then Section 4 presents a simple example – a sketch of a flight tracking system – based on the (social) artifacts provided by the TuCSoN infrastructure technology. Some relevant related work is reported in Section 5; conclusions follow in Section 6.

2 Artifacts features

In its most general acceptation, an artifact conceptually exposes [20]:

Usage Interface — The set of operations provided by an artifact defines what is called its usage interface, which (intentionally) resembles interfaces of services, components or objects – in the object-oriented acceptation of the term. Operations are the means by which an artifact provides for a service or function: an agent executes an action over an artifact by invoking an artifact operation. Execution typically terminates with an operation completion, representing the outcome of the invocation.

Operating Instructions — Coupled with a usage interface, an artifact provides agents with operating instructions, that is a description of the procedure that an agent has to follow to meaningfully interact with an artifact over the time. Moreover, operating instructions also come with the specification of preconditions for actions and effects to perceptions [28].

Function Description — Finally, an artifact is characterised by a description of the functionality it provides. Function description explains what to obtain from the artifact – a key information for artifact selection. Clearly, function description is an abstraction over the actual implementation of the artifact: it hides inessential implementation details while highlighting its key aspects.

For instance, consider the case of a digital camera. In order to choose a digital camera among all that are available in the market, interested customers usually start by checking the cameras' technical specifications, such as its memory, zoom capabilities, etc.: these are actually a form of function description – they explain what the camera does. Also, each camera has its own buttons and panels, which must be operated according to the instruction provided by the vendor in the user's manual. Buttons and panels represent the camera's usage interface – i.e., how to access it – while the user's manual describes how to use them to suitably configure the camera resolution, the zoom ratio, etc. – thus representing the operating instruction.

In addition, artifacts typically exhibit further relevant properties, which enhance MAS engineers' but also agents' ability to use them for their own purposes. For instance, it should be possible to *monitor* artifacts as an observable part of the environment, so as to check the development of the activities, track the system history, and evaluate the overall system performance. Desirable artifact features can then be listed as follows:

Inspectability — The state of an artifact, its content, the laws governing its behaviour, its usage interface, operating instructions and function description might be all or partially inspectable by agents.

Controllability — The operational behaviour of an artifact should be controllable so as to allow engineers and agents to monitor its proper functioning: it should be possible to stop and restart an artifact working cycle, to trace its inner activity, and to observe and control a step-by-step execution.

Malleability — The behaviour of an artifact should be modifiable at execution time in order to adapt to the changing needs or mutable external conditions of a MAS.

Linkability — Artifacts can be used encapsulate and model reusable services in a MAS. To scale up with complexity of an environment, it might be useful to compose artifacts, by allowing artifacts to invoke operations on other artifacts.

As a final remark, it is worth noting that the above artifact features usually play different roles in the viewpoints of agents and of MAS engineers. For instance, operating instructions are mostly seen as a design tool by/for engineers, and as a run-time support by/for rational agents. Instead, features like inspectability and malleability gain particular interest when the two viewpoints can be made one: for instance, an intelligent agent capable of playing the role of the MAS engineer could in principle understand the state and dynamics of the MAS by observing the artifacts, and then possibly change the overall MAS behaviour by suitably altering the artifacts behaviour [20].

3 OWL-S and Artifacts

3.1 From Semantic Web Languages to OWL-S

The Semantic Web project [8] is aimed at defining a common framework for information exchange by giving semantics to the content of the documents shared and reused across applications. For this purpose, several ingredients are put together: customisable Extensible Markup Language (XML) [6] for describing data, descriptive technologies such as the Resource Description Framework (RDF) [14] and the Resource Description Framework Schema (RDFS) [7], and ontology languages such as the Ontology Web Language (OWL) [16] and Ontology Web Language for Services (OWL-S) [3].

RDF has been conceived for describing Web resources – that is, anything identified by a Uniform Resource Identifier (URI): in turn, RDFS specifies the schema for defining properties, classes and inter-relationships among classes. Their applications include the description of any kind of web content for multiple purposes, such as content rating, labelling for search engines, site maps, collaborative services, and e-commerce applications (e.g. to express price, availability, etc. of shopping items). Yet, the lack of some particular features (such as the local scope of properties, the disjointedness of classes, and other special characteristics) led to the development of a new language built on top of RDF and RDFS, specifically designed for processing Web information: *OWL* (Ontology Web Language, [16]). More precisely, three versions of OWL are available which provide different trade-offs between expressive power and efficient reasoning: *OWL Lite*, *OWL DL* (Description Logic) and *OWL Full*. OWL Lite supports users who just need a classification hierarchy and simple constraint features, while OWL DL is for users demanding the maximum expressiveness without losing computational completeness and decidability of reasoning systems; OWL Full is meant for users who need the maximum expressiveness and syntactic freedom of RDF, though with no computational guarantees.

However, just describing resources is not enough in the perspective of discovering, invoking, composing and monitoring web resources which offer services: this is why a specialisation of OWL for service description, called *OWL-S* (Ontology Web Language for Services), has also been defined. Figure 1 shows the (top-level) ontology of OWL-S in terms of an *RDF graph* – a kind of graph used to represent the relationships among resources, property and property values, where nodes (ovals) represent resources or property values, and arcs represent properties. According to that ontology, the discovery, selection and exploitation of a service calls for a suitable description of (i) what the service does, reported by the *Service Profile*; (ii) how it works, explained in the *Service Model*; and (iii) how to access it, detailed in the *Service Grounding*.

Service Profile — The service profile includes a description of what can be accomplished by the service, its limitations in terms of applicability and quality of service, and what is required from the service requester in order to use the service [3]. In addition, the service profile can contain information about the organisation or entity which provides the service and the service category, possibly referring to some standard classification system, such as United Nations Standard Products and Services Code (UNSPSC).

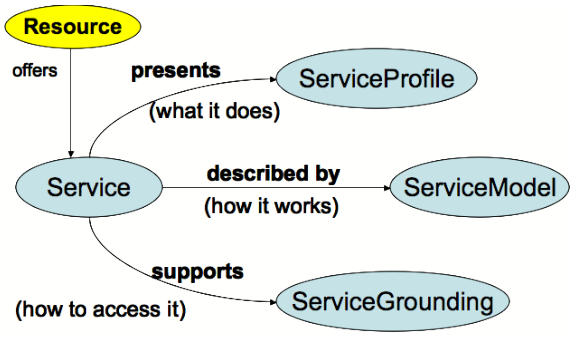

Figure 1: Top level of the service ontology represented as an RDF Graph

Service Model — After the service is found based on the service profile, the user needs to know how to ask for the service and what happens when the service is executed. To this end, the service model describes the semantic content of requests, the conditions under which particular outcomes may occur and, optionally, the step-by step processes leading to such outcomes.

OWL-S defines a subclass of the service model, namely the *process model*, which provides further details about the number of inputs, outputs, preconditions and effects. Three types of processes can be defined: *atomic*, *simple*, and *composite* processes [3]. Atomic services feature no ongoing interaction between user and service – just think of a service that returns the postal code of a city, or the longitude and latitude of a given place. Simple processes are a sort of 'intermediate abstractions', which can be used in two ways – as a view of atomic processes, and as a simplified representation of composite processes. Finally, composite services are composed of multiple services, and may require extended interaction or conversation between the requester and the set of services that are being used – think, for instance, of the online sell of a book.

Service Grounding — Service grounding basically specifies how to access a service, linking Web Services' semantic and syntactic description levels: in particular, its main task is to transform the semantic data to XML messages to be sent over the network, and conversely to define how the received XML messages should be interpreted as semantic data [15].[1]

So, it typically defines the communication protocol, the message formats and the serialisation techniques adopted for each semantic type of input or output specified in the service model.

Since the current definition of OWL-S does not account for any means to express grounding information, the WSDL [10] language has been selected as the initial grounding mechanism for describing the service interface. However, it should be noted that OWL-S does not ascribe inputs, outputs, preconditions and effects directly to WSDL operations; instead, such conditions are attached to atomic processes, which are then bound to WSDL operations by the OWL-S grounding service [15, 1].

In the next subsection we will discuss how OWL-S can be brought to the artifact world, i.e. exploited to discover and invoke not only Web Services, but also the services provided by artifacts.

3.2 Bringing OWL-S to the Artifact World

In order to enable agents to identify and use the services provided by artifacts – which is the key for building open MASs where intelligent agents dynamically look for artifacts and select the most adequate to their goals –, the artifacts' features need to be expressed in some high-level language.

Such a high-level language should also make it possible for agent to adopt a uniform cognitive level for interaction.

[1] Web Services are usually described at two abstraction levels: one defining atomic operations in terms of input / output messages, the other mapping operations and their associated messages to physical endpoints (ports and bindings). Ports declare the operations available with the corresponding inputs / outputs, while bindings declare the transport mechanism (usually SOAP [12]) used by each operation [9].

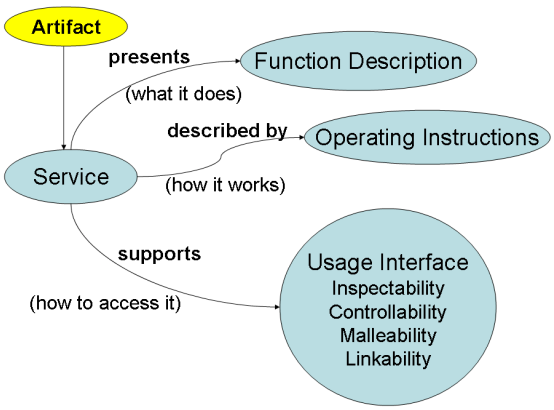

Figure 2: Artifact service ontology represented as an RDF Graph

Usually, agents speak with each other via languages like FIPA-ACL [11], while exploiting the environmental resources as a means for lower-level interaction. So, the agents' *interaction space* spans over different cognitive levels, because agents cannot interact with the other components (agents and resources) of the MAS in a uniform way. Artifacts, instead, make it possible to wrap the resources of a MAS and bring them to the cognitive level of agents, so that both the interaction among agents (on the one side) and between agents and artifacts (on the other) can occur at the same cognitive level, exploiting the high-level language that describes artifacts' services as the common language. In order to make this possible, the selected high-level language should be standard enough to support both the MAS openness and agent mobility, allowing heterogeneous agents to join a MAS, discover and use the services provided by the artifacts. By the way, this choice also simplifies the design of the MAS interaction space, since engineers no longer need to design ad-hoc protocols for each resource in the environment (the relationship between each resource and the artifact that wraps it concerns the internal design of the artifact and does not affect the interaction space of the MAS).

Among the possible choices, OWL-S seems a good candidate for several reasons. First, most of today's services are available as Web Services, and OWL-S is specially designed for that purpose. Moreover, Web Services can be "implemented" by agents and artifacts as in simpA-WS [23], using agents to model service users and providers, and artifacts as high-level mediating entities: so, artifacts operate as interfaces, encapsulating the technology that enables interaction via standard Web service protocols. As a further aspect, in principle OWL-S makes it possible to express the artifact features – function description, usage interface, and operating instruction – in a natural way via the service profile, the service model and the service grounding, respectively.

3.3 Artifact Representation

The mapping between the artifact features and OWL-S could be represented through an RDF Graph as depicted in Figure 2.

The service profile can be used to represent what the service offered by artifacts does, i.e. the function description; a fragment of the OWL-S file relating to the service profile is shown in Figure 3. The profile class represents the general profile of a service whose text description is contained in the property called `textDescription`. The profile description can be accompanied by a list of properties called `serviceParameters`: the actual parameter name, represented by `serviceParameterName`, can be just a literal, or the URI of the process parameter (a property). Of course, only one text description per profile is admitted, as stated by the value `1` in the ⟨`owl : cardinality`⟩ tag.

As stated in the previous section, OWL-S introduces the process model as its own special subclass of service model: so, the service model is expressed here as a process model. Correspondingly, Figure 4 defines inputs, outputs, preconditions and results for each process model. The `Parameter` class is the union of the `Input` and the `Output` classes; in the same way, results are represented in as a class, too. `hasPrecondition`, instead, is a property of `Process`.

```
<owl:Class rdf:about="#Profile">
 <rdfs:comment>
    </rdfs:comment>
 <rdfs:subClassOf>
  <owl:Restriction>
 <owl:onProperty rdf:resource="#textDescription"/>
    <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">
      1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:DatatypeProperty rdf:ID="textDescription">
 <rdfs:domain rdf:resource="#Profile"/>
 <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:ID="serviceParameterName">
 <rdfs:domain rdf:resource="#ServiceParameter"/>
 <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:Class rdf:about="#ServiceParameter">
 <rdfs:comment>
 A ServiceParameter should have at most 1 name (more precisely only
  one serviceParameterName)
 </rdfs:comment>
 <rdfs:subClassOf>
  <owl:Restriction>
 <owl:onProperty rdf:resource="#serviceParameterName"/>
    <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 3: Some fragments from the OWL document that describe the service profile

```
<owl:Class rdf:ID="Process">
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#parameterType" />
   <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">
   1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
 <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Parameter"/>
  <owl:Class rdf:about="#Result"/>
 </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="Parameter">
 <rdfs:comment>
    Parameter is the disjoint union of Input and Output
 </rdfs:comment>
 <owl:unionOf rdf:parseType="Collection">
  <owl:Class rdf:about="#Input"/>
  <owl:Class rdf:about="#Output"/>
 </owl:unionOf>
</owl:Class>
```

```
<owl:Class rdf:ID="Input">
 <rdfs:subClassOf rdf:resource="#Parameter"/>
 <owl:disjointWith rdf:resource="#Output"/>
</owl:Class>

<owl:Class rdf:ID="Output">
 <rdfs:subClassOf rdf:resource="#Parameter"/>
</owl:Class>

<owl:Class rdf:ID="Result">
 <rdfs:label>Result</rdfs:label>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasPrecondition">
 <rdfs:domain rdf:resource="#Process"/>
 <rdfs:range rdf:resource="&expr;#Condition"/>
</owl:ObjectProperty>
```

Figure 4: Some fragments from the OWL document that describe the service model

Finally, the usage interface that describes how to access the artifact is represented through the service grounding, which takes the form of a collection of `AtomicProcessGrounding` instances (see Figure 5) – one for each atomic process in the process model: this class relates elements of an OWL-S atomic process to a WSDL specification (or any other specification). Each instance of `AtomicProcessGrounding` must have exactly one value for `owlsProcess`: the other properties depend on the specifics of the grounding type.

```
<owl:Class rdf:ID="Grounding">
 <rdfs:comment>
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="&service;#ServiceGrounding"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasAtomicProcessGrounding">
 <rdfs:domain rdf:resource="#Grounding"/>
 <rdfs:range rdf:resource="#AtomicProcessGrounding"/>
</owl:ObjectProperty>
```

```
<owl:Class rdf:ID="AtomicProcessGrounding">
 <rdfs:comment>
 </rdfs:comment>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#owlsProcess"/>
   <owl:cardinality rdf:datatype="&xsd;#nonNegativeInteger">
   1</owl:cardinality>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="owlsProcess">
 <rdfs:comment>
  The atomic process to which this grounding applies.
 </rdfs:comment>
 <rdfs:domain rdf:resource="#AtomicProcessGrounding"/>
 <rdfs:range rdf:resource="&process;#AtomicProcess"/>
</owl:ObjectProperty> <owl:FunctionalProperty
rdf:about="#owlsProcess"/>
```

Figure 5: Some fragments from the OWL document that describe the service grounding

The `MessageMap` class maps the OWL-S parameters onto the parameters of the grounded operation. In particular, `InputMessageMap` maps inputs to grounding specification (see Figure 6), while `OutputMessageMap` maps outputs to grounding specification (see Figure 7); the `owlsParamater` property specifies the OWL-S parameter. According to the model shown in Figure 2, the usage in-

```
<owl:Class rdf:ID="MessageMap">
 <rdfs:comment>
 </rdfs:comment>
</owl:Class>

<owl:ObjectProperty rdf:ID="owlsParameter">
 <rdfs:comment>
  An input or output property of an atomic process.
 </rdfs:comment>
 <rdfs:domain rdf:resource="#MessageMap"/>
 <rdfs:range rdf:resource="&process;#Parameter"/>
</owl:ObjectProperty>
```

```
<owl:Class rdf:ID="InputMessageMap">
 <rdfs:subClassOf rdf:resource="#MessageMap"/>
 <rdfs:comment>
 </rdfs:comment>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#owlsParameter"/>
   <owl:allValuesFrom rdf:resource="&process;#Input"/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 6: Some fragments from the OWL document that describe the MessageMap and InputMessageMap

terface also includes other artifact properties such as inspectability, controllability, malleability and linkability. These properties enable agents to access artifact after selecting the service. Therefore, their OWL-S representation depends on the language chosen for grounding.

3.4 Discussion

Let us report here some preliminary considerations about the use OWL-S for describing artifacts.

```
<owl:Class rdf:ID="OutputMessageMap">
 <rdfs:comment>
 </rdfs:comment>
 <rdfs:subClassOf rdf:resource="#MessageMap"/>
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#owlsParameter"/>
   <owl:allValuesFrom rdf:resource="&process;#Output"/>
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Figure 7: Some fragments from the OWL document that describe the OutputMessageMap

As a first aspect, using OWL-S to describe what the service provided by artifact does and how it works in terms of preconditions, inputs, outputs and results is relatively simple: indeed, all we have to do is to make the information that agents are usually supposed to know semantically readable. However, the OWL-S description of how the service is actually carried out is not straightforward, mainly because of the OWL-S' inability to describe the information related to the physical functioning of the service: this forces to include another language – WSDL being the most natural choice in the Web service context – to add that kind of expressiveness. Yet, WSDL is definitely not the most adequate choice for describing the services offered by artifacts, since it requires information that simply do not fit the artifact case – e.g. the data and message types to be transmitted, the supported operations, how the message is going to be transmitted, where the service is located, etc. – because it cannot be generalised to any services provided by artifacts, given that the technical details depend on the specific implementation.

On the artifact side, it should be pointed out that there is currently no semantic language similar to WSDL to describe the implementation details of a service: so, further investigation is needed in order to evaluate possible candidates, or possibly define some new ad hoc language.

Among the existing composing languages, BPEL [2], for instance, could be worth exploration, despite some overlaps with OWL-S; this and other approaches are briefly considered below.

4 OWL-S and Artifacts: a simple example

As outlined in the previous section, OWL-S makes it possible to express the artifact features (the function description, describing what the service does; the usage interface, describing how to access it; and the operating instructions, explaining how the service works) via the service profile, the service model and the service grounding, respectively.

However, while expressing the service offered by an artifact via the OWL-S service profile notion is relatively simple from the conceptual viewpoint, mapping the usage interface and the operating instructions onto the service model and the service grounding notions calls for some hypotheses on both the artifact model and its supporting infrastructure – i.e., about the environment where the artifact itself lives and is called to operate.

To this end, in the following we adopt TuCSoN tuple centres [19] as the the reference model for (social) artifacts, and the TuCSoN technology [22] as the corresponding infrastructure. For the sake of concreteness we consider the case of a flight tracking service, and present it from its OWL-S description to its implementation onto the TuCSoN infrastructure.

4.1 TuCSoN Infrastructure

TuCSoN[2] is an example of agent coordination infrastructure supporting a notion of social artifact, namely a coordination artifact called *tuple centre*. Tuple centres are programmable tuple spaces, that agents access by writing, reading, and consuming tuples – that is, ordered collections of

[2]TuCSoN technology is available as an open-source project at the TuCSoN Web site http://tucson.sourceforge.net.

```
<profile:Flight_Tracking
  rdf:ID="Profile_Flight_Tracking_Service">
  <service:presentedBy
    rdf:resource=
      "&flightService;#FlightTrackingService"/>
  <profile:has_process
    rdf:resource=
      "&flightProcess;#FlightTrackingProcessModel"/>
  <profile:serviceName>
     Flight_Tracking_Agent
  </profile:serviceName>
  <profile:textDescription>
    This agentified service provides the opportunity to
    request information about the delay and the location
    of a flight whose number is given as input.
  </profile:textDescription>
</profile:Flight_Tracking>
```

```
<process:AtomicProcess rdf:ID="FlightTracking">
 <process:hasInput>
  <process:Input rdf:ID="FlightNumber">
   <process:parameterType rdf:resource="&xsd;#string"/>
  </process:Input>
 </process:hasInput>
 <process:hasPrecondition
      rdf:resource="#FlightNumberExists"/>
 <process:hasEffect>
  <process:ConditionalEffect rdf:ID="InfoRequestEffect">
   <process:ceCondition
         rdf:resource="#FlightNumberExists"/>
   <process:ceEffect rdf:resource="#InfoProvided"/>
  </process:ConditionalEffect>
 </process:hasEffect>
</process:AtomicProcess>
```

Figure 8: Some fragments from the OWL document that describe a) the service profile, on the left, and b) the service model, on the right, of the flight tracking service.

heterogeneous information chunks – via simple communication operations (*out*, to insert a tuple; *rd*, to read a tuple; *in*, to remove a tuple; tuples are accessed associatively. While the behaviour of a tuple space in response to communication events is fixed, the behaviour of a tuple centre can be programmed by defining a set of specification tuples (expressed in the ReSpecT language [18]), which define how a tuple centre should react to incoming/outgoing communication events. As a result, tuple centres can be seen as general-purpose customisable coordination artifacts, whose behaviour can be dynamically specified, forged and adapted so as to automate coordination among agents [24].

4.2 Example

In case of the flight tracking service, a tuple centre can be used to mediate the interaction between the user requesting an information (delay and location) about a given flight and the agent charged of retrieving this information.

As a service provided by an artifact, the flight tracking service can be described through:

- a function description — here providing information about the delay and the location of a selected flight;
- a usage interface — that is, the set of communication operations: here, *out*, to insert a tuple in the tuple centre; *rd*, to read a tuple from a tuple centre; *in*, to remove a tuple from the tuple centre;
- operating instructions — that is, the procedure to be followed by the user in order to obtain the desired flight information; here, the user should first request the information by inserting a request tuple such as `flight_request(flightNumber)`, then retrieve the desired information by reading a response tuple such as `flight_status(flightNumber,Delay,Location)`.

In the OWL-S model[3], the tuple centre offering the flight tracking service is characterised by a function description (what the service does), is described by some operating instructions (how the service works), and supports a usage interface (how to access to the service).

In turn, while the function description is provided through a service profile which contains a simple text description (see Figure 8.a), the service model can be described in terms of inputs, outputs, preconditions and effects. Figure 8.b depicts a fragment of the corresponding OWL-S document which specifies the input (`FlightNumber`), the condition under which the information will be provided (`FlightNumberExists`) and the effect (`InfoProvided`).

Finally, in order to represent the usage interface, we need a language similar to WSDL for describing both the type of actions (in, out, rd) and the correct sequence for using them and

[3]For the sake of shortness, here we show only a sketch of the full OWL-S documents (a specialisation of the generic OWL-S documents presented in Section 3.3).

so obtain the information required. As explained in the previous section, further investigation is needed in order to evaluate possible candidates among the existing languages or possibly define a new ad hoc language.

5 Related work

The approach examined in this paper combines Semantic Web and the theory of coordination with the aim of describing the services provided by artifacts via OWL-S. There are also other Web Service composition languages, such as BPEL4WS [2] and WSCI [5].

Business Process Execution Language for Web Services (BPEL4WS) enables the specification of executable business processes, covering also Web Services, and business process protocols in terms of their execution logic or control flow. Executable business processes specify the behaviour of individual participants within Web Service interactions and can be invoked directly, whereas business process protocols abstract from internal behaviour to describe the messages exchanged by the various Web Services within an interaction. The main drawback of BPEL's approach is that it enables only the composition of Web Services, while OWL-S also supports Web Services' publication and discovery.

Many other approaches could be used to express how a service is implemented in terms of action protocol (the remaining part of operating instructions) and usage interface. For instance, one could think of a semantic language for describing services offered by an artifact based on the research area on semantic tuple spaces [13, 26]. As an alternative, non-semantic approaches could be followed, such as [29, 28]: for instance, process algebra – a standard approach to describe interaction protocols in the distributed systems field – could be adopted as a specification language for operating instructions.

Moreover, it could be interesting to study not only OWL-S but also the Web Service Modelling Ontology (WSMO) [25], which shares with OWL-S the vision that ontologies are essential to support automatic discovery, composition and inter-operation of Web services, although they greatly differ in the approach to achieve these results [4]. Indeed, WSMO defines a conceptual framework within which the ontologies developed through OWL-S have to be created. Futhermore, while OWL-S does not make any distinction between the types of Web Services, WSMO specifies *mediators* – that is, mapping programs aimed at solving the inter-operation problems between Web Services (such as translation between ontologies, between the messages produced by a Web service and those expected by another Web service, etc). In the process of mediator definition, WSMO introduced a taxonomy of possible mediators that helps to define and classify the different tasks that mediators are supposed to solve.

6 Conclusions

By drawing an analogy between the OWL-S model and the artifact notion, in this paper we exploited OWL-S for describing the services offered by artifacts. We first presented the notion of artifacts for MAS and how the services provided by such artifacts could be described; then, we presented a preliminary proposal for describing the services provided by artifacts based on OWL-S and a simple example of such a proposal. One conclusion is that OWL-S can be exploited rather easily to express the function description – that is, what the service does – and the operating instructions – that is, its preconditions, effects, inputs and outputs – but can not be directly exploited, in its current version, for describing the usage interface which is strictly related to the implementation of the service

Accordingly, future work will be devoted on the one hand to explore semantics languages for the description of usage interface; on the other, to compare OWL-S and WSMO for artifact description, both from an abstract viewpoint and in concrete examples of artifacts.

References

[1] H. Peter Alesso and Craig F. Smith. *Developing Semantic Web Services*. A. K. Peters, Ltd., 2004.

[2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. Business Process Execution Language for Web Services. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[3] Anupriya Ankolekar. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl- s/1.0/owl-s.pdf, 2003.

[4] Anupriya Ankolekar, David Martin, Deborah McGuinness, Sheila McIlraith, Massimo Paolucci, and Bijan Parsia. Owl-s' Relationship to Selected Other Technologies. http://www.w3.org/Submission/2004/SUBM-OWL-S-related-20041122/.

[5] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, and Sinisa Zimek. Web Services Choreography Interface. http://www.w3.org/TR/wsci/, 2002.

[6] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible Markup Language (XML) 1.0 (Third Edition). http://www.w3.org/TR/2004/REC-xml-20040204.

[7] Dan Brickley and R.V. Guha. Rdf Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/tr/rdf-schema.

[8] Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors. *The Semantic Web: Research and Applications*, volume 3053 of *Lecture Notes in Computer Science*, ESWS 2004, Heraklion, Crete, Greece, may 2004. Springer.

[9] Liliana Cabral, John Domingue, Enrico Motta, Terry Payne, and Farshad Hakimpour. Approaches to semantic web services:an overview and comparisons. In Christoph Bussler, John Davies, Dieter Fensel, and Rudi Studer, editors, *The Semantic Web: Research and Applications*, volume 3053 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2004. First European Semantic Web Symposium, ESWS 2004 Heraklion, Crete, Greece,.

[10] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, W3C. http://www.w3.org/TR/wsdl, 2001.

[11] FIPA. Fipa-acl. `http://www.fipa.org/specs/fipa00061/index.html`.

[12] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. Simple Object Access Protocol. http://www.w3.org/TR/soap12-part1/, 2003.

[13] Deepali Khushraj, Ora Lassila, and Tim Finin. stuples: Semantic tuple spaces. In *1st Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous 2004)*, pages 267–277, MobiQuitous 2004, Boston, Massachusetts, USA, August 2004.

[14] Graham Klyne and Jeremy Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. http://www.w3.org/rdf, 2004.

[15] Jaceký Kopecky, Dumitru Roman, Matthew Moran, and Dieter Fensel. Semantic web services grounding. *aict-iciw*, 0:127, 19–25 February 2006. International Conference on Internet and Web Applications and International Conference on Internet and Web Applications and Services.

[16] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/tr/owl-features, 2004.

[17] Andrea Omicini. Formal ReSpecT in the A&A perspective. In Carlos Canal and Mirko Viroli, editors, *5th International Workshop on Foundations of Coordination Languages and Software Architectures (FOCLASA'06)*, pages 93–115, CONCUR 2006, Bonn, Germany, 31 August 2006. University of Málaga, Spain. Proceedings.

[18] Andrea Omicini and Enrico Denti. Formal ReSpecT. *Electronic Notes in Theoretical Computer Science*, 48:179–196, June 2001. Declarative Programming – Selected Papers from AGP 2000, La Habana, Cuba, 4–6 December 2000.

[19] Andrea Omicini and Enrico Denti. From tuple spaces to tuple centres. *Science of Computer Programming*, 41(3):277–294, November 2001.

[20] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. *Agens Faber*: Toward a theory of artefacts for MAS. *Electronic Notes in Theoretical Computer Sciences*, 150(3):21–36, 29 May 2006. 1st International Workshop "Coordination and Organization" (CoOrg 2005), COORDINATION 2005, Namur, Belgium, 22 April 2005. Proceedings.

[21] Andrea Omicini, Alessandro Ricci, and Mirko Viroli. Coordination artifacts as first-class abstractions for MAS engineering: State of the research. In Alessandro F. Garcia, Ricardo Choren, Carlos Lucena, Paolo Giorgini, Tom Holvoet, and Alexander Romanovsky, editors, *Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications*, volume 3914 of *LNAI*, pages 71–90. Springer, April 2006. Invited Paper.

[22] Andrea Omicini and Franco Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, September 1999. Special Issue: Coordination Mechanisms for Web Agents.

[23] Alessandro Ricci, Claudio Buda, Nicola Zaghini, Antonio Natali, Mirko Viroli, and Andrea Omicini. simpA-WS: An agent-oriented computing technology for WS-based SOA applications. In Flavio De Paoli, Antonella Di Stefano, Andrea Omicini, and Corrado Santoro, editors, *WOA 2006 – Dagli oggetti agli agenti: sistemi grid, p2p e self-**, pages 1–3, Catania, Italy, 26–27 September 2006. Technical University of Aachen.

[24] Alessandro Ricci, Andrea Omicini, and Enrico Denti. Activity Theory as a framework for MAS coordination. In Paolo Petta, Robert Tolksdorf, and Franco Zambonelli, editors, *Engineering Societies in the Agents World III*, volume 2577 of *LNCS*, pages 96–110. Springer-Verlag, April 2003. 3rd International Workshop (ESAW 2002), Madrid, Spain, 16–17 September 2002. Revised Papers.

[25] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubn Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. volume 1, pages 77–106, 2005.

[26] Robert Tolksdorf and Dirk Glaubitz. Xmlspaces for coordination in web-based systems. In *10th IEEE International Workshops on Enabling Technologies (WETICE 2001)*, pages 322–327, WETICE 2001, Washington, DC, USA, august 2001. IEEE Computer Society.

[27] Mirko Viroli, Andrea Omicini, and Alessandro Ricci. Engineering MAS environment with artifacts. In Danny Weyns, H. Van Dyke Parunak, and Fabien Michel, editors, *2nd International Workshop "Environments for Multi-Agent Systems" (E4MAS 2005)*, pages 62–77, AAMAS 2005, Utrecht, The Netherlands, 26 July 2005.

[28] Mirko Viroli and Alessandro Ricci. Instructions-based semantics of agent mediated interaction. In Nicholas R. Jennings, Carles Sierra, Liz Sonenberg, and Milind Tambe, editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, volume 1, pages 102–110, AAMAS 2004, New York, USA, jul 2004. ACM.

[29] Mirko Viroli, Alessandro Ricci, and Andrea Omicini. Operating instructions for intelligent agent coordination. *The Knowledge Engineering Review*, 2006.