Engineering a Brokering Framework for Providing Semantic Services to Agents on Lightweight Devices

Nikolaos I. Spanoudakis[1] and Pavlos Moraitis[2]

Abstract. This paper describes an approach towards allowing lightweight nomad devices like mobile phones to access semantic services that have either been advertised by agents or follow the semantic web services paradigm. The limitations of lightweight devices like lack of capability to process XML documents or to deal with complex data types and perform computationally demanding tasks are overcome by using this approach. Thus, we consider that when a user or agent is in a nomad or mobile context this approach can aid him in searching for and acquiring simple or complex - added value services from the web.

1 INTRODUCTION

There is a growing interest on the launching of agents on lightweight devices and that comes from many different business and research sectors, including the Ambient Intelligence, the infomobility, the collaborative working environments and others.

Lightweight devices pose certain limitations on the available resources (CPU speed, memory capacity, storage capacity, etc) for programs. Services are becoming semantic so that agents can adequately locate and execute them in order to achieve their goals. Semantic services imply the use of XML, RDF and OWL [13] technologies. The use of such technologies requires more than what is available on a lightweight device.

Brokering is the solution to this problem, since the broker can always be on a server computer side having access to needed computational and storage resources. The nomad devices residing agents need access to services that require computational power (for example filtering 100 hotels in order to present to a user the best 10). Such services are proposed to be offered by server-based provider agents. Important works from the agent technology domain, but also from that of the semantic web, have addressed the issue of brokering and matchmaking ([3], [11], [7]). However, these works lack the support for agents on the specific context of being resident on nomad, lightweight devices.

Our work builds on this previous work and provides a framework for defining services using the OWL-S [12] paradigm and making them available to lightweight devices. We use the FIPA-ACL [2] standard for defining the agent messages that are used by our novel interaction protocol. The content of the messages is encoded using the FIPA-SL [2] language for lightweight agents.

In section 2 we describe our approach in detail and we conclude with a discussion in section 3. We use italics in order to type concepts of the ontology that we developed, their properties and ACL message performatives.

[1] Singular Software SA, Greece, email: nspan@singular.gr
[2] René Descartes University, Paris, France,
email: pavlos@math-info.univ-paris5.fr

2 THE BROKERING FRAMEWORK

In order to address our requirements we used the broker agent type [6], who can actively interface between the requester and provider agents by facilitating the requested service transaction. Thus, all communication between requester and provider agents has to go through the broker. In this process, the requester's identity is unknown to the service provider. Thus, assuming the business role of a service aggregator the broker services his customers using providers as resources. The service requesters are assumed to be aware of the services that they need. In our system, the role of the broker is to either select the best service for the requester, or to redirect the request to the appropriate broker. The added value of our approach is the service protocol that allows for anonymous brokering for agents in a nomad context adding, for the first time, the possibility to broker subscription services.

The matchmaking process is a subset of LARKS [11], suited for the nomad device applications domain. In this context, the requester is assumed to use the same ontology with the broker. Our process's novelty lies in the manipulation of the inter-agent messages content that is delivered using the LEAP protocol, which poses specific limitations and is in byte code format. Heterogeneous services are wrapped by service provider agents who advertise and offer services using the application domain ontology.

Moreover, brokers can be distributed and each one can specialize to a specific domain of services. We follow the notion of broker specialization of Infosleuth [9]. We model this requirement using a broker capability property concept allowing a broker to define its specialization in terms of service parameters constraints and share it with other brokers. The advantages of our approach compared to Infosleuth are a) brokers do not simply exchange their advertisements but define their special capabilities over the provided services in the domain, b) the requester agent doesn't have to define a search policy for the broker, and, c) compatibility with FIPA standards.

The way to profile the services, the matchmaking process and the brokering protocol are described in detail in the following paragraphs.

2.1 Service Profiling

For service profiling we follow the semantic web trend and thus are compatible to OWL-S. The service profile (SP) defines the type of the service (e.g. mapping service), describes input and output parameters, as well as preconditions and post-conditions. Here we would like to note that in the service parameters definition we have defined semantics for declaring a parameter as optional or mandatory.

2.2 The matchmaking process

Having defined the input requests and profiles we can proceed to defining the matchmaking process. We need to match a service advertisement to a service request. Two types of matching best serve our needs ([7], [11]): a) *the exact matching*, which demands that the advertised service has the same semantics and equal input/output parameters with the requested service, and b) the *plug-in matching*, which allows for the advertisement to have more input/*output parameters* than the ones requested. The exact matching is obviously always preferable.

Our matchmaking algorithm gradually filters the repository of advertisements until the one best to serve the request is found. Three types of filters, originally proposed by [11], are used: a) *Semantic Match (SM)* searches the service *profile* advertisements (PAs) for a service that matches the request (RP), b) *Profile Match (PM)* searches the PAs provided by the SM for input and output parameters that match those of the request. PM determines which PAs are exact or plug-in matches and sorts them accordingly, and, c) *Constraint Match (CM)* determines which of the PAs provided by the SM, match the constraints of a request. CM is performed to the sorted list provided by PM and either the first or all PAs that successfully match the constraints are selected depending on the broker's policy.

A special CM is needed before SM (named Pre-CM) so that the broker agent (BA) can determine if he can serve the request or it needs to redirect the request to another broker. Thus, broker capabilities are described as constraints for parameter values. For querying the PAs repository we use the RDQL (RDF Query Language) of the Jena open source tool [5].

Thus, according to our matchmaking algorithm, the broker first applies the pre-CM filter. If he can handle the request, he then sequentially applies the three other filters (SM, PM, CM) to his PA repository.

Technical challenges were related to this matchmaking process. The first was relevant to the transformation of a LEAP message to RDF format for filtering. In order to overcome it we use the LEAP codec for decoding a message at the broker side and then encode it again with the use of the RDF codec [4] (see an example in Figure 1). Thus, the request gains the necessary semantics so as to be processable by Jena. From that point forward the matchmaking process takes place and whenever the response of the service provider is ready, it is encoded at the broker side with the LEAP codec and sent to the lightweight agent requester.

Another problem that we had to overcome is that FIPA ACL allows for a single ontology to be included in the content of an ACL message. Thus, it is not possible to use different existing ontologies when defining the ACL protocols (e.g. import all OWL-S namespaces and use their concepts). That is why we added in our ontology all the concepts that we need in order to define a service profile similar to OWL-S. However, these are reusable since the Protégé tool [10] that we used in order to define our ontology doesn't associate namespaces to ontologies before deployment.

For describing an input/output parameter within a request for a service we created the *CallParameter* concept. There, we encountered another technical challenge related to the fact that we used the LEAP codec and thus, we could not add dynamically a value concept to a parameter property as we could easily do in an RDF document. This happens because the agent on the nomad device uses ontology java beans [1] in order to represent ontology concepts. These beans are normal Java classes containing properties that cannot be ambiguous, i.e. defined of type *Object* because the LEAP encoding and decoding process needs specific data types to instantiate as properties of concepts.

```
<?xml version="1.0"?>
<rdf:RDF   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#"
xmlns:fipa-rdf="http://www.fipa.org/schemas/FIPA-RDF#"
xmlns:O="http://imagine-it.eu/ontology#">
 <rdf:object>
  <fipa-rdf:CONTENT_ELEMENT>
   <rdf:Description>
    <rdf:type>http://imagine-
it.eu/ontology#RequestForEService</rdf:type>
    <O:agent>
     <rdf:Description>
      <O:name>broker@nspan2kp:1099/JADE</O:name>
     </rdf:Description>
    </O:agent>
    <O:requestEService>
     <rdf:Description>
      <O:serviceName>http://imagine-it.eu/ontology#
createMap</O:serviceName>
      <O:hasParameterIn>
       <rdf:Seq>
        <rdf:li>
         <rdf:object>
          <rdf:type>http://imagine-it.eu/ontology#
CallParameter</rdf:type>
          <O:withName>http://imagine-
it.eu/ontology#forCountry</O:withName>
          <O:withType>http://www.w3.org/2001/XMLSchema#
string</O:withType>
          <O:withStringValue>DE</O:withStringValue>
         </rdf:object>
        </rdf:li>
        <rdf:li>
         <rdf:object>
          <rdf:type>http://imagine-it.eu/ontology#
CallParameter</rdf:type>
          <O:withName>http://imagine-it.eu/ontology#
screenSize</O:withName>
          <O:withType>http://imagine-it.eu/ontology#
ScreenSize</O:withType>
          <O:withScreenSizeValue>
           <rdf:Description>
            <O:hasPixelsHeight>200</O:hasPixelsHeight>
            <O:hasPixelsWidth>320</O:hasPixelsWidth>
           </rdf:Description>
          </O:withScreenSizeValue>
         </rdf:object>
        </rdf:li>
       </rdf:Seq>
      </O:hasParameterIn>
     </rdf:Description>
    </O:requestEService>
   </rdf:Description>
  </fipa-rdf:CONTENT_ELEMENT>
 </rdf:object>
</rdf:RDF>
```

Figure 1. A Service Request Message (RDF)

We overcame this issue by defining all possible values that a parameter can have as properties of the *CallParameter* concept. The basic properties of the *CallParameter* in an OWL/RDF setting would be the *withName*, *withType* and *withValue*. In this case, however, we must cater for all possible types defined in our ontology concepts. Figure 1 shows an instance of a request message related to a specific application [8], which is based on the *http://imagine-it.eu/ontology#* namespace. The reader can observe the *hasParameterIn* RDF sequence element that is composed of a list of *CallParameter* elements that have a name (parameter *withName*), a type (it can be a simple data type such as string or a complex type like for example the *ScreenSize* type) and a value corresponding to the type.

For example, for the *ScreenSize* type (these types are also related to application [8]) the relevant property of *CallParameter* that is used is the *withScreenSizeValue*. Similarly the *forCountry CallParameter* is of type *string* and has the *withStringValue* property. Thus, the *CallParameter* concept has as many such

properties as the number of the data type concepts defined in our ontology. However, for each instance the requester defines the *withName* and *withType* (*withType=PropertyType*) properties and the relevant *withPropertyTypeValue* property. It is obvious that a designer can define appropriate parameter types related to his own application.

2.3 The Service Protocol

The service provisioning protocol is presented in Figure 2 in the form of a FIPA interaction diagram [2].

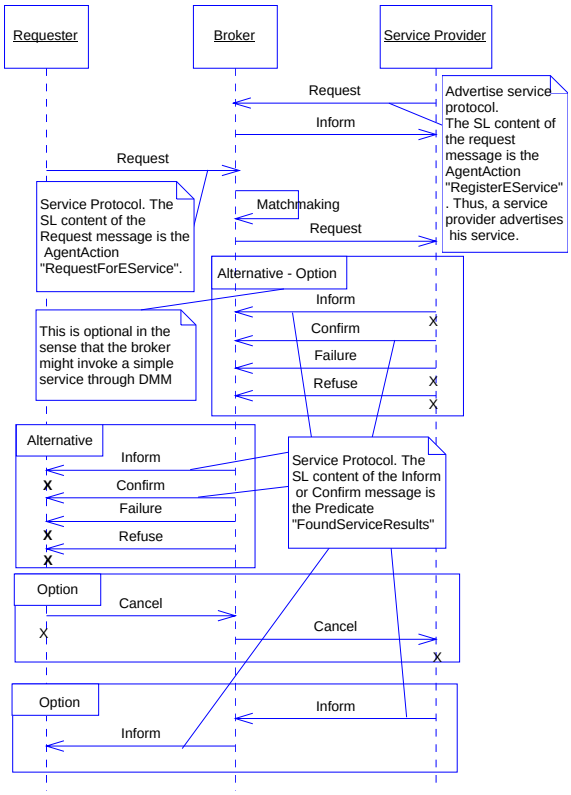

Figure 2. Service Protocol Definition

The service protocol can also be used for subscription services provisioning. We use the FIPA *Request*, *Inform*, *Refuse*, *Failure* and *Confirm* performatives. The important *AgentAction* and *Predicate* concepts [1] that are used as content in the ACL message are also presented. The participants are the Requester agent type (RE), the Broker agent type (BR) and the service provider agent type (SP). In the case of distributed brokering the SP is another broker, who considers the broker, who received the original request, as a RE (implementing the relevant part of the same protocol).

Finally, for the service subscription protocol, the broker always retains the recent addresses of the communicating agents so as to be able to forward new messages to the latest address of the requester agent. This is important since the requester agent is on a nomad device and usually is assigned a dynamic IP whenever he accesses the network.

3 DISCUSSION

We used this brokering framework in the context of IST project Im@gine IT in the infomobility sector domain [8]. An Im@gine IT prototype has been developed and deployed. Two added value service providers were developed.

This work is meant as an extension of important works in the brokering domain ([3], [11], [7]), towards offering semantic services to nomad devices. We provide a complete solution for the nomad devices service provisioning including not only simple services but also the delegation of complex tasks and subscription services. The solution is composed of a protocol, a service profiling scheme and the relevant matchmaking process.

As future work we aim to enrich the broker with the capability to use directly OWL-S services advertisements (along with those received by other agents) where the broker performs the service grounding himself.

Acknowledgements

We gratefully acknowledge the European Commission Information Society Technologies (IST) Programme and specifically the Integrated Project (IP) “Ambient Intelligence System of Agents for Knowledge-based and Integrated Services for Mobility Impaired users” (ASK-IT, IST-2003-511298) project for contributing in the funding of our work.

REFERENCES

[1] Caire, G., Van Aart, C., Bergenti, F., Pels, R. : Creating and Using Ontologies in Agent Communication. Workshop on Ontologies and Agent Systems at AAMAS 2002

[2] FIPA, Foundation for Intelligent Physical Agents, www.fipa.org

[3] Gomez, M., Abasolo, C., Plaza, E.: Domain-independent ontologies for cooperative information agents. In: Cooperative Information Agents V. LNAI 2128, Springer-Verlag, 2001, p. 118-129

[4] JADE, Java Agent Development Environment, http://jade.tilab.com/

[5] Jena, A Semantic Web Framework for Java, http://jena.sourceforge.net/

[6] Klucsh M., Sycara K.: Brokering and Matchmaking for Coordination of Agent Societies: A Survey. In Omicini et al (editor), Coordination of Internet Agents, Springer, 2001

[7] Li. L. and Horrocks, I.: A software framework for matchmaking based on semantic web technology. Int. Journal of Electronic Commerce, Vol. 8, No 4, 2004

[8] Moraitis, P., Petraki, E. and Spanoudakis, N.: An Agent-Based System for Infomobility Services. In: 3rd European Workshop on Multi-Agent Systems (EUMAS2005), Brussels, Belgium, December 7 - 8, 2005

[9] Nodine, M., Bohrer, W., Hee Hiong Ngu, A.: Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth. In Proc. of the International Conference on Data Engineering, Sydney, Australia, March 23-26, 1999

[10] Protégé, An Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu

[11] Sycara, K., Widoff, S., Klusch, M. and Lu, J.: LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. JAAMAS, 5, 173–203, 2002.

[12] The OWL Services Coalition (Martin, D. et al.): Bringing Semantics to Web Services: The OWL-S Approach, Proc. of the 1st Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC2004), July 6-9, 2004, San Diego, California, USA.

[13] W3C, The World Wide Web Consortium, www.w3c.org