Using Ontological Concepts for Web Service Composition

Claude Moulin
Compiègne University of Technology
UMR CNRS 6599, Heudiasyc, France
claude.moulin@utc.fr

Marco Sbodio
Hewlett Packard – Italy Innovation Center
C.so Trapani 16, 10139 Torino, Italy
marco.sbodio@hp.com

Abstract

This paper describes an approach for a composition of web services based on their semantic descriptions. The process section of OWL-S service descriptions is built with references to ontology concepts which represent service input and output data types.

We present a engine that receives a request containing a concept (OC) corresponding to a service output and a set of concepts (ICs) corresponding to a service inputs. The engine produces a sequence of services whose first element has ICs as inputs and whose last element has OC as output. The result of the composition is described as a BPEL process.

1. Introduction

The research area targeted by the TERREGOV Project is the "*Usability and Applicability in large, complex and real-life environments for coherent development of interoperable government services*". To tackle this research issue, the project focuses on the requirements of governments at local, intermediate and regional levels for flexible and interoperable tools to support the change towards eGovernment services. The "Web Service" paradigms appears as a major brick for applications interoperability and integration. However, the implementation of complex and flexible government processes with Web Services still requires additional efforts.

The enhancement of Web Services with semantics is a crucial step for making as easy as possible access to these services both by 3rd party applications and human users. Some efforts propose methods for standardizing e-government services development at the national and international levels [7], [8], [9]. Some procedures will only embed already existing legacy systems but new procedures will substitute hand made tasks. Elementary processes or sub processes must be completely transparent to civil servant. Some tasks containing sequences of subtasks must be dynamically elaborated by algorithms at run time.

Discovery of services leads to build sequences of service based on constraints. We consider the following issue: having some input and output data type, find services and build a chain whose first service accepts the input type and the last service accepts the output type? Two services are chained, if output of the first one is input of the second one.

2. Scenario

We describe here a simplified but real use case developed in the TERREGOV project, which focuses on services that support socio-economic assistance processes for citizens of the region of Venice. It involves a citizen asking for socio-economic assistance. A specific process must be enacted in order to decide the eligibility of the citizen to this kind of assistance. The process begins with the collection of relevant personal, medical and economical data relative to the citizen. The local municipality uses a web service for obtaining the personal information (first and last name, residence, etc.). The input of this service is the personal identifier (in Italy, the "fiscal code"). A local health care administration uses a web service for obtaining the relevant medical data. The input of this service is the patient's identifier, i.e. the "health card number".

Unfortunately the two web services require different inputs: the "fiscal code" and the "health card number". The solution resolving this case is a composite process establishing the minimal list of inputs to ask for. The system searches other services which could return the other needed data.

In this case, the minimal list of inputs is reduced to the fiscal code because it exists a service which can produce an health care number given a fiscal code. The existence of the third service is completely transparent in the process.

3. Environment

3.1. Semantic service registry

It is necessary to build new registries, containing enough semantic information for service discovery [10] (that common UDDI[1] registries do not contain). They allow the publication of semantic service descriptions and some service discovery mechanisms based on service features. We have decided to adopt OWL-S[2], even if this standard is still evolving, because some technical resources such as Java libraries are already available and will be updated.

In OWL-S a service presents a profile, is described by a model and supports a grounding. The profile is related with the knowledge domain of the service. The model (or process) is used to define the inputs, outputs, preconditions and effects of a service. The grounding describes the implementation of a service. It is linked with WSDL[3] service descriptions.

We use profile and process for service discovery. The concepts such as *HealthCard* and *HealthCardId* are defined in our ontology.

3.2. Workflow engine extension

Current technologies address the service composition issue through web services orchestration; the major industry initiative is BPEL4WS[4]. BPEL4WS distinguishes between *executable processes* (actual behavior model of a participant in a business interaction) and *abstract process* (technological interface description of business processes). BPEL4WS focuses on representing *static compositions*, where both the flow of the process and its building blocks (the web services) are known a priori.

In order to achieve the dynamic composition of web services, it is also necessary to extend this traditional workflow engines, with the addition of modules allowing the dynamic discovery of web services based on their semantic information. Accordingly, abstract and executable processes have to be updated in such a way that dynamic processes based on qualified patterns could be implemented at run time. The algorithms they perform, use the ability of discovering available web services.

[1] http://www.uddi.org/

[2] http://www.daml.org/services/owl-s/

[3] http://www.w3.org/TR/wsdl

[4] http://www-128.ibm.com/developerworks/library/ws-bpel/

3.3. Ontology

The backend of the application interoperability in the eGovernment domain relies on ontologies. They have to be rich enough for fulfilling several goals. The first one is the complete description of services allowing a dynamic discovery. Search engines have to merge elements extracted from service descriptions and ontology concept for making the right inferences. Our eGovernment ontology is also used for indexing documents and resources in knowledge bases. Elements for building the upper eGovernment domain ontologies can be found in [9].

4. Model and Algorithm

This section gives a description of the data model and the algorithm used in our system. As a general background we assume to have a set of web services exposing a single operation and having both WSDL and OWL/S descriptions of these services. The OWL-S service description provides information about inputs and output referring to concepts (C_i) defined in an ontology. We write $\{C_1, C_2, \ldots, C_n\} \rightarrow WS \rightarrow C_x$ to represent a web service WS having a set of inputs represented by concepts $C_1, C_2, \ldots, C_n$ and whose output is represented by concept C_x.

Dynamic composition of Web Services is done building sequences of web services that use available inputs (specified as ontology concepts) and produce a required output (specified as an ontology concept). We write $\{C_i, C_j, \ldots, C_p\} \rightarrow < S > \rightarrow C_y$ to represent a sequence that consuming inputs defined by concepts C_i, C_j, …, C_p produces an output represented by the concept C_y.

4.1. Data model

The relations among inputs/output concepts C_i and web services are extracted from the descriptions of the services, and are mapped in an internal representation model consisting of RDF statements. Let's consider a simple example: a web service WS_1 and the following fragment of its OWL/S description:

```
<process:AtomicProcess rdf:ID="Process">
  <process:hasInput>
    <process:Input rdf:ID="in0">
      <process:parameterType
        rdf:resource="&eg;C1"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input rdf:ID="in1">
```

```
      <process:parameterType
        rdf:resource="&eg;C2"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="out0">
      <process:parameterType
        rdf:resource="&eg;C3"/>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>
```

The above fragment describes $\{C_1,C_2\} \rightarrow WS_1 \rightarrow C_3$. WS_1 is identified by the URI pointing to its OWL-S description. We build our internal representation model as a set of RDF statements which maps the relations among C_1, C_2, C_3 and WS_1:

@prefix eg: <urn:x-hp-terregov:eg/> .
<eg:C1> <eg:i> <eg:WS1>: "C1 input WS1"
<eg:C2> <eg:i> <eg:WS1>: "C2 input WS1"
<eg:WS1> <eg:o> <eg:C3>: "WS1 output C3"

The internal representation model is built at start-up time reading all information from the available OWL/S descriptions, and is intended to be dynamically updated whenever a new service is published (i.e. a new OWL/S description becomes available). We only assume that any OWL/S description is identifiable and referable through a URI.

The internal representation model serves as a data model for the composition algorithm described in the following section, and avoids re-iterated readings of the service description. The composition algorithm is essentially based on the inputs/output of services, and our model captures exactly this information.

4.2. Algorithm

The algorithm that has been implemented is a simple composition algorithm. It is intended to discover sequences of web services invocations consuming a set of available inputs and return an expected output.

The algorithm is an implementation of the recursive back-changing pseudo-code available form the work done in [1] and listed here:

```
initialization:
weHave = {input set}; weWant = {output set};
findServiceChain (weHave, weWant)
  {svcs = getServicesOutputtingWeWant(weWant);
   foreach service in svcs
    {chain = new chain;
    foreach input in service.inputs
      if input not in weHave
        {newSvcs =
          findServiceChain(
            weHave, service.inputs);
         chain.add(newSvcs);}
    if all service.inputs in weHave
      {chain.add(service);
       return chain;}
    }
  return null;} // no chain found
```

Our implementation uses the internal representation model described in section 4.2 as the data model.getServicesOutputtingWeWant(…) uses RDQL[5] to query this model; for example the following RDQL query, returns all services having concept C_3 as output:

```
SELECT ?x
WHERE (?x  <urn:x-hp-terregov:eg/o>
  < urn:x-hp-terregov:eg/C3> )
```

Similarly we retrieve information on required inputs of a service with a query on the data model.

As a generalization, the implemented algorithm can generate a set of sequences satisfying some conditions, where conditions are primarily expressed as the set of available inputs and the required output. Additional conditions could express other constraints. These additional conditions could allow to search the "optimal" sequence of services.

5. Architecture

This section gives a description of the system architecture. The implementation is based on Java (HP Jena framework[6], Mindswap OWL/S API [2]). We isolate the *composition concern*, which requires semantic capabilities, from the *execution concern*, which requires an execution framework:

- *Composition concern*: we exploit OWL/S features for reasoning on service constraints and capabilities.
- *Execution concern*: we exploit BPEL4WS as a formalism having a strong execution orientation.

The major components of the architecture are:

- Interoperability and coordination layer: it is the system entry point. It allows programmatic requests of services specifying inputs and outputs.
- Repository: it is an adapter component that offers access to service description.
- Analyzer/Checker: it is in charge of implementing the algorithm that explore available services and try to dynamically build the service sequence.
- Instantiator: it is an adapter component which translates a web service sequence dynamically composed by the "Analyzer/Checker" into a BPEL4WS description, and instantiates it within a classical BPEL engine.
- Execution Engine/Monitoring: it is a classical BPEL engine, with monitoring capabilities

[5] http://www.hpl.hp.com/semweb/rdql.htm

[6] http://jena.sourceforge.net/index.html

6. Related Works

Our approach is similar to the one described in [1], but we have augmented the capabilities of standard BPEL4WS execution engine, with automated reasoning on OWL/S descriptions. Both approaches try to build web services sequences that consume a set of inputs, and produce a set of expected outputs.

Mandell and McIlraith proposed a "Semantic Discovery Service" (SDS) which sits between the BPEL4WS engine and the service partners. The SDS accomplishes both dynamic discovery and dynamic composition, and act as a proxy between the discovered partners and the BPEL4WS engine. In our approach we separate concerns: the semantic UDDI registry is concerned with dynamic discovery; the eProcedure module is concerned with the dynamic composition; the BPEL4WS engine is concerned with the execution

Another approach to web service composition is described in [3]. A semi-automatic approach at dynamic composition is proposed: an operator is assisted in the composition of a web service sequences by a program, which presents at each step of the composition the possible matching services. In our case, the composition must be dynamic and implemented at run time.

The matchmaking algorithm given in [4] tests if a request can provide all required inputs of a service and if the offered output also satisfies the demands. The test can have several degrees of accuracy (for example exact, or subclass/superclass matching). A similar work has also been done within the matcher for the LARKS developed by Sycara et al. [5]. some interesting insight on the matchmaking problem is given in [6].

7. Conclusions

Besides semantics of inputs/outputs, the preconditions and effects constraints described by OWL/S, can be used for service discovery. Some request, for example, could lead to reject services with undesired effects. We also intend to investigate the possibility of reinforcing the matching using subclasses and superclasses relations as in [4][3].

Regarding the dynamic composition problem, we are aware of the potential performance issues related to the simple recursive algorithm that we're currently using. For this reason we intend to investigate the possibility of building a *composition-engine* for building arbitrary and consistent services sequences.

This component could explore the available services, and try to chain them in consistent sequences (a sequence would be consistent as long as the outputs of the web service at the previous step can be pipelined as inputs of web service at the next step).

8. Acknowledgments

The IP project TERREGOV is co-funded by the European Commission[7] under the IST Programme, eGovernment unit.

9. References

[1] Mandell, D.J. and McIlraith, S.A., Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation, in Proc. of the International Semantic Web Conference (ISWC) (2003), pp. 227-241.

[2] Mindswap Java OWL/S API http://www.mindswap.org/2004/owl-s/api/

[3] Sirin E., Hendler J., and Parsia B., Semi-automatic Composition of Web Services using Semantic Descriptions, in: Proc. of Web Services: Modeling, Architecture and Infrastructure, Workshop in Conjunction with ICEIS2003, Angers, France, (2003).

[4] Paolucci, M., Kawmura, T., Payne, T., Sycara, K.: Semantic matching of web services capabilities. In: Proc. of the First International Semantic Web Conference, Sardinia, Italy (2002).

[5] Sycara, K., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems, 5, (2002), pp. 173-203.

[6] Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A semantic web approach to service description for matchmaking of services. In: Proc. of the Intl. Semantic Web Working Symposium (SWWS), Stanford, CA, USA (2001).

[7] Bakry, S.H., Development of e-government: a STOPE view, International journal of Network Management, 14, (2004), pp. 339-350.

[8] Tarabanis K., Peristeras V., Knowledge Management requirements and Models for Pan-European Public Administration Service Delivery, KmGov conference, (2003).

[9] Peristeras, V. and Tarabanis, K., Advancing the Government Enterprise Architecture -GEA: The Service Execution Object Model. in Database and Expert Systems Applications, DEXA, Zaragoza, Spain, (2004).

[10] Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K., Importing the Semantic Web in UDDI. in Web Services, E-Business and Semantic Web Workshop, CAiSE 2002, (Toronto, Canada, 2002), pp. 225-236.

[7] The content of this paper is the sole responsibility of the authors and in no way represents the views of the European Commission or its services.