eGovernment Application Interoperability

Claude Moulin[1], Marco Sbodio[2] , Jean-Paul Barthès[1]
1. Compiègne University of Technology, France – UMR CNRS 6599, Heudiasyc
Tel: +33 3 4423 4485 - Fax: +33 3 4423 4477
{claude.moulin, jean-paul.barthes}@utc.fr
2. Hewlett Packard, Italy Innovation Center
marco.sbodio@hp.com
Tel: +39 011 7177914 - Fax: +39 011 755254

Abstract: This paper only considers the elements which characterizes the level of application interoperability taken in account in the TerreGov project. We present the main requirements for a real interoperability of applications. As it is demonstrated by an actual scenario, the key point of application interoperability is the dynamic discovery of services. This relies on semantic descriptions of services and semantic engines exploiting this information for discovering services. The glue of these elements is, in backend, a domain ontology rich enough for allowing adequate service descriptions. Then, we describe an experimentation, which illustrates a solution of the use case presented in the scenario. We conclude showing how our experimentation could be enlarged efficiently.

1 Introduction

Considering the rapid introduction of new technologies in government services with the trend towards eGovernment, the strategic 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.

With a diversity of in-house information systems (information sources, databases, legacy applications) and the increasing availability of 3rd Party Web 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 effort in terms of "eGovernment process support" facilities for both design and enactment.

In a growing pool of Web Services, technical interoperability is not sufficient and mechanisms should be set up for ensuring that semantics becomes the glue between applications. 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 and methods for standardizing basic e-government services development at the national and international levels [18, 19].

In the TerreGov project, user needs are analyzed by the study of use cases from four pilot regions. The project addresses three levels of interoperability: application interoperability, semantics interoperability, and cultural interoperability. In this paper, we only focus on the first level. We present a scenario coming from the Italian pilot which introduces the requirements for a real interoperability of applications. From this, we describe the elements of our solution to this problem, general enough for becoming the pillars of a concrete architecture.

2 Scenario

We describe here a real but simplified use case, which is part of the Italian Pilot of TERREGOV. This pilot focuses on services that support socio-economic assistance processes for citizens of “Regione Veneto”. The description illustrates the necessity of a dynamic discovery of services and the corresponding pattern we will describe in the next sections.

The typical use case involves a citizen that asks for socio-economic assistance. A specific process must be enacted in order to decide whether the citizen is eligible or not for this kind of assistance. The first step of the process is the collection of relevant personal, medical and economical data relative to the citizen; the collection of information is called the citizen's profile. The local municipality uses a web service for obtaining the personal information (first name, last name, residence, etc.).

As input to the service, the person must give a personal identifier. in Italy, it is the “fiscal code”. A local health care administration uses a web service for obtaining relevant medical data of a patient. The input of this service is the patient's identifier, i.e. the “health card number”.

This case shows the use of both municipality and health care services. Unfortunately the two web services require different inputs: the “fiscal code” and “health card number”. Let's consider the composite process resolving this case. A predefined process would therefore require two inputs separately. In actual cases the list of intermediate services and their requirements could be longer. A better solution would use a process establishing the minimal list of inputs, the citizen is asked for; the system searching for other services which could return the other needed data.

In this case, the minimal list of inputs is reduce to the fiscal code because, it exists a service which can produce an health care number given a fiscal code. The citizen is only asked for this usual information. The existence of the third service is completely transparent in the process. We call this service, the mediator service and the mediator pattern, the process of searching for services acting as a bridge between two other services. In actual processes, more than one mediator services could be useful.

3 Semantic operability requirements

A major goal of an advanced eGovernment platform is to enable accredited actors (representing government services) to enrich the set of Web Services available by publishing new eProcedures as custom-made Web Services. These services could embed access to existing legacy systems, invoke other Web Services and should be packaged in such a way they can themselves be invoked by other Web services. The key point is that such platforms allow the dynamic discovery of services. We describe the main features of these platforms for service discovery.

3.1 UDDI semantic extensions

Common web services descriptions are currently published in UDDI[1] registries, but containing insufficient data for our purpose. Thus, it is necessary to build new registries, called UDDI semantic extensions, containing enough semantic information for service discovery. They allow the publication of semantic service descriptions, and some service discovery mechanisms based on service features. Several efforts of the W3C [1] in the domain of web services [2] tend to propose standards for formally describe web services capabilities: OWL/S [6], formerly DAML/S, and WSMO [7] (see [16] for WSMO based web services). We have decided to adopt OWL/S, even if this standard is still evolving, because some technical resources as Java libraries are already available and will be updated.

OWL/S is a language based on RDF/S[2] used for describing services. In OWL/S a service presents a profile, is described by a model and supports a grounding. The profile is mainly used for service discovery. It 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. We also used the model for service discovery. The grounding describes the implementation of a service. It is linked with WSDL [3] service descriptions. The following excerpt of a semantic description shows the links with the domain ontology. The concepts whose labels are *HealthCard* and *HealthCardId* are defined in this ontology.

```
<process:hasProcess>
   <process:AtomicProcess rdf:ID="Process">
      <process:hasOutput>
         <process:Output rdf:ID="out0">
            <process:parameterType
                   rdf:resource="http://...owl#HealthCard"/>
            <rdfs:label>HealthCard</rdfs:label>
         </process:Output>
      </process:hasOutput>
      <process:hasInput>
         <process:Input rdf:ID="in0">
           <process:parameterType
                   rdf:resource="http://...owl#HealthCardId"/>
           <rdfs:label>HealthCardId</rdfs:label>
         </process:Input>
      </process:hasInput>
   </process:AtomicProcess
</process:hasProcess>
```

3.2 Workflow engine extension

Enabling heterogeneous enterprise application integration, requires the composition of several web services in a workflow for implementing complex tasks. Current technologies address the web service composition issue through web services orchestration; the major current industry initiative is BPEL4WS[3] [4]. BPEL4WS aims at specifying business process behavior based on Web Services; it distinguishes between *executable processes* (modeling actual behavior of a participant in a business interaction) and *abstract process* (describing the technological interfaces of business processes). BPEL4WS focus on representing *static compositions*, where both the flow of the process and its building blocks (i.e. 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

[1] http://www.uddi.org/
[2] W3C RDF Schema: http://www.w3.org/TR/rdf-schema/
[3] Business Process Execution Language for Web Services

qualified patterns (e.g. the mediator pattern) could be implemented at run time. The algorithms they perform, use the ability of discovering available web services. Such eProcedures can combined together several algorithms for computing complex tasks.

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 able. The search engines have to merge elements extracted from service descriptions into the ontology (or fragments of it) for making the right inferences. The eGovernment ontology is also used for indexing documents and resources in knowledge bases. The following figure (build with the protégé editor[4]) shows some concepts of the ontology that the description in section 3.1 refers.

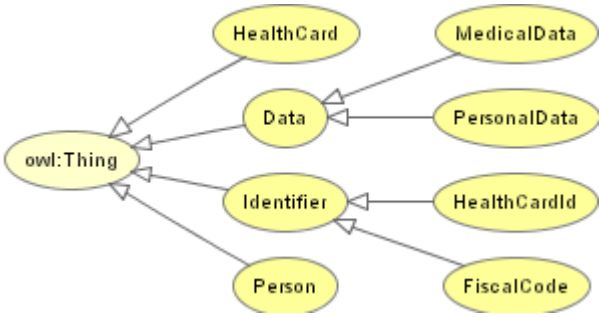

Figure 1: excerpt of the domain ontology

4 Experimentation

This section shows some characteristics of our model and the way they are integrated. The technology used for the implementation relies on Java programming language, Jena APIs [8], and useful OWL/S libraries as Mindswap[5]. The scenario and the experimentation show mainly the use of the inputs et outputs features of a service as they appear in the service description (see section 3.1). Other algorithms mixed matchmaking of concept labels and inferences in ontology.

4.1 System architecture

The architecture of our system contains four main components:

- The “interoperability and coordination layer” is the system entry point. This layer allows programmatic requests of eGovernment services specifying the inputs and outputs of a request. If a single eGovernment service (with specified inputs and outputs) is available, then it is executed by the module, otherwise this module attempts to dynamically compose web services to fulfill the request.

- The “Repository” component offers access to information about known services; it is basically a programmatic interface to a semantic UDDI extension.

- The “Analyzer/Checker” component is in charge of implementing the algorithms that explore available web services and try to dynamically build the web services sequence. It elaborates semantic information and tries to build a sequences using algorithms. It uses two additional toolkits: Jena schemagen [10], which converts OWL [5] vocabularies into Java class file,

[4] http://protege.stanford.edu/
[5] Mindswap Java OWL-S API: http://www.mindswap.org/2004/owl-s/api/

contains static constants associated to the vocabulary terms, and Kazuki [11], a Java API (built on top of Jena) for working with OWL instance data directly from Java code.

- The "Instantiator" component is in charge of taking a complete sequence and to instantiate it as a workflow in a classical BPEL4WS workflow engine. This component translates a web service sequence dynamically composed by the "Analyzer/Checker" component into a BPEL4WS description, and instantiates it within the BPEL4WS engine.

4.2 The mediator pattern

4.2.1 description

A first service S_1 (e. g. a local municipality service) requires an input I_1 (e.g. fiscal code) for returning an output O_1 (e. g. the personal information. A second service S_2 (e.g. a local health care service) requires an input I_2 (e.g. health card number) for returning O_2 (e.g. the medical data). An upper level eProcedure requires both O_1 and O_2. the only input of this eProcedure is I_1.

All inputs/outputs I_1, I_2, O_1, O_2 (such as fiscal code, personal data and health card number) refer to concepts of the ontology. Represented in the OWL formalism, they are described as classes (e.g. FiscalCode, PersonalData, HealthCardId). Our goal is to dynamically build a sequence $I_1 \rightarrow \langle S_{1,2} \rangle \rightarrow \{O_1, O_2\}$. This sequence calls the sequence $I_1 \rightarrow \langle S_{1,} \rangle \rightarrow O_1$ and the sequence $I_2 \rightarrow \langle S_2 \rangle \rightarrow O_2$ and must find a third service S_3 for elaboration the sequence $I_1 \rightarrow \langle S_3 \rangle \rightarrow I_2$.

4.2.2 Algorithm

The goal of this simple composition algorithm is to build a web service sequence with specific input set and output set. The following pseudocode for a recursive back-changing algorithm for finding a sequence of service invocations consuming a set of available inputs and return a set of expected outputs is base on the work done in [12].

```
initialization:
  weHave = {input set};
  weWant = {output set};
steps:
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
}
```

5 Related Work

There are several ongoing efforts and examples of work done in the fields of web services dynamic discovery and composition. Our approach is very similar to the one described in [12]. We try to augment the capabilities of standard BPEL4WS execution engine, with automated reasoning on OWL/S descriptions. Both approaches try and build web services sequences that consume a set of (formally described) inputs, and produce a set of expected (formally described) 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 try and 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 (the "Instantiator" component of the eProcedure module serve as a bridge towards BPEL execution [9]).

Another approach to web service composition is described in [13]. A semi-automatic approach at dynamic composition is described: 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.

There are also several examples of research and work done in the field of matchmaking algorithms for dynamic discovery. We just mentioned some. A prominent example is shown in [14]: the algorithm tests if the request can provide all required inputs of a service and if the offered output satisfies the requestor's 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. [15]. Finally, some interesting insight on the matchmaking problem is given in [15], where the experience report shows a development of a semantic web based matchmaking prototype.

6 Conclusion

Our research may continue in several directions. We intend to enlarge the main algorithms that we use both for matchmaking and composition of web services. Besides semantics of inputs/outputs, the additional preconditions and effects constraints described by OWL/S, can be used for service discovery. Some discovery, for example, could reject services with undesired effects. We also intend to investigate the possibility of reinforcing the matching using subclasses and superclasses as in [14].

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 would not have a specific goal (in terms, for example, of available inputs and expected outputs), but could explore the knowledge-base of 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). We could envisage an indexing of service sequences on patterns as the mediator pattern.

A real interoperability of applications relies mainly on the elaboration of an efficient ontology. We consider that it must be a unique eGovernment ontology, from which service designers have to refer.

7 Acknowledgments

The TerreGov project is an integrated project cofunded by the European Commission[6] under the IST (Information Society Technologies) Programme, eGovernment unit, under the reference IST-2002-507749. The TerreGov Consortium includes: AIRIAL Conseil (F), Université de Technologie Compiègne (F), GFI Informatica (E), Hewlett Packard Italiana (I), Ratio Consulta (I), Oxford Computer Consultants (UK), Technion – Israel Institute of Technology (IL), HEC School of Management (F), Associazione Impresa Politecnico (I), Business Flow Consulting (F), European Regional Information Society Association (B), Stowarzyszenie Miasta w Internecie (PL), Aquitaine Europe Communication (F), Regione del Veneto (I), R&S Info (I), Epektasis (G).

8 references

1. W3C Semantic Web: http://www.w3.org/2001/sw/
2. W3C Web Services Activity: http://www.w3.org/2002/ws/
3. Christensen E., Curbera F., Meredith G., Weerawarana S. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl
4. BPEL4WS. Business Process Execution Language for Web Services. http://www-128.ibm.com/developerworks/library/ws-bpel/
5. W3C Web Ontology Language (OWL): http://www.w3.org/2004/OWL/
6. OWL-S, DAML Web Service Ontology: http://www.daml.org/services/owl-s/
7. Web Service Modeling Ontology (WSMO): http://www.wsmo.org/
8. Hewlett-Packard Jena framework : http://jena.sourceforge.net/index.html
9. ActiveBPEL engine: http://www.activebpel.org/
10. Hewlett-Packard Jena schemagen: http://jena.sourceforge.net/how-to/schemagen.html
11. Kazuki toolkit: http://projects.semwebcentral.org/projects/kazuki/
12. 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.
13. 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).
14. 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).
15. 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.
16. 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).
17. Domingue J., Cabral L., Hakimpour F., IRS-III: a Platform and Infrastructure for Creating WSMO-based Semantic Web Services, WIW workshop on WSMO implementation, Frankfort, September 29-30th, (2004).
18. Bakry, S.H., Development of e-government: a STOPE view, International journal of Network Management, 14, pp. 339-350, (2004).
19. Tarabanis K., Peristeras V., Knowledge Management requirements and Models for Pan-European Public Administration Service Delivery, KmGov conference, (2003).

[6] 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.