MEDIATE-2005

First International Workshop on

Mediation in Semantic Web Services, December 12, 2005

co-located with the Third International Conference on Service Oriented Computing, Amsterdam, The Netherlands

Proceedings

Sponsored by the European Commission under the DIP project (FP6 - 507483)

Martin Hepp, Axel Polleres, Frank van Harmelen, Michael Genesereth
(Editors)

Proceedings

First International Workshop on Mediation in Semantic Web Services (MEDIATE 2005)

in conjunction with the

3rd International Conference on Service-Oriented Computing (ICSOC 2005)

Amsterdam, The Netherlands, December 12, 2005

Organizing Committee:

Michael Genesereth (Stanford University)

Frank van Harmelen (Vrije Universiteit Amsterdam)

Martin Hepp (DERI, University of Innsbruck)

Axel Polleres (DERI, University of Innsbruck)

Workshop URI:

http://www.deri.at/events/workshops/mediate2005

Sponsored by the European Commission under the DIP project (FP6 - 507483).

Table of Contents

Preface

The usage of computer systems is widely characterized by decentralized design and autonomous evolution. If we look at system components from a global perspective, they are frequently developed and modified without alignment in the design stage. Also, components follow individual paths of evolution during their life-cycles. It can be observed that this is a major cause for interoperability problems, contributing to the brittleness of systems integration efforts. If we want to increase the degree of automation in general, it seems important to provide software components that can help overcome occurring interoperability conflicts and this in an automated fashion. This functionality is known as mediation and the respective components are called mediators.

Mediation can take place on various levels, e.g. on the level of data, ontologies, processes, protocols, or goals. To a great extent, it will depend on the availability of sophisticated, industry-strength mediation support whether the promise of Semantic Web services and dynamic value chains can become a reality.

As a consequence, mediators are a fundamental component of a comprehensive Semantic Web services framework. However, they are not yet fully developed as a research topic in this community. Many theoretical and practical issues of yielding sophisticated, scalable, and reliable mediators are not solved. The workshop aims at bringing together experts from various areas of research in order to advance the theoretical and practical knowledge about the design and implementation of mediators in Semantic Web services.

We received a very broad spectrum of submissions and are confident that the nine papers that we finally selected for publication and presentation will contribute to a better understanding of mediation in the context of Semantic Web services. All papers were reviewed by at least two members of the Program Committee.

The organizers would like to thank all authors for their submissions and the members of the Program Committee for their time in reviewing the papers.

Program Committee

Organizing Committee

Michael Genesereth (Stanford University)
Frank van Harmelen (Vrije Universiteit Amsterdam)
Martin Hepp (DERI, University of Innsbruck)
Axel Polleres (DERI, University of Innsbruck)

Program Committee

Diego Calvanese (Free University of Bozen/Bolzano)
Emilia Cimpian (DERI)
Jos De Bruijn, (DERI)
John Domingue (Open University)
Jérôme Euzenat (INRIA)
Dieter Fensel (DERI)
Fausto Giunchiglia (University of Trento)
Rick Hull (Bell Labs)
Michael Kifer (University at Stony Brook)
Deborah McGuinness (Stanford University)
Enrico Motta (The Open University)
Marco Pistore (University of Trento)
Pavel Shvaiko (University of Trento)
Jianwen Su (UC Santa Barbara)
York Sure (AIFB)
Paolo Traverso (ITC/IRST)
Michael F. Uschold (Boeing)
Ludger van Elst (DFKI)
Holger Wache (Vrije Universiteit Amsterdam)
Gio Wiederhold (Stanford University)

Mediation of Semantic Web Services in IRS-III

Liliana Cabral and John Domingue

Knowledge Media Institute, The Open University, Milton Keynes, UK
{L.S.Cabral, J.B.Domingue}@open.ac.uk

Abstract. Business applications composed of heterogeneous distributed components or Web services need mediation to resolve data and process mismatches at runtime. This paper describes mediation in IRS-III, a framework and platform for developing WSMO-based Semantic Web Services. We present our approach to mediation within Semantic Web Services and highlight the role of WSMO mediator types when solving mismatches at the semantic level between a service requester and a service provider. We describe the components of our mediation framework and how it can handle data, goal and process mediation during the activities of selection, composition and invocation of Semantic Web Services.

1 Introduction

Integrating software applications developed in heterogeneous platforms has a high cost for most businesses today, because it means manually providing mappings for data and message formats exchanged between business processes of partner agencies. The advent of Web Services, as part of a trend in XML-based distributed computing, made the integration of applications on the Web a far easier task. Companies can keep intact their legacy implementation of computing systems and provide services by exposing functionalities through a standard interface description. Thus, applications in diverse areas such as e-commerce and e-government can interoperate through Web services implemented in heterogeneous platforms.

However, integration requires that requesters of Web services agree on the meaning of the messages being exchanged with the providers before they can invoke the Web services. In addition, a service requester has to map his request to the requirements of available services.

Despite the advance in the use of standards for Web Service description (e.g. WSDL) and publishing (e.g. UDDI), the syntactic definitions used in these specifications do not completely describe the capability of a service and cannot be understood by software programs. It requires a human to interpret the meaning of inputs, outputs and applicable constraints as well the context in which services can be used. Semantic Web Services (SWS) combine Semantic Web and Web Service technologies to provide the infrastructure for semantically describing Web services facilitating automatic service discovery, composition, execution and mediation.

This paper describes mediation in IRS-III [6], an infrastructure for developing WSMO-based Semantic Web Services [17]. IRS-III is an operational semantic plat-

form for the representation and execution of knowledge models. We present our top-down approach to mediation within Semantic Web Services and highlight the role of WSMO mediator types when solving mismatches at the semantic level between a service requester and a service provider. We describe the components of our mediation framework and how it can handle data, goal and process mediation during the activities of selection, composition and invocation of Semantic Web Services.

The rest of the paper is structured as follows: section 2 describes mediation issues faced by applications using Semantic Web Services; section 3 gives a brief overview of IRS-III and the Web Service Modeling Ontology – WSMO; section 4 describes the IRS-III mediation approach and in particular our mediation framework; in section 5 we present a case-study on e-government, which uses our approach; in section 6 we discuss related work and in section 7 we present our conclusions.

2 Semantic Mediation Issues

Business applications composed of heterogeneous distributed components or Web Services need mediation to resolve data and process mismatches at runtime. We view mediation in the context of Semantic Web Services and define it as an activity for solving conceptual mismatches during the interaction between a service requester and a service provider. One can model specialized mediators which provide a mediation service or declarative mappings for solving different types of mismatches.

Providing a semantic description for a Web Service allows a broker to use the knowledge available for managing the different levels of mediation needed. In this case the conversation between a client and a provider can be handled by a Semantic Web Service execution environment (broker) which can provide mediation during the activities of discovery, composition and invocation for solving mismatches at the semantic level.

A Semantic Web Service can be associated with one or more domain ontologies for describing its functional and non-functional capabilities. This description is used whenever a service is queried or invoked. Usually, a mapping between elements of the ontology used by the client application (or another service) and the ontology used by the service has to be provided. In particular, a developer might want to represent the connections and transformations between elements representing different aspects of the service, for example, for supporting dataflow of composed services. It might be also necessary to transform inputs during the selection and invocation process.

We present our view on the levels of mediation needed within a Semantic Web Service infrastructure that can be handled by different mediation components as well as a specific approach for modeling mediators which can represent types of mismatches and provide the mappings needed.

Semantic data mediation tackles the problem of alignment between ontologies associated with data resources. This problem alone is one of the main research topics on ontology management and coordination (e.g. [2], [7], [10]) in the Semantic Web, which investigates solutions in terms of automatically or semi-automatically generating declarative mappings between different ontological elements.

Within a Semantic Web Services infrastructure, domain ontologies are associated with the descriptions of the different elements of the service. Mediators between ontologies can carry out mappings when other elements such as Goals and Web Services import ontologies.

When composing services for providing functionality, the connections between services must match. Explicit mediators can be defined for mapping or transforming the output of a source service into the input of a target service.

When an application or another service has to interact with a service during invocation, mismatches can occur for example between the format or the order in which the information is requested and the way in which information is provided. A communication protocol in terms of message exchanges has to be followed by the service requester.

The mediation issues mentioned above can be solved by a mediation framework as part of the Semantic Web Service infrastructure. Different components of the run-time environment have access to the semantic descriptions of the service and are able to solve existing mismatches.

A mediation framework can be supported by a design-time tool. The design-time tool should support users in generating conceptual mappings between ontologies associated with Semantic Web Services. These declarative mappings can be made available through Mediators to the run-time environment, which is able to execute them during the invocation of a Semantic Web Service. Alternatively, the run-time environment can consume a mediation service associated with a Mediator, which can perform generic types of transformations on behalf of the service, for instance concatenations or sorting.

3 IRS-III Overview

IRS-III [6] is an implemented infrastructure which allows the description, publication and execution of Semantic Web Services according to the WSMO conceptual model [17]. The meta-model of WSMO defines four top level elements:

- Ontologies,
- Goals,
- Web Services, and
- Mediators.

Ontologies [8] provide the foundation for semantically describing data in order to achieve semantic interoperability and are used by the three other WSMO components. Goals define the tasks that a service requester expects a Web Service to fulfil. In this sense they express the service requester's intent. Web Service descriptions represent the functional behaviour of an existing deployed Web Service. The description also outlines how Web Services communicate (choreography) and how they are composed (orchestration). Mediators handle data and process interoperability issues that arise when handling heterogeneous systems. One of the main characterizing features of WSMO is that Ontologies, Goals and Web Services are linked by Mediators. In particular, WSMO provides four kinds of mediators:

- **OO-mediators** enable components to import heterogeneous ontologies;

- **WW-mediators** link Web Services to Web Services;
- **WG-mediators** connect Web Services with Goals;
- **GG-mediators** link different Goals.

The incorporation of four classes of mediators in WSMO facilitates the clean separation of different mapping mechanisms. For example, an OO-mediator may specify an ontology mapping between two ontologies whereas a GG-mediator may specify a process transformation between two Goals.

IRS-III provides a powerful execution environment for knowledge models. A WSMO description representing the capability of a deployed service can be instantiated within IRS-III operational framework and used for discovery, composition and invocation. IRS-III is based on a distributed architecture which communicates via SOAP. The server component handles ontology management and the execution of knowledge models for Semantic Web Services. The server also receives SOAP requests (through the API) from client applications for creating and editing WSMO descriptions of Goals, Services and Mediators as well as invocation of Goals. The publisher component allows providers of services to attach WSMO descriptions to their deployed services and provides handlers (proxies) to invoke services in specific platforms (lisp code, java code, web services and web applications).

4 IRS-III Mediation Approach

IRS-III Mediation approach is based on: a set of design principles for Semantic Web Services; a mediation framework incorporating a number of components of the IRS-III architecture; and use of the WSMO Mediator meta-model. The following subsections explain our approach in details.

4.1 Design Principles

Our approach is based on the following design principles:

Use of Ontologies – Semantic descriptions of Web Services are ontological meta-models. Furthermore, ontologies can serve as a container and delimit the scope of instances during the execution of a model for mediation.
Executable Semantic Descriptions – All aspects of a web service needed for mediation including choreography, orchestration and ontology mappings can be interpreted by the IRS-III execution environment (reasoning engine).
IRS-III as a broker – IRS-III mediates between client requests and service providers whenever a Semantic Web Service is invoked. The interaction with the service occurs via the choreography of a single service or via the orchestration of a composed service (multiple choreographies).
Goal-based invocation – Client requests are given by Goal descriptions. IRS-III selects a Web-Service which can achieve the Goal. Mismatches can occur between the Goal description and the Web Service description.

Goal-based decomposition – A Web-Service can decompose its functionality into sub-goals described by the orchestration. There can be mismatches between sub-Goals.
Explicit Semantic Mediator description – IRS-III uses mediators to explicitly connect and provide mapping rules or mediation services between services elements.

4.2 Mediation Framework

The IRS-III mediation framework implements data mediation, goal mediation and process mediation of Semantic Web Services. The main objective is to provide mediation components which solve types of mismatches by reasoning over the given Goal, Web Service and Mediator descriptions.

The following sub-sections will explain in more details the use of WSMO conceptual models by the *data mediator*, *goal mediator* and *process mediator* framework components.

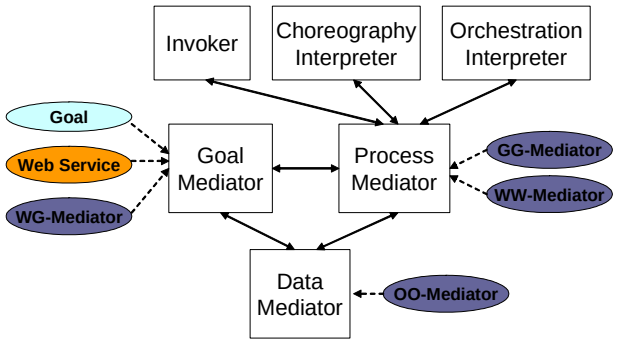


Fig. 1. Mediation framework of IRS-III

Figure 1 illustrates the main architecture components incorporated in the mediation framework of IRS-III. In the steps below we describe the overall sequence of mediation activities taking place during selection, composition and invocation of Semantic Web Services.

1. The Goal Mediator searches for WG-Mediators whose source component matches the current Goal when IRS-III receives an achieve-goal request from a client application. It selects the first targeted Web-Service which matches the requested capabilities (input types, preconditions, assumptions, non-functional properties etc). The types of mismatches that can occur are: a) the input types of a Goal are different from the input types of the target Web Service; and b) Web Services have more inputs than the Goal.
2. The Process Mediator establishes an interaction with a deployed web service (code) by executing its Web Service choreography through the Choreography Interpreter. The Process Mediator performs the lifting and lowering of data provided by the Web Service grounding and is able to create the communication messages corresponding to the choreography communication primitives.

It keeps the state of the communication throughout operation calls executed via the Invoker component.

3. The Process Mediator component also executes the orchestration of a composite Web Service using the Orchestration Interpreter. It keeps the state of the orchestration (control and data flow) between invocations of sub-Goals. The Process Mediator searches for GG-mediators connecting sub-Goals in the orchestration. The types of mismatches that can occur are: a) output types of a sub-goal are different from the input types of the target sub-Goal; b) output values of a sub-goal are in a different order from the inputs of the target sub-Goal; c) the output of a sub-Goal has to be split or concatenated into the inputs of the target sub-goals.
4. The Data Mediator component is used by the Goal Mediator and by the Process Mediator for mapping data across domain ontologies. It executes the mapping rules of OO-mediators used by other WSMO elements.

As a knowledge-based framework, IRS-III models the WSMO specification as a set of related knowledge models for the WSMO top level components of Goals, Web Services and Mediators, which are meta-models in corresponding ontologies. In the following we describe data, goal and process mediation from the perspective of the given Mediator model.

Listing 1. IRS-III mediator meta-model.

```
(def-class mediator (invokable-entity wsmo-entity)
  ((has-source-component :type wsmo-entity)
   (has-target-component :type wsmo-entity)))

(def-class wg-mediator (mediator)
  ((has-source-component :type (or web-service goal wg-mediator))
   (has-target-component :type (or web-service goal wg-mediator))
   (uses-mediator :type oo-mediator)
   (has-mediation-service :type goal)))

(def-class ww-mediator (mediator)
  ((has-source-component :type (or web-service ww-mediator))
   (has-target-component :type (or web-service ww-mediator))
   (uses-mediator :type oo-mediator)
   (has-mediation-service :type goal)))

(def-class gg-mediator (mediator)
  ((has-source-component :type (or goal gg-mediator))
   (has-target-component :type (or goal gg-mediator))
   (uses-mediator :type oo-mediator)
   (has-mediation-service :type goal)))

(def-class oo-mediator (mediator)
  ((has-source-component :type wsmo-ontology)
   (has-target-component :type wsmo-ontology
   (has-mapping-rules :type mapping-rule)))
```

Listing 1 shows the meta-model specification of a Mediator in IRS-III. The main concept is defined by the class *Mediator* which is subclassed into more specific types of mediators (wg-mediator, ww-mediator, gg-mediator, oo-mediator). Source and target components can be any of the WSMO top level components (class *wsmo-entity*). The mediators differ according to the type of source and target components they can

handle and whether it uses a mediation service or mapping rules. Thus, mediators are bridges which can provide conceptual mappings or input transformations from source components to target components. IRS-III supports the implementation of Mediation Services as Goals as well as the explicit declaration of mapping rules. Since mediation services are implemented as Goals they can simply be invoked resulting in the transformation of the relevant input data. IRS-III's reasoning engine can for example match the inputs of the mediation service with the inputs of the source component; and the output of the mediation service with the input of the target component.

4.2.1 Data Mediation

The Goal, Web Service and other Mediator descriptions associated with a web service can refer to an OO-mediator in order to use ontologies which do not match. IRS-III handles data mediation by executing the mapping rules provided by an OO-Mediator (fig. 2). In IRS-III, the source and target components of an OO-mediator are ontologies.. Furthermore, the source and target can be the home ontologies of associated Goals or Web Services.

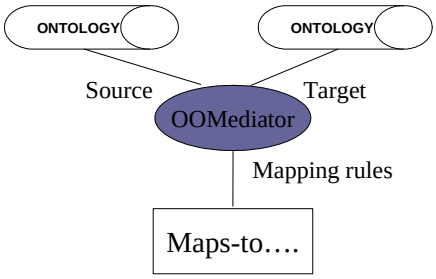


Fig. 2. Mediation between two ontologies

The underlying modeling language OCML [12] has a mechanism for mapping between entities associated with knowledge models. A simple way of dynamically associating elements of a source and a target ontology is through backward chaining rules using the *def-concept-mapping* and *def-relation-mapping* constructs.

Listing 2 shows how a mapping rule can be used to link the slots of classes in two different ontologies. More specifically, the definitions below link the *has-citizen-name* slot of class *citizen* in the source ontology to the *has-client-name* slot of class *client* in the target ontology. This example is taken from the e-government scenario and reflects the fact that a service requester can refer to an entity as citizen and a service provider can refer to it as client.

The *def-concept-mapping* construct in Listing 2 associates each instance of the *citizen* class to a newly created instance of the *client* class and link them by generating instances of the relation *maps-to* internally. The *def-relation-mapping* construct uses the generated *maps-to* relation within a rule which asserts the value of the mapped citizen name to the value of the client name.

IRS-III executes the mapping rules within a temporary ontology created by merging the source and target ontologies. The temporary ontology is then discarded after the Web Service invocation.

Listing 2. Example of a mapping rule.

```
(def-concept-mapping citizen client)

(def-relation-mapping citizen-client-name-mapping
   ((has-client-name ?client ?value)
    if
    (maps-to ?client ?citizen)
    (has-citizen-name ?citizen ?value)))
```

WG-mediators, GG-mediators and WW-mediators have a data mediation capacity for transforming inputs between source and target components by using mediation services and have different roles within the process mediation as explained in the following sections.

4.2.2 Goal Mediation

The goal mediator component of IRS-III handles mismatches that occur during the process of selection of Web Services for solving a Goal. The IRS-III approach assumes that application developers can create or search for Goal and Web Service descriptions available in a library.

A WG-mediator is created for connecting every Web Service to a Goal it can achieve. The WG-mediator model also specifies a mediation service which can transform the inputs of a Goal into the format of the inputs used by a Web Service. When a user requests a goal to be achieved, the mediation service associated with the mediator of each linked web service is executed so that the matchmaking during selection can be carried out over the mediated data.

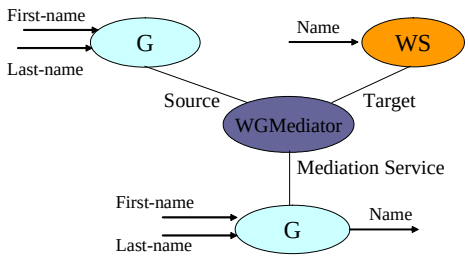


Fig. 3a. Mediation between a Goal and a Web Service. Two inputs of Goal are transformed into one input of the Web Service

Figure 3a shows a graphical illustration of the mediation taking place between a Goal and a Web Service via a WG-mediator. In this example, the Goal requested by the application takes two inputs (first and last names), which are transformed by the mediation service into one input (name) used by the target Web Service.

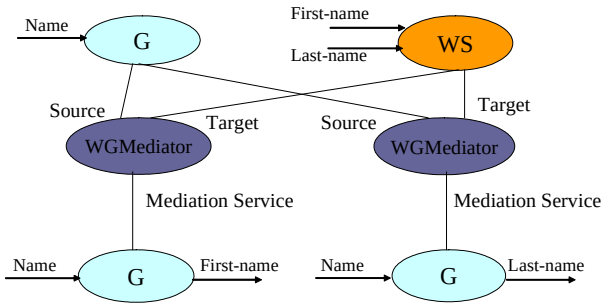


Fig. 3b. Mediation between a Goal and Web Service. One input of the Goal is transformed in 2 inputs of the Web Service

Since a mediation service can return only one output, IRS-III use a set of mediators between the goal (source) and the web service (target) in order to provide the required number of inputs to the target component as shown in figure 3b. In this example, each mediation service transforms (e.g. splits) the goal input (name) in one of the required inputs of the target component (first-name, last-name). The IRS-III engine can match the inputs and outputs for providing values as required.

4.2.3 Process Mediation

The Process mediation component of IRS-III handles mismatches that occur during the invocation or composition of a Web Service. IRS-III either executes the choreography (interaction protocol) of a single Web Service or the orchestration of a composed Web Service, using the values provided by the Goal inputs. Moreover, the Process mediator has to execute the choreography of each single Web Service in the Orchestration.

In IRS-III the choreography of a Web Service describes how to interact with a deployed service (client choreography). A set of rules (guarded transitions) in the choreography are used to specify the flow of operations required for realizing the specific functionality of the Web Service. The Process Mediator uses the Web Service grounding for creating the communication messages based on the operations declared at the conceptual level.

A choreography is provided to interact with a single Web Service. By interpreting the choreography and grounding, the Process mediator component can send messages to the service in the right order and format on behalf of the client. When a Web Service is composite an orchestration has to be provided instead. Nevertheless, its input values have to be passed to the orchestration and the result of the orchestration has to be passed back to the Web Service. The orchestration follows the decomposition of Goals into sub-Goals and uses GG-mediators for connecting sub-goals and mediating the order and types of inputs between them.

We illustrate in the following the role of a GG-mediator during orchestration (fig. 4). The provider of a Web service describes the orchestration through control-flow mechanisms, for instance: (sequence G1 G2 M1). The *Sequence* control command

executes the given sub-goals (G1 and G2) in sequence. Figure 4 shows the graphical representation of the GG-mediator connecting G1 to G2. This mediator supports the data flow between the sub-goals and the necessary transformations. The source goal (G1) produces one output (E1), which is transformed by the mediation service in one input (E2) used by the target Goal (G2). During the execution of the orchestration the input values (SC, TC, A) received by the current invoked Goal are sent onto the sub-goals through matching, then the associated GG-mediator (M1) are used to connect and forward results between sub-goals providing the necessary transformations through the mediation service.

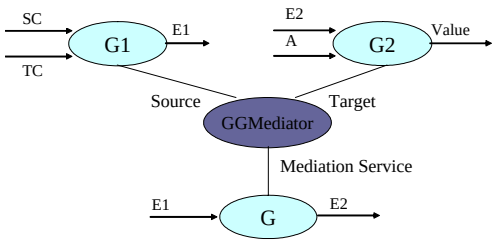


Fig. 4. Mediation between two sub-goals. The input of goal1 is transformed in one input of goal2

WW-mediators can be used in a similar way to GG-mediators by the Process Mediator. In this case, the WW-mediator can provide mappings between the input values of the current Web Service and the Web Services in the orchestration.

5 A case study in E-government

The main requirement for applications in E-government relates to the interoperability of data and processes between services provided by government agencies. Thus, the e-government domain is a natural application area for mediation of Semantic Web Services. The ability to aggregate and re-use all the information resources relevant to a given problem and further to make this available as a basis for transparent interaction with community partner organisations and individual citizens is very restricted. Furthermore, the goals of citizens using e-government services and of government providers of services are often not conceptually aligned, contributing to misunderstanding, low take up and poor relations between citizens and their governments.

We have created a prototype for the case study on e-government within the DIP project (http://dip.semanticweb.org) for illustrating Semantic Web Services. We will comment on the requirements and use of mediation within the scenario implemented.

5.1 Application Scenario and requirements

We illustrate the implementation of our e-government use case through an application scenario. The prototype is a portal for Essex County Council in UK, where two governmental agencies were involved:

- **Community Care (Social Services) in Essex County Council** - they typically have a coordinating role in relation to a range of services from a number of providers and special responsibility for key services such as support for elderly and disabled people (day centers, transportation). It uses the SWIFT database as its main records management tool.
- **The Housing Department of Chelmsford District Council** - handles housing services and uses the ELMS database

In this scenario, a case worker of the Community Care department helps a citizen to report his/her change of circumstance (e.g. address) to different agencies involved in the process. In this way, the citizen only has to inform the council once about his/her change, and the government agency automatically notifies all the agencies involved. An example might be when a disabled mother moves into her daughter's home. The case worker opens a case for a citizen who is eligible to receive services and benefits – health, housing, etc. Multiple service providing agencies need to be informed and interact.

From the scenario above we have gathered the following mediation requirements and solutions:

- Data mediation - Agencies have their own databases and hence different data formats for the same concept (e.g. Address). Different data formats can be lifted to the same concept in a domain ontology. At a semantic level, different concepts can be mapped through mediators.
- Goal mediation – Agencies achieve goals in different ways (e.g assess equipment for a citizen). Here we can define one Goal that can be satisfied in different ways by applicable Web Services developed within different agencies. Multiple Web Services can be linked to the same Goal via Mediators.
- Process mediation - Agencies processes behave in different ways according to their own set of operational procedures, requirements and constraints. Each Web Service presents a choreography describing how a client talks to the deployed service. Furthermore, sub-Goals can be composed together for providing the functionality of one Web Service through the Web Service orchestration.

5.2 Prototype Development

In our approach for developing applications using Semantic Web Services with IRS-III we devise a customer team for creating Goal descriptions according to user requests and a development team for creating Web Service descriptions for the available deployed web services. The application developer is then able to create Mediator descriptions which connect domain ontologies, Goals and Web Services and provide

mediation services or mapping rules for solving mismatches between ontological elements.

The main characteristic of the prototype architecture is that it is service oriented. The portal application created over this architecture implements Semantic Web Services that have integration purpose across the various agencies involved in our e-government scenario. The main services provided through the portal are the ones which can be shared between agencies or used to send/get information to/from more than one agency or even third parties (e.g. list of equipments provided).

The structure of ontologies in fig. 5 represents the libraries of WSMO models for the e-government application. The light-colored rectangles on the top-half of the diagram represent domain ontologies. The dark-colored rectangles on the bottom-half of the diagram represent ontologies with Goal, Web Service and Mediator descriptions available from Community Care (boxes on the left) and Housing Department (boxes on the right. The libraries above provide a clear separation of user goals and web service contexts within agencies and the use of mediators for linking them. Agencies also share the WSMO upper model and the e-government upper level ontology.

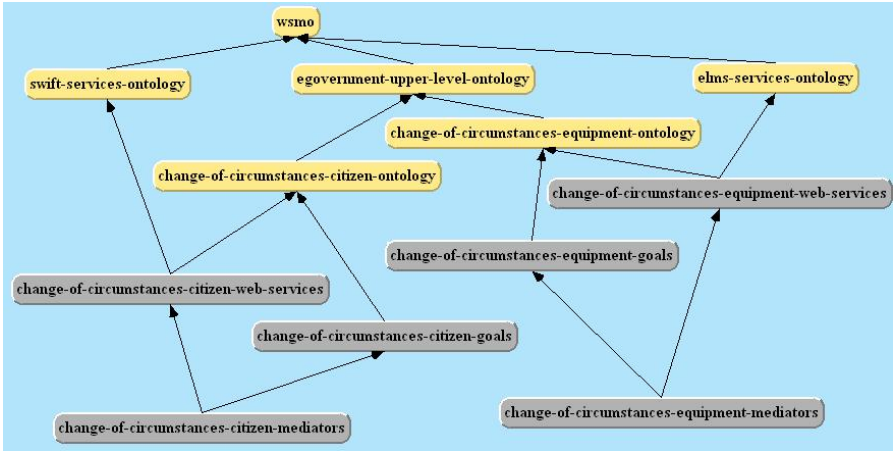


Fig. 5. Structure of ontologies for the e-government application

For illustration purposes (figure 6) we describe in the following the structure of WSMO descriptions associated with one of the goals (*Assess-Equipment-Goal*) defined in our prototype. This Goal describes a request for a service that can assess housing equipments (items) for a citizen who has registered for benefits within Essex County Council. Published services must find all items that suit the citizen's situation (mobility-impairment, visual-impairment, hearing-impairment, baby-care etc) and weight, and the budget of the council's case worker. Restrictions on the way the service can solve the goal are given by pre-conditions and post-conditions.

The Housing Department provides a composed web service (*Housing-Dept-Assess-Items-WS)* for solving the goal described above. The composition is defined by the orchestration in the format: *(Sequence G1 G2 G3 M1 M2),* where G1, G2 and G3 represent sub-goals and M1 and M2 the GG-mediators connecting them. In our example

the sub-goals are: *Find-Items-by-Purpose-and-Weight-Goal*; *Assess-Budget-Goal* and *Select-Suitable-Items-Goal*. A third party company provides a single web service (*Third-Party-Assess-Items-WS)* for solving the above goal, which is described with concepts from the domain ontology *Third-Party-Items-Ontology*.

The mediator descriptions used in this example (fig. 6) are explained in the following. Note that all links coming from mediators connect source to target components (labels were omitted to avoid cluttering the diagram).

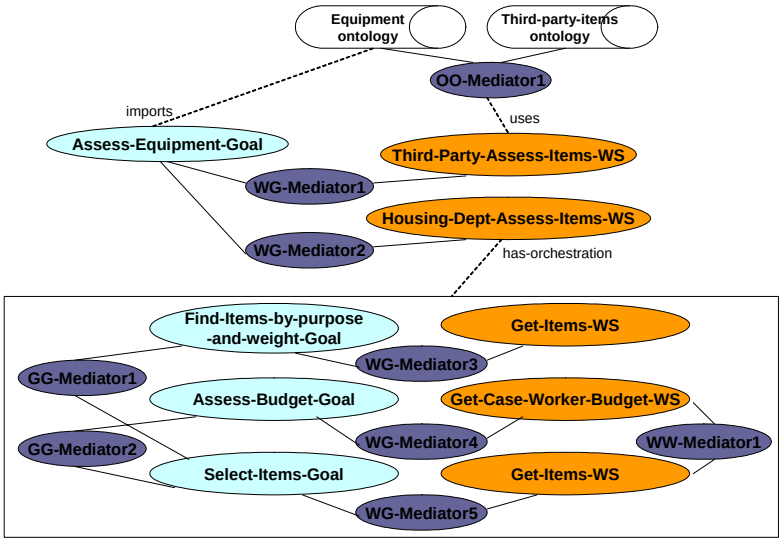


Fig. 6. Sample structure of WSMO descriptions for the e-government prototype

- **WG-Mediator1** – connects *Third-Party-Assess-Items-WS* to E-Gov-Assess-Items-Goal allowing it to be selected for solving the goal. This mediator defines a mediation service for converting the value of input *weight* from pounds (in the goal) to kilos (in the web-service).
- **WG-Mediator2** – connects *Housing-Dept-Assess-Items-WS* to E-Gov-Assess-Items-Goal allowing it to be selected for solving the goal. There is no mediation service and the input types are inherited from the goal.
- **OO-Mediator1** – Defines mapping rules for aligning *Third-Party-Items* domain ontology (used by the Web Service) with Equipment ontology;
- **GG-Mediator1** – Allows the output of *Find-Items-by-Purpose-and-Weight-Goal* to be used as input by *Select-Suitable-Items-Goal*.
- **GG-Mediator2** – Allows the output of *Assess-Budget-Goal* to be used as input by *Select-Suitable-Items-Goal*. It uses a mediation service to map the input type *Budget* (in the source sub-goal) to input type *Cost* (in the target sub-goal).
- **WG-Mediator3**, **WG-Mediator4** and **WG-Mediator5** – Connect corresponding Web Services to Sub-Goals in the orchestration. The Housing

Department has specific services for solving those goals and no mappings are required.

- **WW-Mediator1** – Connects the two web-services for sharing concepts.

6 Discussion and Related Work

Mediation approaches for integration of heterogeneous components or data resources can range from techniques for mapping several resources to a canonical ontology to mediation components which handle transformations between different protocols. In this paper we focus on mediation provided by Semantic Web Services and in particular, on the conceptual modeling and integration aspects of mediation rather than on mapping algorithms. Thus, we have investigated how a mediation framework can handle semantic descriptions for solving mismatches during selection, composition and invocation of services.

Recent work within the knowledge representation research community (e.g. [9] [16] [10]) has contributed to the formalization of ontological mappings, which can be used by SWS mediators, specifically OO-mediators. Reuse in ontology mappings is also discussed in [7], where types of mappings between ontologies, called alignments, are viewed as objects which can be created and used by different users. However, in that case the API proposed is more likely to be used within a design tool which would generate the mappings declaration.

OWL-S [15] does not model the mediator concept. Yet, mediation plays a key role in the approach [14]. The OWL-S approach considers that mediation is handled during discovery and decomposition by architectural components and that a mediation service is treated just as another web service. This assumption makes mediation very implementation dependent and not visible to the user.

WSMX [18], an execution environment for WSMO, contains a data mediation component [11] and a process mediation component [4]. The main difference is that WSMX is not a knowledge-based execution environment. Thus, the Mediator conceptual model is not used by the Mediation components. The WSMX data mediation component can execute mapping rules generated at design time by a mapping tool, but do not implement mediation services as Goals. The WSMX process mediation component works on predefined types of mismatches between two choreography instances while IRS-III Process Mediator interprets the choreography provided by the Web Service and handles mismatches during orchestration. IRS-III follows the UPML design principles [13] for Goal decomposition within the orchestration, whereby a Web Service can decompose its functionality into sub-Goals.

The work on virtual providers [1], which formalizes WSMX process mediation, follows the same approach for mediating between two business process interfaces. The approach in [5] describes the process mediator as the middleware for handling composition.

7 Summary and Conclusions

In this paper we have identified mediation issues within Semantic Web Services and provided a solution in terms of a mediation framework. IRS-III as a broker follows a top-down approach in which its framework components use semantic descriptions to support the mediation process. IRS-III enables easy integration of heterogeneous services by reasoning over the WSMO conceptual model of domain ontologies, Goals, Web Services and Mediators. In particular, modeling mediators provides design time and runtime support for the automation of data, goal and process mediation of Semantic Web Services.

Explicitly modeling types of Mediators in WSMO has many advantages since IRS-III provides the mechanisms for reasoning and behaving according to the knowledge models. First, representing a mediator as a meta-model enables easy inspection by developers. Second, as independent components, mediators can be indexed and reused through a library. Third, mediation services associated with mediators are defined as goals which can be achieved by an implemented web service. Finally, ontology mappings can be provided by the relation mapping mechanism of the underlying reasoning engine. Developers can also inspect mapping rules and test mediation services by searching a library or repository of mediators.

IRS-III exploits the semantics of the WSMO mediator concept during the selection, composition and invocation of Semantic Web Services. For example, the IRS-III broker can match the inputs of the source component with the inputs of the mediation service when deciding which values will be mediated. In a similar fashion, mediators are used for dataflow between sub-goals during orchestration.

We have presented a case study on e-government, which offers a motivating scenario for the use of mediators. Further work is under development regarding the development of a design tool for generating mapping rules for mediators.

Acknowledgements

This work is supported by DIP (Data, Information and Process Integration with Semantic Web Services) (EU FP6 - 507483) and AKT (Advanced Knowledge Technologies) (UK EPSRC GR/N15764/01) projects. We also thank the contribution of Mary Rowlatt, Robert Davies and Leticia Gutierrez from Essex County Council in UK to the case study.

References

1. Altenhofen, M., Borger, E., Lemcke, J. (2005). A High-level specification for mediators. Accepted at 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management.
2. Bouquet, P., Serafini, L. and Zanobini, S. (2004). Coordinating Semantic Peers. In proceedings of the 11th Int. Conf. AIMSA 2004, Varna, Bulgaria. LNAI 3192

3. Cabral, L., Domingue, J., Motta, E., Payne, T. and Hakimpour, F. (2004). Approaches to Semantic Web Services: An Overview and Comparisons. In proceedings of the First European Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece. LNCS 3053, pp. 225-239
4. Cimpian, E.; Mocan, A. (2005): *WSMX Process Mediation Based on Choreographies*. 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management, 2005.
5. Constantinescu, I., Faltings, B., Binder, W. (2004). Large scale, type-compatible service composition. In IEEE International Conference on Web Services (ICWS 2004, San Diego, CA, USA.
6. Domingue, L. Cabral, F. Hakimpour,D. Sell and E. Motta (2004). IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. Proceedings of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany. CEUR Workshop Proceedings, Vol. 113
7. Euzenat, J. (2004). An API for Ontology Alignment. In proceedings of ISWC 2004, Third International Semantic Web Conference, Hiroshima, Japan. LNCS 3298.
8. Gruber, T. R. (1993) A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2)
9. Heller, B. and Herre, H. (2004). The Theory of Top-level Ontological Mappings and its Application to Clinical Trial Protocols. In proceedings of the 14th Int. Conf. EKAW 2004, Whittlebury Hall, UK. LNAI 3257
10. Kotis, K., Vouros, G. and Stergiou, K. (2004). Capturing Semantics Towards Automatic Coordination of Domain Ontologies. In proceedings of the 11th Int. Conf. AIMSA 2004, Varna, Bulgaria. LNAI 3192
11. Mocan, A.; Cimpian, E. (2005): *D13.3v0.2 WSMX Data Mediation*, WSMX Working Draft 2005, available at: http://www.wsmo.org/TR/d13/d13.3/v0.2/
12. Motta, E. (2004) Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving. IOSPress.
13. Omelayenko, B., Crubezy, M., Fensel, D., Benjamins, R., Wielinga, B., Motta, E., Musen, M., Ding, Y. (2003). UPML: The language and Tool Support for Makiing the Semantic Web Alive. In: Fensel, D. et al. (eds.): Spinning the Semantic Web: Bringing the WWW to its Full Potential. MIT Press 141–170
14. Paolucci. M, Srinivasan, N. and Sycara, K (2004). Expressing WSMO Mediators in OWL-S. In proceedings of the Workshop on Semantic Web Services: Preparing to meet the World of Business Applications, in conjunction with ISWC 2004, Hiroshima, Japan. http://CEUR-WS.org/Vol-119/
15. OWL-S Coalition (2003). OWL-S 1.0 Release. http://www.daml.org/services/owl-s/1.0/.
16. Van Elst, L. and Kiesel, M. (2004). Generating and Integrating Evidence for ontology mappings. In proceedings of the 14th Int. Conf. EKAW 2004, Whittlebury Hall, UK. LNAI 3257
17. WSMO Working Group (2004). D2v1.0. Web Service Modeling Ontology (WSMO). WSMO Working Draft. http://www.wsmo.org/2004/d2/v1.0/
18. WSMO Working Group (2004). D13.3v0.1 Mediation in WSMX. WSMX Working Draft. http://www.wsmo.org/2004/d13/d13.3/v0.1/

Process Mediation in an Extended Roman Model

Gösta Grahne and Victoria Kiricenko

Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, Canada H3G 1M8
grahne,kiricen@cs.concordia.ca

1 Introduction

A *mediator* is a software module that provides sharing of services and and agglomeration of resources into complex services. Mediators will play a pivotal role in successful infrastructures for Semantic Web Services. *Process mediation* in Web Services involves issues of process compatibility and composability. Evolving standards of web services, such as the web service Execution Environment (WMSX [21]) focus mostly on compatibility issues in a B2B environment, whereas the problem of dynamic composition of web services is still maturing in the research community. In this paper we contribute to the composition aspect of process mediation.

Web service process specification has reached well accepted standards, such as the Business Process Execution Language (BPEL4WS [9]). Moreover, it has recently been shown that *process algebra* provides a useful interpretation of BPEL specifications. Works such as [12, 19] provide the mapping from BPEL to process algebra, and show how the algebra can be used to describe Web Services during design stage. These works also demonstrate how process algebraic descriptions can be extracted from existing Web Services for reverse engineering purposes. This enables the use of numerous verification tools that are available for process algebra (e.g. LOTOS) in web service development.

When it comes to the question of composability of web services some of the most fruitful results have been achieved within the *Roman* model [3, 2, 7, 11, 14, 5], an evolving framework that takes a highly computational approach to Web Service composition. In this paper we extend the Roman model by incorporating features from process algebra that will allow a fuller use of important functionalities of Web Services, such as parallelism, nondeterminism, and task decomposition.

Our contributions are as follows:

1. We develop a rigorous extension to the Roman model, and give a formal semantics based on the process algebraic notion of simulation. Our extension not only allows for fuller coverage of the standard languages used for description and execution of web wervices in practice, but also unifies the approaches of modeling and mediating web services. Moreover, our extension

allows for formal verification of mediated web services though the use of the numerous tools available for process algebra.
2. We give an algorithm for mediating a requested Web Service from a resource pool of more basic services.

Process mediation is a complex task, and our work certainly does not address all the problems and all the mediation scenarios that may appear in the Web Service context. In the concluding section we outline some extensions that we are currently investigating.

2 Behavioral Models of Web Services

Web Services are distributed and independent pieces of software working together to achieve given tasks. The Business Process Execution Language (BPEL [9]) is a notation for describing executable business process behaviors. Such behavioral signatures provide the foundation for composing Web Services.

Behavioral signatures are often formalized as state based (infinite) labeled transition systems, i.e. (infinite) labeled graphs. Vertices represent processes and labeled edges represent activity. The node that an edge is incident upon represents the state that the process has evolved to, after performing the action of the label. This approach is also adopted in the Roman model [3, 2, 7, 11, 14, 5], with the restriction that the transition graphs are tree structured.

Process algebra is a mathematical framework for reasoning about behavioral signatures. Numerous processes algebras have been proposed and studied in the literature. Basic ones include CCS, CSP, ACP, extensions are π-calculus and timed CSP [10]. LOTOS [16] is for one of the most expressive process algebras, and it comes with an industrial strength suite of tools for specification and verification. Recently, Ferrara [12] has given a two-way mapping that allows an automated translation between BPEL and LOTOS, thus making the LOTOS suite available for Web Service development.

The "urelements" in all process algebras is a finite set of *atomic processes*. Although syntactically different, all process algebras share a core set of basic constructs for process evolution, namely, sequential composition, non-deterministic choice, parallel composition, communicating (synchronized) composition, and recursion [10]. In our framework we use these core constructs, We build on the Roman model, which provides a basic framework for Web Service specification and verification. The Roman model is a robust foundation, and it is currently being extended with e.g. asynchronous message exchange [6], value passing [5], temporal constraints [14], Presburger constraints, discrete time, and non-regular processes [11].

In the next section we give our extension of the Roman model. In order to have a uniform framework, we build our extension from "first principles." Due to space limitations, some proofs are omitted. They can be found in a longer version of the paper at the authors' websites.

3 An Enhanced Roman Model

Let $\mathbb{N} = \{1, 2, 3, \ldots, \}$. By $\mathbb{N}^*$ we denote the set of all *finite strings* over $\mathbb{N}$. The empty string over $\mathbb{N}^*$ is denoted by ϵ. A *tree domain* D is a non-empty subset of $\mathbb{N}^*$ that is closed under the prefix relation. (i.e. if $u.w \in D$ then $u \in D$).

Now let Q be a *state space*, $F \subseteq Q$ a designated set of *final states*, and Σ a finite set of *actions*, Then an *execution tree* T is a function $T : D \to Q \times (\Sigma \cup \lambda)$, such that $T(\epsilon) = (q, \lambda)$, for some $q \in Q$. The *root* of T is ϵ. A *leaf node* of T is an element $w \in dom(T)$, such that for all $i \in \mathbb{N}$, $wi \notin dom(T)$.

The intuition is that the root is the start state of the service, and if (p, b) is a child of (q, a), then the service was in state p, and went into state q after having executed action a. It came into state p after having executed action b, and at the root it has not yet executed any actions. When the service is satisfying a request, it moves from the root state to some final state, while executing the actions along the path.

We shall in the sequel denote D by $dom(T)$. Figure 1 shows an example of a (finite) execution tree. For this T, we have $dom(T) = \{\epsilon, 1, 2, 11, 12, 13, 21, 121\}$, shown to the left, $F = \{p_0, p_1, p_3\}, Q = \{p_0, p_1, p_2, p_3, \}$, and $\Sigma = \{a, b, c\}$. To the right we show the function T graphically, e.g. $T(12) = (p_2, a)$. The leaves are $\{11, 121, 13, 21\}$. Thus this service T can satisfy the requests $\{b, bb, bab, ba, ca\}$.

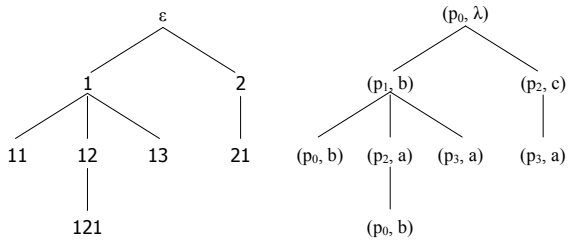


Fig. 1. A tree domain and a tree

To compare the capabilities of Web Services, we need a notion of equivalence, that tells us when two Web Services can perform the same tasks. Research in process algebra has revealed a wide spectrum of equivalences [10]. At one end is *trace* equivalence: two processes are considered equivalent if they can perform the same sequence of tasks. Trace equivalence represents a black box view of the processes. At the other end of the spectrum is *simulation* equivalence. This equivalence comes in various flavors, but the basic idea represents a game theoretic view: two processes are equivalent if they can match each others task sequences action by action.

Previously, in works on the Roman model [3, 2, 11, 14], a notion corresponding to trace inclusion has been used implicitly or explicitly. However, research in formal specification (see e.g. [10]) has shown that trace equivalence (two-way trace inclusion) is not always adequate. Consider the following example.

Example 1. An on-line computer store allows its users to search its product catalogs, to make a selection and then buy the selected item, or to save the results of the search for later use. Both execution trees in Figure 2 could be the web service for (this aspect of) the online computer store, as they both contain the same sets of traces ($\{search.buy, search.save\}$). However, in the execution tree to the right, lets call it T', a user has a choice of executing either *buy* or *save* after performing the *search* action, while in the execution tree to the left, called T, the web service makes a nondeterministic choice when a user initiates the *search* action. Note that one of the nondeterministic choices in T allows the user to subsequently initiate only the *buy* action and the other choice allows him to initiate only the *save* action, whereas in T' both the *buy* and the *save* action are enabled after the *search* action. It is clear that a notion of inclusion finer then trace inclusion is needed to distinguish between these two execution trees. □

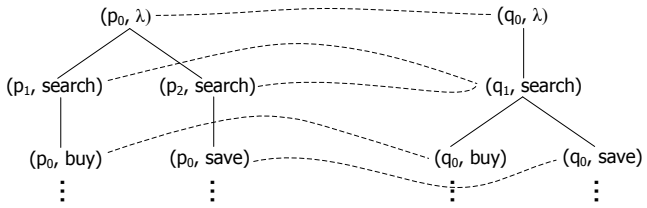


Fig. 2. Forward simulation $T \preceq T'$

We shall therefore use a notion derived from bisimulation, called *forward simulation* [17]. While bisimulation and its variants have received intense attention in the research community, forward simulation is much less investigated.

To define forward simulation formally, let Q and Q' be state spaces, with final states F and F', respectively, and let Σ be an alphabet. Let $T : dom(T) \rightarrow Q \times \Sigma$ and $T' : dom(T') \rightarrow Q' \times \Sigma$ be execution trees. We say that T can be *forward simulated* by T', denoted $T \preceq T'$ if there exists a *simulation relation* $\mathscr{R} \subseteq dom(T) \times dom(T')$, satisfying:

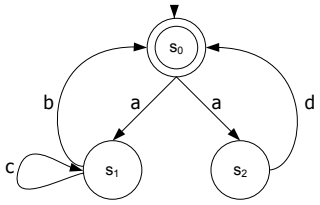
1. $(\epsilon, \epsilon) \in \mathscr{R}$
2. If $(w, w') \in \mathscr{R}$, for some $w \in dom(T)$, $w' \in dom(T')$, and $T(wi) = (s, a)$, for some $i \in \mathbb{N}$, then there exists $j \in \mathbb{N}$ and $s' \in Q'$ such that $(wi, w'j) \in \mathscr{R}$ and $T'(w'j) = (s', a)$.
3. For all $(w, w') \in \mathscr{R}$, where $T(w) = (p, a), T'(w') = (q, b)$, for some $a, b \in \Sigma$, if $p \in F$ then $q \in F'$.

Example 2. Consider again the two execution trees given in Example 1. Forward simulation requires that the simulating web service can simulate the simulee in a lock-step fashion. As we can see from the Example 1, T cannot simulate T'. On the other hand T' can simulate T. For every action that T takes,

T' can take the same action, and has at least the options of T for subsequent actions. In Figure 2 the dotted lines illustrate the simulation relation $\mathscr{R} = \{(\epsilon, \epsilon), (1, 1), (2, 1), (11, 11), (21, 12), \ldots\}$. □

Finite State Machines. So far, we have described web services on an abstract semantic level. In practice, web services need to be finitely specified or implemented. Following the Roman model, we shall represent a web service by a finite state machine.

Let Σ be a finite set of actions, as before. Actions will be denoted by letters $a, b, c \ldots$, and the empty string over Σ is denoted λ. A (nondeterministic) *Finite State Machine* (FSM) is a quintuple $A = (Q, \Sigma, \delta, p_0, F)$ where Q is the finite set of *states* of A, Σ is the finite set of *possible actions*, $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*, $s \in Q$ is the initial state, and F is the set of final states of A (see e.g. [15]).



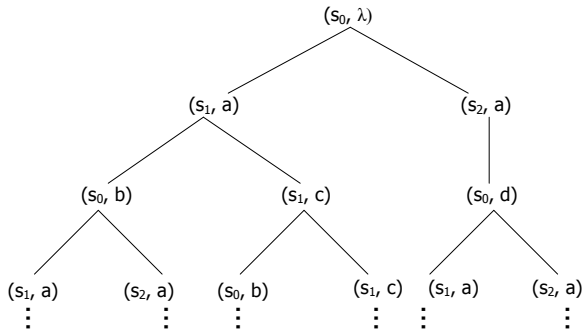


Fig. 3. An FSM an its execution tree

It should be noted that as opposed to the initial Roman model [3], our definition of an FSM allows for nondeterminism. This approach is more realistic as it is computationally closer to the specification languages typically used for description of the web services. Moreover, it better captures the dynamic, not always fully reliable, and partially redundant nature of web services. Nondeterminism has also been adopted in e.g. [11, 14], within the scope of the Roman model.

Given an FSM $A = (\{p_0, p_1, \ldots, p_n\}, \Sigma, \delta, p_0, F)$, its *execution tree* T_A is mapping $T_A : dom(T_A) \rightarrow Q \times \Sigma$, where T_A and $dom(T_A)$ are defined inductively as:

1. $dom(T_A) = \epsilon$ and $T_A(\epsilon) = (p_0, \lambda)$
2. If w is a leaf of $dom(T_A)$, with $T_A(w) = (p_j, a)$, and (p_j, b_1, p_{i_1}), $(p_j, b_2, p_{i_2}), \ldots, (p_j, b_k, p_{i_k}) \in \delta$ are all the transitions emanating from p_j, with $i_1 < i_2 < \ldots < i_k$, then extend $dom(T_A)$ and T_A as follows:
   (a) Add $w1, w2, \ldots, wk$ to $dom(T_A)$.
   (b) Extend T_A by setting $T_A(w1) = (p_{i_1}, b_1), T_A(w2) = (p_{i_2}, b_2), \ldots, T_A(wk) = (p_{i_k}, b_k)$.

Figure 3 shows an FSM and part of its execution tree.

We can now use forward simulation to compare FSM's. We say that a web service specified by an FSM A can be forward simulated by a web service specified by an FSM A', if $T(A) \preceq T(A')$.

Given the fact that execution trees are (typically) infinite, we are interested in a finite characterization of when $T(A) \preceq T(A')$.

Let $A = (Q, \Sigma, \delta, p_0, F)$ and $A' = (Q', \Sigma, \delta', q_0, F')$. We denote (with slight abuse of notation) $A \preceq A'$, if there exists a *simulation relation* $\mathscr{R} \subseteq Q \times Q'$ between the states of A and A', satisfying:

1. $(p_0, q_0) \in \mathscr{R}$
2. If $(p, q) \in \mathscr{R}$, and $(p, a, s) \in \delta$, for some $s \in Q$, then there exists a $t \in Q'$, such that $(q, a, t) \in \delta'$ and $(s, t) \in \mathscr{R}$.
3. For all $(p, q) \in \mathscr{R}$, if $p \in F$ then $q \in F'$.

Example 3. Consider the two web services defined by FSM's A and B in Figure 4. These FSM's are specifications of the execution trees in Figure 2. As expected, B can forward simulate A, but not vice versa. The dotted lines illustrate the simulation relation $\mathscr{R} = \{(p_0, q_0), (p_1, q_1), (p_2, q_1)\}$. □

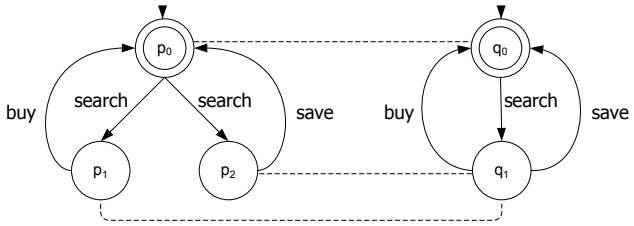


Fig. 4. Forward simulation $A \preceq A'$

It is easy to show that the two notions of forward simulation coincide.

Lemma 1. $A \preceq A' \Leftrightarrow T_A \preceq T_{A'}$

4 Web Service Communities

A *Web Service Community* is a set of cooperating Web Services $\{T_1, T_2, \ldots, T_k\}$, where each sub-service T_i is an execution tree with state space Q_i and action alphabet Σ_i. The action alphabet of the community is $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_k$.

Intuitively, actions of a Web Service Community are enacted by executing actions of the sub-services, either sequentially, or in parallel. This is formalized using a merge operator, inspired by [18].

The merge operator is a significant extension to the classical Roman model. Firstly, it allows for simultaneous execution of actions by two or several different services. This is a feature that is as essential to Web Services as it is to any distributed computations. Note that BPEL supports the `<flow>` construct that allows the specification of one or more activities to be performed concurrently.

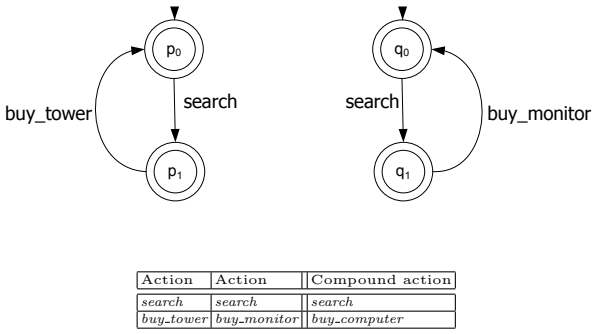
Secondly, the merge allows us to define processes that are composed of two or more sub-processes. This feature can be achieved in BPEL by using links within concurrent activities of the `<flow>` construct.

Formally, let a and b be actions. Then the *merge* of a and b, denoted $a \| b$, consists of either executing a followed by b, denoted $a.b$, or by executing $b.a$, (both $a.b$ and $b.a$ are sequential executions), or executing them in parallel. For the parallel execution we need a partial function $\gamma \colon \Sigma \times \Sigma \to \Sigma$. If $\gamma(a, b) = c$, then the parallel execution of a and b is visible externally as an action c. We demand that, if defined, $\gamma(a, b) = \gamma(b, a)$, and $\gamma(a, \gamma(b, d)) = \gamma(\gamma(a, b), d)$, for all $a, b, d \in \Sigma$, as γ models parallel execution (see [18]).

We now have two types of uses of the merge operator: The first is a parallel execution of one action. In this case we have $\gamma(a, a) = a$, as in the *search* action in Figure 5. The second use of merge is for task decomposition. When $\gamma(a, b) = c$ we take this to mean that externally visible action c is achieved internally by the community through executing a and b in parallel. The *buy_computer* action in Figure 5 is an example of an action achieved internally by two parallel actions, namely *buy_tower*, and *buy_monitor*.

Example 4. Consider the two Web Services defined by the FSMs A and A' given in the Figure 5 to the left and to the right, respectively. The two web services together form an web service community $\{A, A'\}$ and the figure also shows the merge capabilities of the community in the form a Table for the γ function.

The service A allows the user to search for and buy computer towers, while A' provides the same options for computer monitors. Now, what would be the semantics of this community? Intuitively, the executions of the individual web services can be interleaved in any possible order. In addition, with respect to the defined γ searches can be done in parallel, and simultaneously buying a tower and a monitor amounts to buying a computer. The merge of the A and A', with respect to the given γ is illustrated in Figure 6, □



Action	Action	Compound action
search	*search*	*search*
buy_tower	*buy_monitor*	*buy_computer*

Fig. 5. Community of FSM's

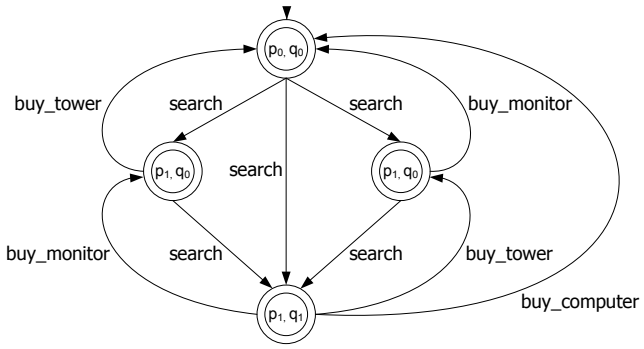


Fig. 6. Merge of community FSM's

5 Semantics of Web Service Communities

As the meaning of Web Service processes is given in terms of process trees, we need to define the community merge in terms of trees. We do this by extending the language-theoretic *shuffle* [15] with the γ-function. (Note that in classical process algebra the merge is defined axiomatically [10].)

Let $T : dom(T) \to Q \times \Sigma$, and $T' : dom(T) \to Q' \times \Sigma$, be trees, with final states $F \subseteq Q \subseteq \{1, \ldots, k\}^*$, and $F' \subseteq Q' \subseteq \{1, \ldots, m\}^*$. To define the *merge*, $T \| T'$, of trees T and T' we need to specify $dom(T \| T')$. For this, let $w \in dom(T)$ and $u \in dom(T')$. We set:

1. $\epsilon \| w = \{w\}$, $\epsilon \| u = \{u\}$, and $\epsilon \| \epsilon = \{\epsilon\}$
2. if $w = w'.i$ and $u = u'.j$, for some $i \in \{1, \ldots, k\}$, $j \in \{1, \ldots, m\}$ then $w \| u = (w' \| u)i \cup (w \| u')k + j$.
   Furthermore, if $\gamma(t(w), t'(u))$ is defined, then add the element $(w' \| u')k + m + 2^i \cdot 3^j$ to $w \| u$.

We can now define

$$dom(T \| T') = \{w \| u : w \in dom(T), u \in dom(T')\}.$$

We regard $dom(T \| T')$ as a subset of $\{1, \ldots, k + m + 2^k \cdot 3^m\}^*$.

Lemma 2. *$dom(T \| T')$ is a tree domain.*

For an arbitrary string w over $dom(T||T')$, we define w_l as the projection of w to the subsequence of symbols $\leq k$, and w_r as the projection of w onto the symbols $> k$ and $\leq m$.

By $T_Q(w)$ we mean p, where $T(w) = (p, \sigma)$, and by $T_\Sigma(w)$ we mean σ. Similarly for T'.

Now we can define $T \| T'(\epsilon)$ as $(\langle T_Q(\epsilon), {T'}_Q(\epsilon)\rangle, \lambda)$, and for all $wi \in dom(T \| T')$,

$$T \| T'(wi) = \begin{cases} (\langle T_Q(w_l i), {T'}_Q(w_r)\rangle, T_\Sigma(w_l i)), & i \leq k \\ (\langle T_Q(w_l), {T'}_Q(w_r i)\rangle, {T'}_\Sigma(w_r i)), & k < i \leq k + m \\ (\langle T_Q(w_l i), {T'}_Q(w_r i)\rangle, \gamma(T_\Sigma(w_l i), {T'}_\Sigma(w_r i)), & i \geq k + m + 6. \end{cases}$$

Example 5. Let us return to the community of the two services in Examples 4. Two execution trees T and T', corresponding to A and A' are given in Figure 7. The semantic merge of these two trees are shown in in Figure 8.

The semantic merge has the following properties:

Theorem 1. *$T \| T' = T' \| T$, up to isomorphism.*

Theorem 2. *$(T \| T') \| T'' = T \| (T' \| T'')$, up to isomorphism.*

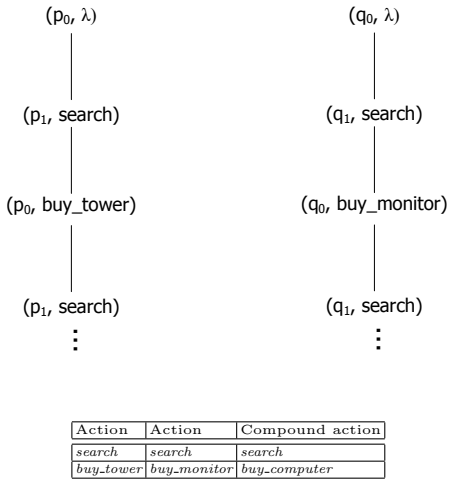
Given a web service community $(T_1, \ldots, T_k)$, we can now define its behavior as $T_1 \| \ldots \| T_k$

Theorem 3. *The semantics $T_1 \| \ldots \| T_k$ is uniquely defined, up to isomorphism.*

When the web services in a community are defined as finite state machines $A = (Q, \Sigma, \delta, p_0, F)$, and $A' = (Q', \Sigma', \delta', q_0, F')$, we can construct an FSM $A \| A' = (Q_\|, \Sigma_\|, \delta_\|, s_\|, F_\|)$, where $Q_\| = Q \times Q'$, $\Sigma_\| = \Sigma \cup \Sigma'$, $s_\| = (p_0, q_0)$, $F_\| = F \times F'$, and $\delta_\| \subseteq (Q \times Q') \times (\Sigma \cup \Sigma') \times (Q \times Q')$ is defined as follows: $\forall\ (p, a, q) \in \delta$, $\forall\ (q, b, q') \in \delta'$, we have the transitions $(\langle p, q\rangle, a, \langle p', q\rangle)$ and $(\langle p, q\rangle, b, \langle p, q'\rangle)$ in $\delta_\|$. Furthermore, if $\gamma(a, b) = c$ then $\delta_\|$ also contains $(\langle p, q\rangle, c, \langle p', q'\rangle)$.

By an encoding similar to those used in the proofs of Theorems 1 and 2 the following can be shown.

Theorem 4. *$T_{A \| A'} = T_A \| T_{A'}$* □



Action	Action	Compound action
search	*search*	*search*
buy_tower	*buy_monitor*	*buy_computer*

Fig. 7. Web service community

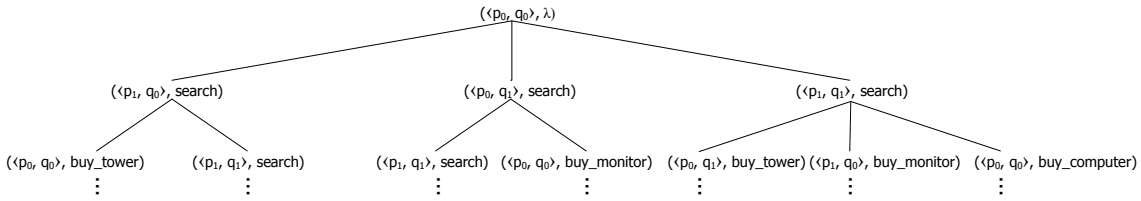


Fig. 8. Web service community execution tree

6 Synthesizing and Orchestrating Client Requests

The *raison d'être* of a Web Service community is to provide services to their clients. When a client requests a certain service there might be no individual Web Service in the community that can deliver it directly. However, as we already discussed in the previous section, the actions of individual Web Services in the community can be interleaved in any order and also executed in parallel. Therefore, it is possible that a *composite* Web Service can satisfy the client's request.

Service composition involves two main issues. The first, often called *composition synthesis*, is concerned with synthesizing a new composite Web Service, thus producing a specification of how to coordinate the Web services available in the community to provide a specified service. (This is the "compile time" phase.) The second, referred to as *orchestration*, is concerned with coordinating the correct execution of of an instance of a client request in the synthesized composition. (That is, the "run time" phase.)

Let us first formally define a service requested by a client, and what it means for Web Service community to be able to satisfy the request.

A *client request* is an execution tree $T_0\colon\ dom(T_0) \to Q_0 \times \Sigma$. We say that a web service community $\{T_0, \ldots, T_k\}$ can *satisfy* the client if $T_0 \preceq\ T_1 \| T_2 \| \ldots \| T_k$.

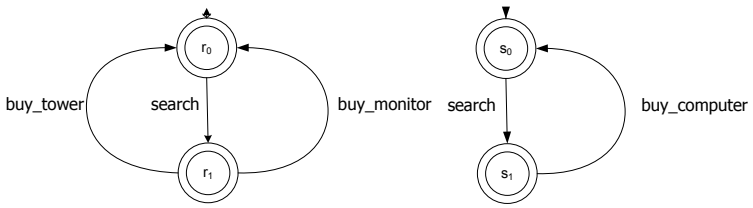


Fig. 9. Two web service clients

In practice each subservice T_i is specified by an FSM A_i, and the client by an FSM A_0. We are thus interested in whether $T_{A_0} \preceq\ T_{A_1} \| T_{A_2} \| \ldots \| T_{A_k}$. For this we need

Theorem 5. *$T_{A_0} \preceq T_{A_1} \| T_{A_2} \| \ldots \| T_{A_k}$ if and only if $A_0 \preceq\ A_1 \| A_2 \| \ldots \| A_k$.*

Proof. By Lemma 1, we have $A_0 \preceq\ A_1 \| A_2 \| \ldots \| A_k$ iff $T_{A_0} \preceq\ T_{A_1 \| A_2 \| \ldots \| A_k}$. By repeatedly applying Theorem 4 we have that $T_{A_1 \| A_2 \| \ldots \| A_k}$ is isomorphic to $T_{A_1} \| T_{A_2} \| \ldots \| T_{A_k}$. The claim now follows. □

Let the client FSM be $A_0 = (Q_0, \Sigma, \delta_0, p_0, F_0)$ and let the community be $\mathscr{A} = \{A_1, \ldots, A_k\}$, where $A_i = (Q_i, \Sigma_i, \delta_i, q_{0_i}, F_i)$. We denote by $\overrightarrow{q_0}$ the tuple $\langle q_{0_1}, q_{0_2}, \ldots, q_{0_k} \rangle$, where $q_{0_i} \in Q_i$ is the start state of A_i. Similarly, $\overrightarrow{q}$ denotes a tuple of states (not necessarily initial) from $A_1, A_2, \ldots A_n$.

The following algorithm computes the simulation relation $A \preceq \mathscr{A}$.

Forward-Sim$(A, \mathscr{A})$

```
1  U ← ∅
   ▷ pairs of states (p, q) such that p cannot be simulated by q.
2  repeat
3          R ← ∅
           ▷ simulation relation
4          W ← ∅
           ▷ pairs of states (p, q) visited more than once
5          S ← {(p_0, q_0→)}
           ▷ pairs of states (p, q) in the currently checked sequence
           Reliability ← TRUE
6          Result ← Check(p_0, q_0→)
7      until Result = FALSE ∨ Reliability = TRUE
8  if Result = FALSE
9      then R ← ∅
10 return R
```

```
CHECK(p, q⃗)
 1  if p ∈ F ∧ ∃q_{i_j} ∈ q⃗ such that q_{i_j} ∉ F_j
 2     then U ← U ∪ {(p, q⃗)}
 3          S ← S \ {(p, q⃗)}
 4          return FALSE
 5  if {(p, a, p′) : (p, a, p′) ∈ δ, a ∈ Σ, p′ ∈ Q} = ∅
 6     then R ← R ∪ {(p, q⃗)}
 7          S ← S \ {(p, q⃗)}
 8          return TRUE
 9  for each (p, a, p′) ∈ δ
10        do V ← {(q_{m_o}, ..., q_{l_j}, ..., q_{n_k}) :
                        q⃗ = (q_{m_o}, ..., q_{i_j}, ..., q_{n_k}) ∧ (q_{i_j}, a, q_{l_j}) ∈ δ_j}
           ∪
                 {(q_{m_o}, ..., q_{r_j}, ..., q_{s_h}, ..., q_{n_k}) :
                        q⃗ = (q_{m_o}, ..., q_{i_j}, ..., q_{l_h}, ..., q_{n_k})∧
                        (q_{i_j}, b, q_{r_j}) ∈ δ_j ∧ (q_{l_h}, c, q_{s_h}) ∈ δ_h ∧ γ(b, c) = a}
11           if V = ∅
12              then U ← U ∪ {(p, q⃗)}
13                   S ← S \ {(p, q⃗)}
14                   if (p, q⃗) ∈ W
15                      then Reliability = FALSE
16                   return FALSE
17           Flag ← FALSE
18           while V ≠ ∅
19                do
20                   V ← V \ {q⃗′}
21                   if (p′, q⃗′) ∉ U
22                      then if (p′, q⃗′) ∈ R
23                                then Flag ← TRUE
24                                else if (p′, q⃗′) ∉ S
25                                        then S ← S ∪ {(p′, q⃗′)}
26                                             Flag ← Flag ∨CHECK(p′, q⃗′)
27                                        else W ← W ∪ {(p′, q⃗′)}
28                                             Flag ← TRUE
29  if Flag = FALSE
30     then S ← S \ {(p, q⃗)}
31          U ← U ∪ {(p, q⃗)}
32          if (p, q⃗) ∈ W
33             then Reliability = FALSE
34          return FALSE
35  R ← R ∪ {(p, q⃗)}
36  return TRUE
```

The algorithm is based on the partial depth first search approach used for "on-the-fly" verification of behavioral equivalences introduced in [13].

The procedure CHECK starts with a pair of start states and traverses the client FSM constructing "on-the-fly," as needed, part of the community FSM in depth first order. The construction of the next state of $\mathscr{A}$ is based on the transitions of the component FSMs $A_1, \ldots A_k$, and on the γ-table of the community.

First the procedure checks that if the state of A_0 is a final state then the corresponding state of $\mathscr{A}$ is also a final state, that is, all of the component states are final. Next, the procedure checks whether the state of A_0 has any outgoing transitions, and, if this is not the case, then the corresponding state of $\mathscr{A}$ can definitely simulate it. After that, it picks one after another the transitions going out of the state of A_0 and checks if there is at least one corresponding transition in $\mathscr{A}$. If the algorithm finds such transition, it checks the pair of states that these transitions are incident upon. If no corresponding transition exists in $\mathscr{A}$, then, obviously, $\mathscr{A}$ cannot simulate A_0

The list S of transitions that are considered in the current traversal is maintained in order not to go in cycles. When the algorithm sees the same pair of states again it does not visit them again, but starts returning and constructing the simulation relation. Thus, the pairs of states are visited in prefix order, while the conditions for the simulation relation are checked in postfix order. Consequently it is possible to reach a pair of states $(p, \overrightarrow{q})$ which have already been visited, but not yet determined to be in the simulation relation. In this case the algorithm makes an optimistic assumption that $(p, \overrightarrow{q})$ will be determined to be in $\mathscr{R}$ at a later time. When this pair is eventually analyzed by the algorithm (the algorithm maintains a set W of such pairs), if it is determined not to belong to the simulation relation, then the procedure sets the flag *Reliability* to FALSE, which means that the simulation relation that the algorithm has constructed is not guaranteed to be correct as the optimistic assumption used by the algorithm was wrong.

If the algorithm decided that A_0 cannot be simulated by $\mathscr{A}$ this decision is always reliable because not assumptions are used to decide the FALSE. However, if the procedure CHECK returned TRUE the *Reliability* has to be TRUE as well, otherwise the simulation relation constructed by the procedure has to be discarded and CHECK has to be called again. In order to omit repeating the same dead-end traversals the algorithm maintains global list U of pairs that are determined to be not in the simulation relation by previous calls to CHECK.

We note that Shukla et al. [20] previously have given a decision procedure for testing whether one FSM can be forward simulated by another. This is achieved by a reduction to a variant of Horn clause satisfiablity. However, their procedure does not construct an actual simulation relation.

Theorem 6. *Algorithm* FORWARD-SIM$(A_0, \mathscr{A})$ *runs in time* $\mathcal{O}(|\delta_0| \times |\delta|)$*, where* $|\delta_0|$ *and* $|\delta|$ *denote the sizes of the transition relations in* A_0 *and* $\mathscr{A}$*, respectively.*

Since the size of the community FSM $\mathscr{A}$ is at most exponential in the size of the component FSM's $A_1, \ldots, A_k$, the following holds.

Corollary 1. *The algorithm* FORWARD-SIM *runs in EXPTIME.*

Client Action	Community Action
$r_0 \xrightarrow{search} r_1$	$p_0 \xrightarrow{search} p_1$ $q_0 \xrightarrow{search} q_1$
$r_1 \xrightarrow{by\_tow} r_0$	$p_1 \xrightarrow{by\_tow} p_0$
$r_1 \xrightarrow{by\_mon} r_0$	$q_1 \xrightarrow{by\_mon} q_0$

Fig. 10. Orchestration of left client

Fig. 11. from Figure 9

Client Action	Community Action
$s_0 \xrightarrow{search} s_1$	$p_0 \xrightarrow{search} p_1$ $q_0 \xrightarrow{search} q_1$
$s_1 \xrightarrow{by\_comp} s_0$	$p_1 \xrightarrow{by\_tow} p_0$ $q_1 \xrightarrow{by\_mon} q_0$

Fig. 12. Orchestration of right client from Figure 9

Theorem 7. *The algorithm* FORWARD-SIM *is sound and complete.*

Example 6. Consider a client given by the FSM on the left in the Figure 9, and the Web Service community $\{A_1, A_2\}$ from Figures 5 and 6.

Since we have $\gamma(search, search) = search$, allowing the searches to be executed in parallel on many services, it is easy to see that there indeed is a simulation relation from A_0 to $A_1 \| A_2$, namely $\mathscr{R}\{(r_0, \langle p_0, q_0\rangle), (r_1, \langle p_1, q_1\rangle)\}$. From this an orchestration engine (a Mealy-machine) can straightforwardly be constructed. For simplicity we illustrate the orchestration engine informally in the table in the Figure 10.

The need for γ can be easily seen, as this client cannot be simulated by the community in Figure 6, unless we use the look-ahead mechanism from [14]. However, look-ahead is not always possible, e.g. postponing the payment transaction in on-line shopping. □

In the full paper we show how a process algebra description can be obtained from the simulation relation computed by our algorithm. Then, the process algebra description can be automatically translated to executable BPEL-code, using the two way mapping between process algebra and BPEL given in [12]. The following example illustrates this, and points out important BPEL constructs that can be handled by our extended model, and which were not present in the *ESC* implementation of the classical Roman model [4].

Example 7. Consider the right client Figure 9. We can satisfy this client, as we have $\gamma(buy\_tower, buy\_monitor) = buy\_computer$. In this case the simulation relation from the client to $A_1 \| A_2$ is $\mathscr{R} = \{(s_0, \langle p_0, q_0\rangle), (s_1, \langle p_1, q_1\rangle)\}$

Note that γ allows composite services to be merged in the external action alphabet of a web service community, thereby also achieving ease of integration of new services into the community. The simulation relation for the right client is given informally as the table in the Figure 11.

The BPEL pseudo-code for the actual implementation of this service is given in Figure 13. Note that there are two `<flow>` constructs in this pseudo-code. The first one results from $\gamma(search, search) = search$ The two participating services are invoked to perform *search* in parallel. The second corresponds to $\gamma(buy\_tower, buy\_monitor) = buy\_computer$. □

7 Conclusions and Future Directions

We have develop a rigorous extension to the Roman model and given a formal semantics using a process algebraic approach [18, 10]. Process algebra works well at the stages of design and formal verification of web services [12, 19]. In our work we show that this approach gives advantages if used at the process mediation stage as well. Our extension allows for fuller coverage of the standard languages used for description and execution of Web Services in practice and unifies the modeling and mediating aspects of Web Services. Moreover, our extension allows for formal verification of mediated Web Services though the use of the numerous tools available for process algebra.

We gave an algorithm for mediating a requested Web Service from a resource pool of more basic services. The algorithm constructs the required composition "on the fly" without constructing the FSM for the entire Web Service community. The produced composition is complete in the sense that it covers all alternatives and the final decisions can be made at run time, based on the availability of the component services, network bandwidth, or some cost model. The algorithm runs in the exponential time which is the same as in the classical Roman model.

To see that the proposed solution is in fact practical consider the complete mediation procedure, which starts with the executable web services, produces the abstract descriptions (here is a possibility to formally verify some properties of the available services, if required), composes the abstract descriptions to produce a mediated service (at this stage also formally verifiable) and, finally, translates the mediated service to an actual executable web service.

We are currently working on the implementation of a prototype system where the mediator extracts algebraic descriptions from existing web services and the abstract specification of the produced mediated web service is translated into executable BPEL code, using the techniques in [12]. We are also working on further extensions of the formal model.

References

1. Marco Aiello, Mikio Aoyama, Francisco Curbera, and Mike P. Papazoglou, editors. *Service-Oriented Computing - ICSOC 2004, Second International Conference, New York, NY, USA, November 15-19, 2004, Proceedings*. ACM, 2004.
2. Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Automatic composition of e-services that export their behavior. In Maria E. Orlowska, Sanjiva Weerawarana, Mike P. Papazoglou, and Jian Yang, editors, *ICSOC*, volume 2910 of *Lecture Notes in Computer Science*, pages 43–58. Springer, 2003.
3. Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. A foundational vision of e-services. In Christoph Bussler, Dieter Fensel, Maria E. Orlowska, and Jian Yang, editors, *WES*, volume 3095 of *Lecture Notes in Computer Science*, pages 28–40. Springer, 2003.
4. Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Massimo Mecella. Esc: A tool for automatic composition of services based on logics

of programs. In Ming-Chien Shan, Umeshwar Dayal, and Meichun Hsu, editors, *TES*, volume 3324 of *Lecture Notes in Computer Science*, pages 80–94. Springer, 2004.
5. Daniela Berardi, Diego Calvanese, Giuseppe De Giacomo, Richard Hull, and Massimo Mecella. Automatic composition of transition-based semantic web services with messaging. In Böhm et al. [8], pages 613–624.
6. Tevfik Bultan, Xiang Fu, Richard Hull, and Jianwen Su. Conversation specification: a new approach to design and analysis of e-service composition. In *WWW*, pages 403–410, 2003.
7. Daniela Berardi, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella, and Diego Calvanese. Synthesis of underspecified composite -services based on automated reasoning. In Aiello et al. [1], pages 105–114.
8. Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors. *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. ACM, 2005.
9. BPEL. Business process execution language for web services (version 1.1), May 2003.
10. J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001.
11. Zhe Dang, Oscar H. Ibarra, and Jianwen Su. Composability of infinite-state activity automata. In Rudolf Fleischer and Gerhard Trippen, editors, *ISAAC*, volume 3341 of *Lecture Notes in Computer Science*, pages 377–388. Springer, 2004.
12. Andrea Ferrara. Web services: a process algebra approach. In Aiello et al. [1], pages 242–251.
13. Jean-Claude Fernandez and Laurent Mounier. "on the fly" verification of behavioural equivalences and preorders. In Kim Guldstrand Larsen and Arne Skou, editors, *CAV*, volume 575 of *Lecture Notes in Computer Science*, pages 181–191. Springer, 1991.
14. Cagdas Evren Gerede, Richard Hull, Oscar H. Ibarra, and Jianwen Su. Automated composition of e-services: lookaheads. In Aiello et al. [1], pages 252–262.
15. J. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Language, and Computation*. Addison–Wesley, Reading, MA, 1979.
16. ISO. Lotos: a formal description technique based on the temporal ordering of observational behaviour. Technical Report 8807, International Standards Organisation, 1989.
17. Nancy A. Lynch and Frits W. Vaandrager. Forward and backward simulations: I. untimed systems. *Inf. Comput.*, 121(2):214–233, 1995.
18. R. Milner. A calculus on communicating systems. *Lecture Notes in Computer Science*, 92, 1980.
19. Gwen Salaün, Lucas Bordeaux, and Marco Schaerf. Describing and reasoning on web services using process algebra. In *ICWS*, pages 43–. IEEE Computer Society, 2004.
20. Sandeep K. Shukla, Harry B. Hunt III, Daniel J. Rosenkrantz, and Richard Edwin Stearns. On the complexity of relational problems for finite state processes (extended abstract). In Friedhelm Meyer auf der Heide and Burkhard Monien, editors, *ICALP*, volume 1099 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 1996.
21. WSMX. Web service execution environment, June 2005.

```
<process name = "ComputerPurchase" ...>
   ...
   <partnerLinks>
      <partnerLink name = "customer" .../>
      <partnerLink name = "towerStore" .../>
      <partnerLink name = "monitorStore" .../>
      ...
   </partnerLinks>
   <variables>
      <variable name = "searchRequest" .../>
      ...
   </variables>
   ...
   <pick>
      <onMessage partnerLink = customer
         portType = ...
         operation = "searchRequest"
         variable = ...>
         <sequence>
            ...
            <flow>
               <invoke partnerLink = "towerStore"
                  portType = ...
                  operation = "searchRequest"
                  ... />
               <invoke partnerLink = "monitorStore"
                  portType = ...
                  operation = "searchRequest"
                  ... />
            </flow>
            ...
            <pick>
               <onMessage partnerLink = customer
                  portType = ...
                  operation = " buy_computerRequest"
                  variable = ...>
                  <sequence>
                     ...
                     <flow>
                        <invoke partnerLink = "towerStore"
                           portType = ...
                           operation = "buy_towerRequest"
                           ... />
                        <invoke partnerLink = "monitorStore"
                           portType = ...
                           operation = "buy_monitorRequest"
                           ... />
                     </flow>
                     ...
                  </sequence>
                  ...
               </onMessage>
               ...
            </pick>
            ...
         </sequence>
         ...
      </onMessage>
      ...
   </pick>
   ...
</process>
```

Fig. 13. BPEL pseudo-code for the orchestration of the right client in Figure 9

The mediator centric approach to Automatic Web Service Discovery of Glue

Emanuele Della Valle, Dario Cerizza and Irene Celino

CEFRIEL - Politecnico of Milano, Via Fucini 2, 20133 Milano, Italy
dellavalle@cefriel.it, cerizza@cefriel.it, celino@cefriel.it

Abstract. Automatizing the Web Service Discovery is crucial for truly implementing a Service Oriented Architecture. Semantics has been shown to be useful. Several initiative, namely OWL-S, WSMO and WSDL-S, are successfully employing ontologies, but we believe that WSMO is right in highlighting mediation as *the* missing element. In this paper we provide a mediator centric refinement of the conceptual model for WSMO discovery and the related architecture as well as the prototypical implementation (named Glue) we are using in the projects COCOON and Nomadic Media.

1 Introduction

In a Services Oriented Architecture (SOA) a requester entity may not know which provider entity to engage. A requester entity should only know the functional criteria of the service it wishes to interact with and it can find out suitable candidate services by inquiring a *discovery agency*. In this way such discovery agency decouples requester entities from provider ones.

Web Services are meant to enable the implementation of a SOA. As reported in the Web Service glossary[1], **discovery** is "*the act of locating a machine-processable description of a Web Service that may have been previously unknown and that meets certain functional criteria*". UDDI [1] provides a general purpose model for Web Service discovery by gathering metadata about a collection of Web Services and making that information available in a searchable way (white, yellow and green pages). But, as stated in section 1.4.5 "Overview of Engaging a Web Service" of W3C Note on WSA [2], such meta-data requires initial knowledge about both Web Service existence (e.g., which tModel specified by EAN/UCC it implements) and location (e.g., the name of a green page category). UDDI does not provide any formal and explicit way to exchange such initial knowledge; therefore today the most common approach to obtain it is through e-mail exchanges or word of mouth. This requires to keep humans in the loop and limits scalability as well as economy of Web Services.

The Semantic Web is making available technologies which support (and, at some degree, automate) knowledge sharing. In particular, several initiatives (e.g.

[1] http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

OWL-S[2], WSMO[3], WSDL-S[4]) provide evidence that ontologies, with their ability to interweave human understanding of symbols with their machine processability, can play a key role in automating Web Service Discovery [3][4][5][6][7]. All approaches share the idea that:

- *at publishing time*, a set of relevant domain ontologies can be used to semantically annotate Web Service descriptions (i.e., describing the capabilities of such Web Services),
- *at discovery time*, the *same set of ontologies* can be used to describe the functional criteria of the service the requester entity wishes to interact with; therefore, the Discovery engine, accessing the knowledge modeled in the ontologies, is not limited to syntactic matching techniques, but it can take into consideration also semantic matching techniques (e.g. subsumption base matching [5], transaction logic [8]).

The problem is that such approach may work in a small controlled environment, but it is unrealistic in open environments in which actors suffers from various form of **polarization** (e.g., philosophical position, sense of belonging, etc.). We observe that in the real world:

1. in the same domain *different proposals for domain ontologies supported by competing cartels* are under development and are likely to be maintained also in the future.

   For instance, in the health care field, in which the need for sharing a common knowledge has been addressed for more than a decade, several competing terminology standards are available for describing pathologies, e.g. International Classification of Disease[5] (ICD) and SNOMED Clinical Terms[6].
2. *the points of view of providers and requesters on the shared knowledge are different*, and this is reflected in different ontologies respectively used for annotating Web Services and for formulating requests.

   For instance, consider the knowledge of date-time used in the field of arranging meetings for consultancy. A week-based calendar (e.g. each Thursday afternoon and Friday morning) may be preferred by provider entities in stating the nominal availability of a consultant, whereas a Gregorian calendar (e.g. April, 9th from 10.00 to 12.00) may be preferred by a request entity in requiring advice. Even if both provider and requester entities share the same knowledge on date-time, they may prefer to use different ontologies.

A possible objection is that OWL already provides constructs to reason on equality (e.g. `sameAs`, `equivalentClass`, `equivalentProperty`, etc.), but those are not sufficient to deal with ontology alignment.

The Web Service Modeling Ontology (WSMO) working group has moved a step forward in the direction of semantic empowered Web Service Discovery [8][9]

[2] http://www.daml.org/services/
[3] http://www.wsmo.org/
[4] http://lsdis.cs.uga.edu/projects/wsdl-s/
[5] http://www.who.int/classifications/icd/en/
[6] http://www.snomed.org/

by introducing the notion of *mediation* [10]. Mediation in WSMO is, together with strong decoupling, a foundation principle. *Mediators address the problem of handling the heterogeneities that naturally arise in open environments.* WSMO proposes a classification of mediators according to their role in WSMO conceptual model: Ontology to Ontology mediators (ooMediators), Goal to Goal Mediators (ggMediators), Web Service to Goal Mediators (wgMediators), etc[7].

A model for automatic location of services (i.e. service discovery plus service contracting) in WSMO is introduced in [9] and in [12]. Such model provides the basis for a variety of solutions that are easy to use for requesters, and that provides efficient pre-filtering of relevant services and accurate contracting of services that fulfill a given requester goal. Moreover the preliminary architecture for WSMO compliant discovery engines is presented in WSMO Discovery Engine D5.2 [13] and in WSMX Discovery D10 [14].

In this paper we provide a **refinement of WSMO discovery conceptual model** centered on mediators (cf. section 2):

- by making the notion of class of goals and class of Web Service descriptions explicit,
- by using ggMediators for automatically generating a set of goals semantically equivalent to the one expressed by the requester but expressed with a different form or using different ontologies;
- by *making wgMediators the conceptual element responsible for evaluating the matching*;
- by using *ooMediators* for solving any terminological mismatch that can appear with different polarized ontologies for the domains, and
- by redefining the *discovery mechanism as* ***a composite procedure*** *where the discovery of the appropriate mediators and the discovery of the appropriate services is combined.*

Moreover, in section 3, we provide a description of the **refinement of WSMX Discovery Engine architecture** according to our refined WSMO discovery conceptual model both in terms of components and execution semantics (cf. section 3.1) and we show how we **implemented it in** Glue[8] using F–logic[15] (cf. section 3.2). In section 4 we introduce a *use case of Service Discovery in the healthcare field* and we show how we put Glue at work in the projects COCOON[9] and Nomadic Media[10]. Finally in section 5 and 6 we respectively present related works and we draw some conclusions providing insight view of our future developments.

[7] We do not provide here a complete list of them and we prefer to refer to WSMO D.2 [11] for a detailed explanation of their usage.

[8] http://glue.cefriel.it

[9] COCOON is a 6th Framework EU integrated project aimed at setting up a set of regional semantics-based healthcare information infrastructure with the goal of reducing medical errors.

[10] The Nomadic Media project resides under the Eureka/ITEA program. Within this project we addressed the problem of out to provide mobile access to healthcare services.

2 The concept

In this paper we strictly refer to the automatic location of services proposed in [9] for WSMO. We state this because we agree with [16] that when talking about Web Services many notions (including *service*) are semantically overloaded and one should commit to a precise meaning when using such notions.

We agree on the definition of Service (including the distinction between *abstract* and *concrete* service) and of requester need (that reflects the conceptual element of WSMO named *goal*). We hold with the idea of pre-defined goals and with parameterizing them, so that a requester has only to locate (eventually unconsciously by interacting with a standard application) one of the pre-defined goals and to provide concrete values for its parameters. But we prefer to reserve the name goal for a concrete goal whereas we suggest to call **class of goals** a specific *parametrized and pre-defined goal*.

We agree with the idea of having parametrized and pre-defined Web Service descriptions that can be instantiated by the provider entities, so that two concrete descriptions differ in the values given to the parameters. Consider, for instance, a scenario where there are some technical specialists offering virtual meetings through a collaboration platform. This platform exposes one distinct Web Service for booking a meeting with each specialist. The structures of all these Web Services are similar but the Services provided are different since they depends on the technical capabilities of each specialist. The Web Service descriptions of these Web Services are multiple instances of the same class of parametrized pre-defined Web Service description. As we suggest for goals, we prefer to make explicit the notion of **class of Web Service description** (a pre-defined Web Service description) and of *instance of Web Service description* (a Web Service description).

Having in mind this class-based approach, we can adopt the inheritance model typical of Object-Oriented programming languages, where a class can be subsequently refined in child classes providing a more specific structure and behavior. In this way, it is possible to extend classes of Web Service descriptions and classes of goals in a more fashionable manner, useful in domains where a lot of similar but non-identical Web Services need to be discovered.

In order to describe our concept, we consider the whole process in three subsequent phases: *Set up time*, *Publishing time* and *Discovery time*.

During *Set up time*, the system is being initialized with all the necessary information for performing automatic service discovery in a generic domain $\mathbf{D}$. This information includes domain ontologies, Web Service description classes, goal classes and the required mediators. As discussed in the introduction, given a specific domain $\mathbf{D}$, it is natural that provider's point of view is reflected in a particular polarized $\mathbf{D}^+$ domain understanding and, on the other hand, requester's point of view is reflected in a differently polarized $\mathbf{D}^-$ domain understanding. Thus, having different points of view the standard process of agreement on an ontology for the domain $\mathbf{D}$ would require a lot of effort. Instead of developing one ontology, we suggest to follow the activities shown in figure 1. The Semantic Web service expert (SWS Expert) is the main actor involved, since he is able

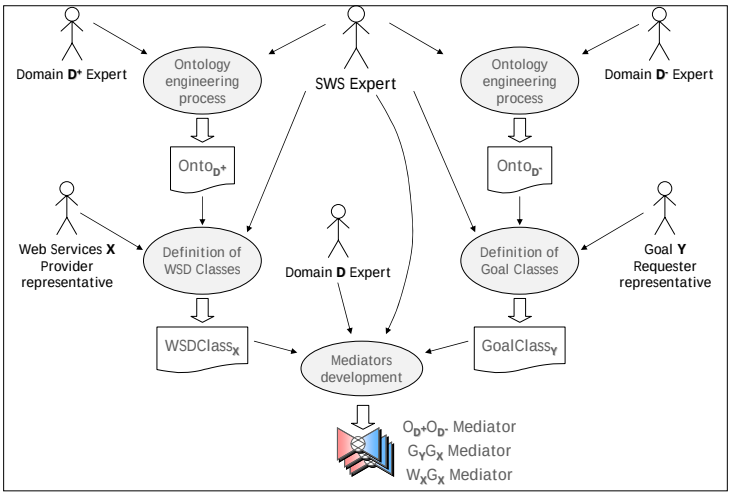


Fig. 1. The sequence of activities performed by actors at Set up time

to understand the internal semantic notation of the system. Domain ontologies for $\mathbf{D}^+$ and $\mathbf{D}^-$ are developed (integrating already available sources) during separated engineering processes that involve the relative domain experts with the SWS Expert. When the ontologies are ready, a provider representative and the SWS Expert define the class of Web Service descriptions for the class **X** of Web Services using the ontology for domain $\mathbf{D}^+$. Meanwhile, a requester representative and SWS Expert define the class **Y** of goals. As highlighted, the SWS Expert does not have neither to ask provider and requester entities to commit to the same set of ontologies (which may prove to be an unworkable plan) nor to develop a complete mediator between all the conflicting ontologies (which may be unnecessary if only a small portion of such ontologies is involved in the matching process). For this reason, the SWS Expert needs to be supported by a non-polarized expert for the domain **D**, that can help him solving ontology alignment problems between the two different domains. **This process generally involves the definition of various mediators**. In our case, a ooMediator, a ggMediator and a wgMediator will work, but more complex scenarios can be imagined. The first mediator implements the domain-specific rules for solving terminology mismatches between the involved portion of ontology $\mathbf{D}^+$ and ontology $\mathbf{D}^-$. The ggMediator is developed in order to translate the instance of goal **Y** into semantically equivalent instances of goals that are more suitable for discovery instances of Web Services **X** (later named goal **X**). This mediator uses the functionalities provided by the previously developed ooMediator. The latter wgMediator implements the domain-specific rules for matching instances of just-translated goal **X** with instances of Web Service description **X**.

At *publishing time*, provider entities publish instances of Web Service descriptions simply by referring to the correct class of Web Service description

and providing values to all the necessary parameters. The system creates the instances and register them in its internal repository in order to retrieve them when a wgMediator will require it.

Finally, *at discovery time*, when a goal instance is created and submitted, a *look up* mechanism can be used for selecting the class of goals (being the submitted goal an instance of a class). Similarly, a look up mechanism can be used for selecting the set of appropriate ggMediators able to translate the user goal in a set of semantically equivalent goals. Then again, the appropriate wgMediators can be looked up and the instances of Web Services descriptions can be roughly *filtered* on the basis of the target of the detected wgMediators. A set of ooMediators is used to solve any terminological mismatch. And, finally, the ***similarity rules coded in the wgMediators can be used to match*** the given goal against the instances of Web Service descriptions obtained by filtering, returning an ordered list of references to the concrete Web Services.

3 Glue Architecture and Implementation

The goal of our work was to design and build a system suited for medium scale deployment (up to some tens of classes of Web Service Description and of classes of goals, using ontologies of a couple of thousand concepts each, but with some hundreds of instances of Web Service Description in each class) while providing a lightweight stand alone implementation[11].

3.1 Architecture

Having a such mediator–centric vision in mind, we propose to refine the architecture described in WSMO/X Discovery [13][14] accordingly to the refinement we propose in section 2. WSMO/X discovery architecture envision the presence of a *Communication Manager* (which handles the incoming requests), two *WSMO Element Repositories* (one for goals and one for Web Service descriptions), a *Message Parser* (which analyzes the incoming message), a *Proof Generator* (which constructs logical formulas for running the matchmaker) and a *Reasoner Manager* (which abstracts from the different interfaces of the reasoners).

We propose to extend the WSMO Discovery Engine architecture, refining several **components** (see figure 2). First, we decomposed the Message Parser into three components: Web Service description constructor, Goal Constructor and Goal Translator. The two *Constructor Components* serve for instantiating respectively the instances of Web Service descriptions and goals of classes registered at set up time. The *Goal Translator* looks up ggMediators that are appropriate for the goal instance passed by the goal constructor and translates such goal in a set of provider–oriented goals. Secondly, we refine the *Proof Generator* component giving it also some of the responsibilities of the *Message Parser*. The part we include in the Message Parser is responsible for looking up the

[11] In WSMX the discovery engine is part of a much larger architecture and is not implemented as a stand alone component.

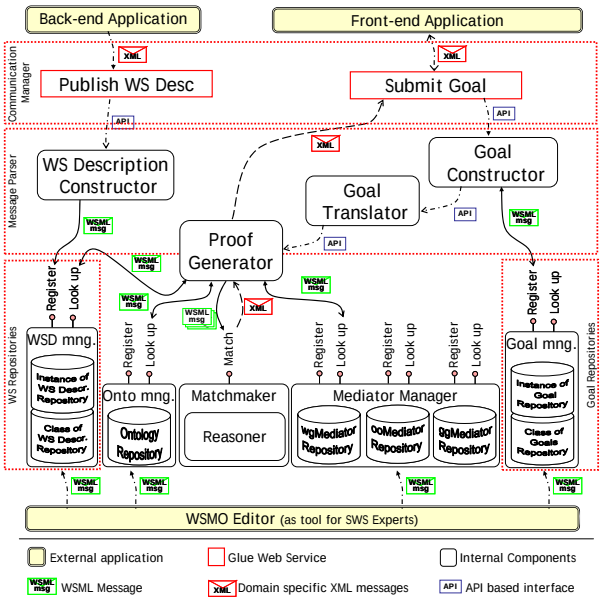


Fig. 2. The architecture of the mediator centric extension to the WSMO Discovery Engine we propose in this paper. Dashed lines represent the components described in the original WSMO Discovery Engine described in [14], while the other components are those we introduce in this paper.

mediators and the ontologies required for executing the matching whereas the part outside the Message Parser roughly filters the Web Services descriptions, it loads them and it executes the logical formulas for running the matchmaker. Aside, we introduce all those components that satisfies our need for a lightweight implementation of Glue, even if they are generally envisioned as part of the entire WSMX architecture [17]: the two Repositories, one for ontologies and one for mediators, and a WSMO Editor (such as WSMO Studio) as tool for SWS Experts.

In figure 3, we depict the **execution semantics** describing the interactions among the components both in case a new Web Service description is published (the interactions labeled with 'a', 'b' and 'c') and when a goal is submitted (the interactions labeled with numbers).

The first set of interactions describe how the "publish Web Service description communication component" invokes the WS Description Constructor passing a class identifier and a set of parameters (cf. line 'a'), how this component uses the class identifier to look up the class of Web Service description (cf. line 'b'), how it instantiates the Web Service description and how it registers the constructed instance in the Web Service description instances repository (cf. line 'c').

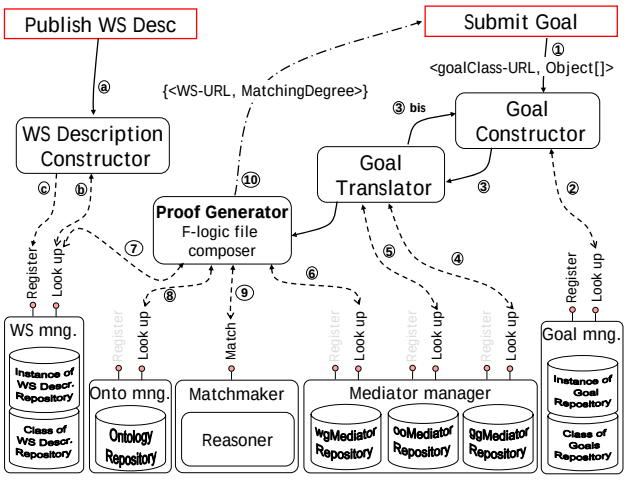


Fig. 3. This collaboration diagram depicts the components of Glue and their role in supporting the discovery process.

The second set of interactions describes our *composite discovery procedure* that expects as input an identifier of a class of goals together with a set of structured parameters (cf. line 1) and provides, as result (cf. line 10) a list of pairs in which the first element is a reference to a Web Service and the second element is the degree of matching [12].

The first part of the procedure (cf. line 2) consists in looking up the WSML description of the class of goals corresponding to the given identifier and instantiating the concrete representation of the goal using the given parameters.

The second part of the procedure concerns the use of ggMediators in translating the constructed instance of goal in an equivalent instance expressed using a different terminology (cf. line 4). In turn, each ggMediator optionally uses an ooMediator for reconciliating semantic heterogeneities (cf. line 5). If necessary, the Goal Translator component can call back the Goal Constructor (cf. line "3 bis") for instantiating other goals instances.

The third part of the procedure is undertaken by the Proof Generator. This component is responsible for constructing the formulas for running the matchmaking. For each instance of goal, it starts looking up the wgMediators that *has as target* the class of goal having such goal as an instance (cf. line 6). Then, for each wgMediator, the Proof Generator looks up the Web Service description instances of the class that *is the source* for such wgMediator (cf. line 7) and it also looks up all the required ontologies (cf. line 8). All the retrieved ontologies and instances are sent to the matchmaker (cf. line 9), that uses the reasoner for

[12] exact, subsumed, plug-in and intersection as proposed in WSMO D5.1 and in many other earlier works related to OWL-S [5][6]

evaluating the similarity rules coded in each wgMediator, and returns references to the discovered Web Services and the degree of matching as a list of pairs (cf. line 10).

3.2 The prototype implementation of Glue

In developing Glue, a prototype of this proposed architecture, we faced two major choices: the reasoner and the format for the logical language. We knew that we needed both an ontological language and a rule language, but we were also constrained by the efficiency we aim at.

WSML working group[13] proposes in [18] a subset of OWL (named OWL$^-$) that can be translated into f–logic [19]. This, according to WSML working group, allows for efficient query answering and for easy implementation of a rule on top of the ontology. Moreover, we were looking for an expressive datatype support and WSML working group showed in [20] that OWL–E [21] (a proposal for extending OWL with expressive datatype expressions) can be added to OWL$^-$ and the resulting ontological language, named OWL-Flight, can still be translated into f–logic. But, at the time we started developing Glue (September 2004), WSML efforts, in providing a language for formalizing WSMO, were a work–in–progress and the tools for translating WSML into reasoner–specific formats were missing. Therefore we decided to directly use a dialect of f–logic implemented in Flora–2, an open source f–logic inference engine that runs over XSB[14], an open source implementation of tabled-prolog and deductive database system. This choice allowed for an early proof of concept without constraining compatibility with WSML.

Plugging Flora–2 in Glue involved some consequent implementation choices: the proof generator in the prototype is mainly a *f–logic file composer*; all the information exchanges, which we envision in the architecture as based on WSML, are implemented directly using f–logic; and all the WSMO element repositories manages directly f–logic files, keeping in the internal SQL syntax the necessary relationships between mediators, ontologies and classes.

4 A case of mediator centric Discovery for eHealth

Most of times, Service Discovery is treated as a back-end problem, but actually, if Semantic Web Services efforts will succeed, the use of Service Discovery tools in the future will be as frequent as using search tools today. Therefore, in COCOON project, beside envisioning a clear back-end use of Service Discovery (in combination with healthcare application protocols such as HL7), we also envision a realistic use case of its daily employment. In this section, we describe the usage scenario for Service discovery as implemented in COCOON project and some details about the real usage of Glue in this scenario.

[13] http://www.wsmo.org/wsml/
[14] http://xsb.sourceforge.net/

4.1 Usage scenario

In COCOON project, we are evolving a usage scenario[15] of Web Service Discovery that describes an interaction between a General Practitioner (GP) and COCOON platform with the intent to find *medical advice* and *teaching* services offered by specialists organized in communities of practice (CoP). In particular, Glue takes responsibility for enabling on demand access to services for arranging virtual meetings; the actual arrangement and the subsequent meeting is supported by the collaboration services provided by the COCOON platform that acts as a front-end application from the Glue point of view (see figure 2).

The general criteria for matching a GP goal against the description of the services offered by a CoP are the correspondence between the GP's problem and CoP's medical capabilities and the matching between the GPs' date–time preferences and the nominal availability time of each CoP.

Medical advice will predictably be the most frequent reason for a GP to start an interaction, as it normally could be triggered during the practice time (e.g. questions by patients). Most of times, the *clinical capabilities* of the CoP may be the ones the GP wishes. But, sometimes (e.g. for more difficult and rare patient cases) *research capabilities* of the specialists involved in a CoP may be sought.

Teaching, on the other hand, will be predictably less frequent and a reason to start a request to the system could happen in the 1-hour/week time that is normally reserved for contacting peers[16], as it normally could be triggered by GP's reflection on his/her week's practice.

In order to facilitate the understanding of this scenario, in figure 4, we show Glue surrounded by a set of CoPs (which are provider entities) and a COCOON platform (which is a requester entity). Each CoP exposes the functionality of arranging the two types of meeting as a Web Service. As discussed in 2, the process that enables a GP to arrange a meeting with the most suitable CoP can be broken down in the following tasks:

- Set up time:
  1. the service provider and requester entities **agree on the ontologies to use** for modeling pathologies (e.g. they may agree on using the ICD), drugs (e.g. they may choose International Nonproprietary Names for Pharmaceutical Substances – INN[17]), advice services, date–time, etc.;
  2. **if they cannot agree** on the use of a specific set of common ontologies, the use of mediators is required as discussed in 2 . In this scenario, the CoP providers and the requester entities cannot agree on the use of a common date–time ontology. The CoP provider entities prefer to express the nominal availability of each CoP using a *week-based calendar* (e.g. the advice service is available on Thursday afternoon and Friday morning),

[15] The various versions of our usage scenarios are publicly available at http://cocoon.cefriel.it/RD2/usecases/

[16] As reported in a February 2005 national survey of Italian GPs.

[17] http://www.who.int/medicines/organization/qsm/activities/qualityassurance/inn/orginn.shtml

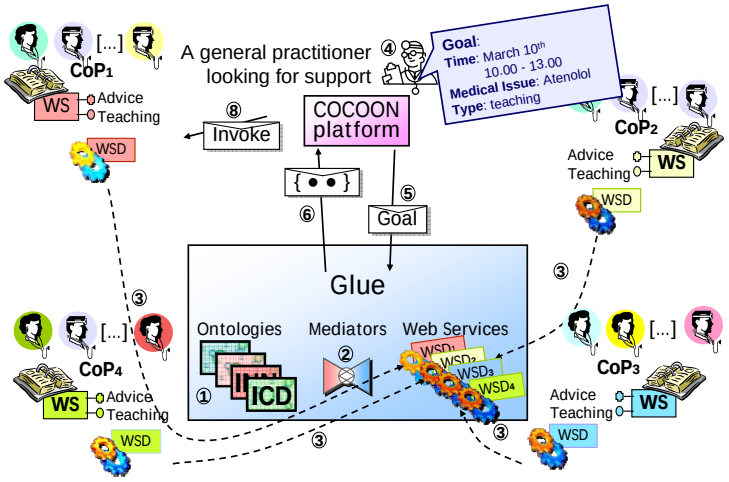


Fig. 4. A case of Service Discovery that enables a general practitioner to find the most appropriate medical advice/teaching service.

whereas the requester entity prefer to express users' preferences using a *Gregorian calendar* (e.g. is the service available on April, 9th from 10.00 to 12.00?);

– Publishing time:

3. each CoP provider entity **publishes** inside Glue its Web Service descriptions for arranging a meeting, describing the clinical capabilities the CoP holds and the date–time intervals the CoP is normally available (the nominal availability for advice and teaching may differ). For instance, a CoP provider entity may register its CoP as "*a community that delivers intervention based on alpha and beta blockers with nominal availability on Monday, Tuesday and Friday in the afternoon for advice and on Tuesday for teaching*";

– Discovery time:

4. similarly, a GP can discover the most suitable CoP by using a GUI, provided by COCOON platform, in order to **express his/her goal** in term of the available ontologies. For instance the GP asks "*a teaching session on the use of Atenolol, preferring the meeting to be arranged on June 8th from 10.00 to 13.00 or on June 9th from 13.00 to 16.00*";
5. COCOON platform **submits** the goal to Glue;
6. Glue **performs the composite discovery process** described in previous sections **matching** the GP goal against the descriptions of the advice/teaching services offered by each CoP; then it returns a list of references to Web Services for arranging a meeting, ordered by decreasing relevance;
7. the **results list is displayed** to the GP;

8. the GP interactively **selects** one of CoPs untill he/she finds one to arrange a meeting with.

4.2 Putting Glue at work

In order to test Glue, we modeled in WSMO the use case just illustrated. We used f–logic to describe the ontologies, the classes/instances of Web Services descriptions, the classes/instances of goals and the wgMediators. Then, we populated Glue with some tens of realistic descriptions of Web Services for arranging meetings with a CoP.

The **ontologies** necessary to support this use case are the COCOON medical ontology, the advice/teaching services ontology and two calendar ontologies.

COCOON ontology is a demonstrative ontology of hypertension and breast cancer domains derived from ICD-10 and INN. It contains the definition of a hundred concepts (like `disease`, `hypertension`, `breast neoplasm`, etc., `medication`, `beta-blockers`, etc., `part of the body`, `heart`, etc., `specialist`, `cardiologist`, etc.) and the relations among them (like beta blockers `control` hypertension, cardiologists `deal with` heart, hypertension `affects` heart and arteries, etc.).

The *advice/teaching ontology* describes the concepts of clinical, research and teaching capabilities of a Community of Practice.

- *Clinical Capabilities* describes the CoP in terms of:
  - `hasClinicalSpecialists`: the list of the kind of specialists grouped by the CoP (e.g. Cardiologist, Urologist, Pneumatologist, Dermatologist, etc.),
  - `managesDiseases`: the list of diseases managed by the CoP as ICD codes (e.g. Diabetes – ICD9CM 250.00), and
  - `deliversInterventions`: the list of the diagnostic / therapeutic / preventive interventions (including pharmaceuticals) delivered by the CoP;
- *Clinical Research Capabilities* describes the CoP in terms of
  - `hasResearchSpecialists`: the list of the kind of specialists grouped by the CoP (e.g. Statistician, Social worker, Psychologist),
  - `studiesDiseases`: the list of diseases which are actively researched by the CoP (e.g. Gastric ulcer [ICD10–K25] Prevention), and
  - `studiesInterventions`: the list of the diagnostic / therapeutic / preventive interventions (including pharmaceuticals) which are actively researched by the CoP; and
- *Teaching Capabilities* describes the CoP in terms of
  - `hasTeachingExpertise`: the list of teaching roles that the CoP can fulfill (e.g. Teacher, OnlineTeacher, Tutor, OnlineTutor, etc.)
  - `hasAuthoringExpertise`: states the availability of online/offline collaborative working tools (i.e. for teaching) within the CoP (e.g. NetMeeting, Skype, Messenger, etc.)

Finally, as discussed in 4.1 two *calendar ontologies* are necessary in our use case to express the date–time intervals. Therefore an *ooMediator* has been employed in translating the date–times from the Gregorian calendar to the week-based one. In our implementation, this ooMediator was realized with a Java program exposed as a Web Service used at discovery time by Glue.

Having these ontologies, we were able to describe in WSMO the capabilities of the class of **Web Services** for arranging a meeting with a CoP. We define a class hierarchy of Web Service descriptions with a generic class on top (for meeting arrangement in a given set of date–time intervals) and two specific classes below (for arranging respectively advice and teaching meetings).

The description of the generic meeting arrangement class of Web Service asserts that:

- the *pre-conditions* are: the input has to be the information about an advice request, the general practitioner has to ask an advice on one of the medical issues treated by the various CoPs; and the booking date has to be after the current date;
- the only *assumption* is that the general practitioner has the right to use the advice service;
- the *post-conditions* describe the possible meetings the CoP is available for: it can offer support that regards its capabilities and it can provide support only during its nominal available times;
- the *effect* is that the agendas of both the GP and the specialists in the CoP are updated with a reference to the scheduled meeting.

In a similar manner, we defined a hierarchy of **classes of goals** that asserts GP's need of finding a CoP that can provide advice or teaching support on a given medical issue in the date–times intervals the GP prefers.

As described in section 4.1, in our use case no agreement was reached in the date–time ontology to use. To bypass such heterogeneity we defined also a parallel hierarchy of classes of goals that express the GP goal in terms of the week-based calendar ontology (the one chosen by CoP providers) and we used a **ggMediator** for translating instances of goal from one class to the other. This ggMediator, when invoked, simply rewrites the goal formulated by the GP using Gregorian dates (e.g. June, 8th 2005), translating it into days of the week (e.g. Wednesday) through the ooMediator illustrated above.

Finally, as we described in section 3.2, we expect SWS Expert to encode in Glue a set of **wgMediators** with the similarity rules for matching a class of goals against a class of Web Services descriptions. For instance, the rule that performs an exact match between what the `GP is asking for` and the medical capabilities of a CoP says that there is an exact matching when:

- the `GP is asking for` a specialist and
  - the CoP `has that clinical specialist`,
  - or the CoP `manages a disease` that affects a body part dealt by the specialist the GP is asking for,
  - or the CoP `delivers an intervention` that controls one of diseases treated by the specialist the GP is asking for,
- the `GP is asking for` a disease and
  - the CoP `has a clinical specialist` that deals with a body part affected by the disease the GP is asking for,
  - or the CoP `manages the disease` that the GP is asking for,
  - or the CoP `delivers an intervention` that controls the disease the GP is asking for,

– the `GP is asking for` an intervention and
  - the CoP `has a clinical specialist` that deals with a body part affected by a disease controlled by the intervention the GP is asking for,
  - or the CoP `manages a disease` controlled by the intervention the GP is asking for,
  - or the CoP `delivers the intervention` that the GP is asking for.

The rules for subsume and plug-in matching mainly differ from the one presented above because they broaden the search space to subconcepts and superconcepts respectively, navigating the COCOON domain ontology. Beside these rules that match medical capabilities, there are other different rules that matches date-time intervals between goal and Web Services description.

Having two parallel hierarchies of classes, we wrote three wgMediators: one links a generic service for arranging a meeting to a generic goal for requesting support, while the other two link respectively a service for arranging advice meetings to a request for advice and a service for arranging a teaching meeting with a request for teaching support. Since the rules in the three wgMediators largely overlap, we found useful the possibility of building also hierarchies of wgMediators, so that the two specific wgMediators can be defined by extending the generic one and reusing its rules.

For lack of space we don't present in this paper all the internal f–logic syntax of our scenario. Readers can refer to Glue Web site[18] for more detailed information.

5 Related works

The work we present in this paper is strictly related to the activities of WSMO/L/X working groups. Like other articles proposed in the context of WSMO, it moves away from OWL-S. In OWL-S approach goals and Web Service description must be defined using the same ontologies and the automation of Web Service discovery normally relays on subsuption reasoning (e.g. [4][22][6]). In WSMO, on the other hand, goals and Web Services can be annotated using different ontologies and the relationships between them can be captured by the mean of wgMediators. This, clearly, requires to step aside the lack of rules in the OWL languages. In this paper, we proposed to use f–logic rules within the wgMediator to capture the knowledge for reconciliating the conflicts that arise when goal and Web Service descriptions are not annotated using the same ontologies. This approach is similar to an early work on OWL-S, in which the service discovery problem is formulated as a rewriting problem where requests are attempted to be rewritten in terms of available services [3], but it takes a much easier approach in coding the matching rules directly in the wgMediator.

The relatively easiness in describing classes of Web Service description and of goals and in coding the matching rules makes the approach we propose very efficient. We easily modeled the presented use case and the performances[19] of

[18] http://glue.cefriel.it/

[19] the machine we used for the tests is a 2800MHz Pentium 4 processor with 1 gigabyte of RAM

the COCOON Glue Discovery Engine with 50 Web Service descriptions remains under 3 seconds. However one can move to our approach the criticism that it does not relies on generic notion of matching such as subsumption reasoning or transaction logic. Our position is that such criticism is only partially true, because one can encode in the wgMediator a subsumption based matching, but at the time we are writing this article we cannot provide evidence that such approach is convenient.

6 Conclusion and future work

The main lesson we are learning bin applying WSMO in the healthcare field is that the clear separation between the ontologies used by each entity simplifies and speeds up the gathering of consensus, which is alway difficult to reach in large groups, and especially in healthcare field. This is mainly due to the adoption, in WSMO, of mediators. In particular, wgMediators appear to offer a flexible way for describing similarities between goals and Web Service descriptions, hence for enabling a semantic match between them.

Finally, the tasks that are currently being the subject of our research are:

- WSMO discovery
  1. extending our approach with the notion of *intention* as presented in [9], such extension will provide COCOON Glue WSMO Discovery Engine with more degrees of matching and it will enable future support for contracting;
  2. aligning our work with the WSML family of languages that are currently being defined as part of the WSML working group activity, in particular with WSML-rule; and
  3. aligning our work with the WSMX architecture providing COCOON Glue WSMO Discovery Engine as a WSMX plug-in.
- the COCOON project
  1. selecting and adjusting the ontologies required for describing the other healthcare services offered in COCOON, through the development of (possibly ad-hoc) mediation services to overcome heterogeneity of the various healthcare related ontologies;
  2. extending the test cases of our Discovery Engine to include the other components still under development in COCOON project (i.e. semantic information retrieval and clinical guideline based decision support system); and
  3. extending the approach to the regional eHealth services (starting from the SISS in Lombardy - Italy).

Acknowledgements

The research has been supported by the COCOON (IST FP6-507126) integrated project and Nomadic Media ITEA (Information Technology for European Advancement) project, financed by the Italian Public Authorities. We thank Prof. Stefano Ceri, Doc. Michele Tringali, Lara Gadda, Maria Rodriguez and Irene Celino for their precious comments and support.

References

1. OASIS: The uddi technical white paper. Technical report, OASIS (2004)
2. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web services architecture. Technical report, W3C (2004)
3. Benatallah, B., Hacid, M.S., Rey, C., Toumani, F.: Request rewriting-based web service discovery. In: International Semantic Web Conference. (2003) 242–257
4. Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A semantic web approach to service description for matchmaking of services. In: SWWS. (2001) 447–461
5. L. Li, I. Horrocks: A software framework for matchmaking based on semantic web technology. (2003)
6. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: International Semantic Web Conference. (2002) 333–347
7. Oldham, N., Thomas, C., Sheth, A.P., Verma, K.: Meteor-s web service annotation framework with machine learning classification. In: SWSWPC. (2004) 137–146
8. Kifer, K., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., Fensel, D.: A logical framework for web service discovery. In: Semantic Web Services Worshop at ISWC. (2004)
9. Keller, U., Lara, R., Lausen, H., Polleres, A., Fensel, D.: Automatic location of services. In: ESWC. (2005) 1–16
10. Wiederhold, G.: Mediators in the architecture of future information systems. IEEE Computer **25** (1992) 38–49
11. Roman, D., Lausen, H., Keller, U.: D2 web service modeling ontology (WSMO). Technical report (2005)
12. Keller, U., Lara, R., Polleres, A., Toma, I., Kifer, M., Fensel, D.: D5.1 WSMO Web Service Discovery. Technical report (2004)
13. Keller, U., Lara, R., Lausen, H., Polleres, A., Predoiu, L., Toma, I.: D5.2 WSMO Discovery Engine. Technical report (2004)
14. Keller, U., Lara, R., Lausen, H., Polleres, A., Predoiu, L., Toma, I.: D10 WSMX Discovery. Technical report (2005)
15. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM **42** (1995) 741–843
16. Preist, C.: A conceptual architecture for semantic web services. In: International Semantic Web Conference. (2004) 395–409
17. Cimpian, E., Moran, M., Oren, E., Vitvar, T., Zaremba, M.: D13.0 overview and scope of WSMX. Technical report (2005)
18. de Bruijn, J., Polleres, A., Lara, R., Fensel, D.: Owl-. Technical report, WSML (2004)
19. Angele, J., Lausen, G.: Ontologies in f-logic. In: Handbook on Ontologies. (2004) 29–50
20. de Bruijn, J., Polleres, A., Lara, R., Fensel, D.: D20.3 OWL flight. Technical report, WSML (2004)
21. Pan, J. Z., Horrocks, I.: OWL-E: Extending owl with expressive datatype expressions. Technical report, IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester (2004)
22. K. Verma, K. Sivashanmugam, A. Sheth, A. Patil: METEOR-S WSDI: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. (2003)

Mediating Capabilities with Delta-Relations

Michael Stollberg[1], Emilia Cimpian[2], and Dieter Fensel[1,2]

[1] Digital Enterprise Research Institute Innsbruck (DERI Austria),
Institute for Computer Science, University of Innsbruck,
Technikerstrasse 21a, A-6020 Innsbruck, Austria
[2] Digital Enterprise Research Institute (DERI Ireland),
IDA Business Park, Lower Dangan, Galway, Ireland

Abstract. Mediation is concerned with handling heterogeneities that potentially occur between resources that shall interoperate. Heterogeneity being an inherent characteristic of open and distributed environments like the Internet, mediation becomes a core issue for next generation Web technologies. Recent developments around the Semantic Web and Semantic Web services address mediation on the data level and on the process level. This paper identifies the *teleological level* as a novel level of mediation that deals with heterogeneities of capabilities as the functional descriptions of Web services and service requests. The central mediation technique therefore are so-called Δ-relations that explicitly denote the logical relationships between capabilities. These can be used to perform central reasoning tasks for Semantic Web Services by simple inferences instead of more complex reasoning procedures, hence allow gaining efficiency in Semantic Web service technologies.

1 Introduction

The initial Web service technology stack around SOAP, WSDL, and UDDI remains on a syntactic level for describing Web services that limits Web service usage to manual inspection and integration. As this is considered to fail as a basis for dynamic service-oriented architectures, the emerging discipline of Semantic Web services develops semantically enabled technologies for automated discovery, composition, communication, cooperation, and execution of Web services. On basis of exhaustive semantic description frameworks and usage of ontologies as the underlying data model, Semantic Web services strive towards an integrated technology for realizing the vision of the Semantic Web [19], [6].

Apart from enabling advanced techniques for automated Web service usage, a main merit of Semantic Web service technology is the inherent support for handling heterogeneities on a semantic level [8]. Heterogeneity is an inherent characteristic of the Internet that hampers successful and efficient inter-operation of Web services, requests, and other resources. Semantic resource descriptions allow utilization of semantically enabled techniques for depicting and resolving heterogeneities. This is commonly referred to as mediation, wherefore different levels are distinguished with respect to the type of heterogeneities that can occur and techniques used for handling these.

The most prominent frameworks for Semantic Web services address mediation as follows. OWL-S [18] defines an ontology for semantically describing Web services that is comprised of Service Profiles, Service Models, and Grounding as top-level elements. Mediation is not considered as a central element but as an architectural aspect arising in concrete Web service systems. However, the intention of OWL-S is to provide a semantic description model for Web services while leaving technology development open to respective efforts; following this, OWL-S is declared to be orthogonal to mediation [23]. In contrast, the Web Service Modeling Ontology WSMO [16] depicts mediation as an integral aspect and hence defines an architectural model for mediators along with techniques for handling heterogeneities at different levels.

Ongoing research and development efforts on mediation address the data level including heterogeneities on terminologies and representation formats, and the process level as mismatches in Web service communication and cooperation. This paper identifies an additional level of mediation that deals with heterogeneities between functional descriptions of Web services and requests. Arising from decoupled and decentralized development, these might not hamper Web service usage but can cause significant decelerations in automated Web service technologies. This can be overcome by explicitly defining the logical relationship between functional descriptions that allow replacing complex procedures for central reasoning tasks by simpler inferences. We refer to this as the *teleological level of mediation* wherefore this paper presents so-called Δ-relations as the main mediation technique and usage in mediators for gaining efficiency in Semantic Web service technology.

This paper is structured as follows: Section 2 recalls the state of affairs in mediation techniques developed for Semantic Web services and introduces the teleological level of mediation; Section 3 presents the definition of Δ-relations and their usage in mediators; Section 4 exposes the benefits of teleological level mediation within exemplary scenarios; Section 5 discusses related work, and Section 6 concludes the paper.

2 Mediation Levels and Techniques

In order to motivate the new level of mediation introduced in this paper, the following recalls the need for mediation within Semantic Web services and depicts the potential benefits and requirements for teleological level mediation.

Semantic Web services aims at developing integrated technology for automating the complete Web service usage process [12]. This consists of the following, possibly iterative steps wherefore semantically enabled mechanisms are developed. At first, create Web service implementations along with their semantic description and make them accessible (*publication*), then detect appropriate Web services for solving a given request (*discovery*) or combine several services therefore (*composition*), choose the most adequate Web service out of the applicable ones for a request (*selection*), and finally access the chosen Web service (*invocation*) and control the information interchange for completing the service usage

(*conversation*). Orthogonal to this, heterogeneities might occur that hamper automated Web services usage. Handling these is subject to mediation that becomes a major concern in Semantic Web and Semantic Web service technologies with regard to the open and decentralized nature of the Web.

Hence, architectural models for Semantic Web services are proclaimed that treat mediation as a first class citizen [8]. Following [25], a mediator is an architectural component capable of establishing interoperability of resources if not given a priori by resolving heterogeneities. Aiming at generic, domain and application independent mediation facilities, mediation techniques are envisioned that work declarative resource descriptions for detecting and handling heterogeneity on the semantic level. The ultimate aim of an integrated mediation framework as aspired in WSMO [20] is to provide means for handling and resolving all kinds of heterogeneities that might hamper Web service usage. Therefore, potentially occurring heterogeneities are classified into different levels with respect to the distinct mediation techniques and architectural requirements needed for handling these. The following inspects heterogeneity types and respective mediation levels along with recent developments and then reveals the necessity for teleological level mediation.

2.1 Data and Process Level Mediation

One kind of heterogeneity that can occur is usage of different terminologies by entities that shall interchange information. Within ontology-based environments like Semantic Web services, this means that heterogeneous ontologies are used as the terminological basis for element descriptions which hinders prosperous information interchange. Mediation techniques for handling terminological mismatches on an ontological level are ontology mapping, merging, and alignment, collectively referred to as ontology integration techniques [1]. Mismatches that hamper information interchange can also result from usage of different data representation formats or technical transfer protocols. A suitable way of resolving such heterogeneities is to lift the data from the syntactic to an ontological level, resolve the mismatches on this level, and then lower them again to the syntactic representation [22]. Because these two types of heterogeneity can be handled by similar techniques, they are consolidated as *data level mediation* [21].

Another type of heterogeneity can occur on the behavioral level that hamper entities from interacting successfully with respect to their individual business processes. For instance, at some point during the interaction of a requester R with a Web service S, R expects an acknowledgement while S waits for the next input; so, the interaction process between R and S runs into a deadlock situation. Such heterogeneities can be resolved by inspecting the individual business processes of the entities that shall interact and establish a valid process for interaction on basis of generic mediation operations on business processes. This is referred to as *process level mediation* [4].

The need for data and process level mediation for establishing interoperability of Web services and related elements if not given a priori has already been indicated in [8].

2.2 Functional Heterogeneity

Realization of Semantic Web technology reveals another type of heterogeneity that occurs as functional differences between Web services and requests. With respect to the distributed nature of the Web and the accompanying dispersed description of Web services and requests, the usual case is that the requested functionality does not precisely match with the one provided by a Web service. If we would have additional information on the relationship between functional descriptions, we could use this for improving the efficiency of semantic matchmaking components.

Consider the following example that we examine in Section 4 in more detail. There is a request R of 'finding information on Italian restaurants in Innsbruck' and some available Web services: WS_1 offers a hotel and restaurant guide for Innsbruck, WS_2 is a Tyrol restaurant guide, and WS_3 provides information on restaurants with traditional Tyrolean cuisine in Innsbruck. For detecting the Web services WS_1 and WS_2 to be usable for achieving R, we need to run a discovery procedure between R and each service. Existing developments for discovery with semantic matchmaking consist of complex, expensive reasoning procedures (see [17], [13], [15], [24]).

This effort can significantly be reduced when knowing and considering the relationship between R and the Web Services. Imagine that we have the following additional information: R is a specialization of another request R_O for finding information on restaurants, and from a previous discovery run, we know WS_1, WS_2, and WS_3 are usable for resolving R_O. If we compute Δ_{R,R_O} as the logical relationship between the requests, we can determine the Web services usable for resolving R as those usable for resolving R_O that also satisfy Δ_{R,R_O}. Hence, we can replace costly discovery runs for determining usable Web services for R by a much easier, straightforward, more efficient inference.

This is the motivation and aim of what we refer to as *teleological level mediation*. Heterogeneities as in the example appear as differences between functional descriptions, i.e. between OWL-S service profiles or WSMO capabilities. Specifying the functionality of a service or the one required for satisfying a request, these descriptions are concerned with the application purpose of Web services which can be referred to as the teleological level of Semantic Web service description ontologies. Knowing the explicit differences between functional descriptions of Web services and requests allows increasing the efficiency of Semantic Web service technologies by replacing complex reasoning tasks by simpler ones throughout the Web service usage process.

3 Teleological Level Mediation Techniques

With regard to the above examinations, the following introduces techniques for teleological level mediation. The basis of our approach are so-called Δ-relations that explicitly denote the logical relationship between functional resource descriptions. We provide the definition of Δ-relations, expose their beneficial usage, and integrate them into the WSMO mediator architecture.

3.1 Δ-Relations Definition

As a basis for efficient resource management by additional information on the teleological level as outlined above, a Δ-relation denotes the explicit logical relationship between functional resource descriptions. Following [2], this can most appropriately be described as the logical difference.

Hence, we refer to this a Δ-relation that consists of two elements: the **Δ-expression** states the logical difference between functional descriptions, and the **Δ-situation** that denotes the type of the relationship between them. Both can be computed for given functional descriptions. In order to provide a general definition that is adaptable to the semantics of functional descriptions in respective frameworks, we apply a set-theoretic model. Referring to [14] for details, the set-theoretic model defines that if ϕ is a functional description it is interpreted as a subset of the universe $\mathcal{U}$ (that is all possible instances of the ontologies used as terminologies in ϕ) that satisfies ϕ, i.e. $\phi \subset \mathcal{U}$.

Following this, the Δ-expression between two given arbitrary logical formulas ϕ and ψ is their union without their intersection: $\Delta_{\phi,\psi} = \phi \cup \psi \setminus (\psi \cap \phi)$. This means that Δ contains those elements that are models for either ϕ or ψ and not common to them. Considering the above example of two requests R_ϕ = 'finding restaurants in Innsbruck' and R_ψ = 'finding Italian restaurants in Innsbruck', Δ_{R_ϕ,R_ψ} would be 'all restaurants in Innsbruck that are not Italian'. Using de Morgan's laws, we can simplify this formula as follows: $\Delta_{\phi,\psi} = \phi \cup \psi \setminus (\psi \cap \phi) = \phi \setminus (\psi \cap \phi) \ \cup \ \phi \setminus (\psi \cap \phi) = \phi \backslash \psi \ \cup \ \psi \backslash \phi$. This states that the desired logical difference between ϕ and ψ is the union of ϕ without ψ and ψ without ϕ.

The Δ-situation denotes the type of relationship between formulas, respectively functional descriptions by commonly used keywords. In the example, R_ψ is a subset of R_ϕ; we denote is a subsumption relation between R_ϕ and R_ψ. As it holds in this situation that only those Web services usable for R_ϕ might be usable for R_ψ by no others, the information on the Δ-situation appears to be relevant with respect to the aspired usage for efficient resource management outlined above. Hence, we define five Δ-situations that naturally comply with the degrees of matching identified in [17], [14]. While in discovery these are used for denoting the type of commonality between logical expressions, we use them to denote the type of difference. The following defines the Δ-situations and the simplified computation of the corresponding Δ-expression:

1. **equal**: $\phi = \psi => \Delta = \emptyset$.
   this means that the models for ϕ and ψ are exactly the same so that there does not exists any logical difference between them.
2. **plugin**: $\phi \subset \psi => \Delta = \psi \backslash \phi$.
   this means that all models of ϕ are also models for ψ but not vice versa. We can also say that ϕ is subsumed by ψ.
3. **subsume**: $\phi \supset \psi => \Delta = \phi \backslash \psi$.
   as the opposite of the plugin situation, this means that all models of ψ are also models for ϕ but not vice versa. We say that ϕ is subsumes ψ. (the differentiation of the situations "subsume" and "plugin" gets important for enabling efficient reasoning mechanisms, as discussed below).

4. **intersecting**: $\phi \cap \psi \neq \emptyset => \Delta = \phi \backslash \psi \ \cup \ \psi \backslash \phi$.
   if there is no proper specialization or generalization but there exist models common for ϕ and ψ, then the Δ between them is their union without their intersection - i.e. we cannot simplify the computation of the Δ-expression.
5. **disjoint**: $\phi \cap \psi = \emptyset => \Delta = \emptyset$.
   if there does not exist any common model for ϕ and ψ, then we consider the Δ-expression to be empty as there is no correlation between the formulas.

The above definitions provide a general definition of Δ-relations between arbitrary logical formulas that can be applied for WSMO capabilities as follows. As the description of the functionality provided by a Web service as well as for the requested functionality in Goals, a WSMO capability is comprised of shared variables, preconditions and assumptions that denote the pre-state, and postconditions and effects that denote the post-state. While the four latter elements are defined by axioms, the scope of shared variables is the complete capability that allows specifying the coherence between the pre-state and post-state description elements of a capability. The intended semantics is that if the input provided is a valid model for the pre-state, then the execution of the Web service or the solution of the Goal will result in a post-state that is dependent of the respective pre-state. A formal semantics is under development at the point of writing, which is based on the notion of Abstract State Spaces wherein a Web service is understood as a set of state transitions from an initial to a termination state (see [7] for details).

Following this, we cannot write a WSMO capability definition in a single logical formula - at least not without respective signature renaming. Hence, we denote the Δ-relations between two WSMO capabilities $C_1 = (\phi_{pre}, \phi_{ass}, \phi_{post}, \phi_{eff})$ and $C_2 = (\psi_{pre}, \psi_{ass}, \psi_{post}, \psi_{eff}$ as a tuple of the Δ-relations between the corresponding description elements whereby the distinct Δ-relations are computed by the above methods. Hence, under consideration of all ontologies O and mediators M used in the description of two WSMO capabilities C_1, C_2, the logical relationship between them is defined as follows:

$$\mathcal{O}, \mathcal{M}, \mathcal{C}_1, \mathcal{C}_2 \ \models \ \Delta_{C_1,C_2} = (\Delta_{\phi_{pre},\psi_{pre}}, \Delta_{\phi_{ass},\psi_{ass}}, \Delta_{\phi_{post},\psi_{post}}, \Delta_{\phi_{eff},\psi_{eff}}) \quad (1)$$

This allows performing the desired reasoning tasks for improving the efficiency of resource management in Semantic Web service technologies. Thereby, we can compute the Δ-relations between the description elements of capabilities and reason on these. Dependent on what is to be achieved by working with delta-relations, we can also transform WSMO capability definitions into single logical formulas. Δ-relations between OWL-S Profile descriptions can be defined in a similar way.

The set-theoretic definition for computing the Δ-expression is transformed into the respective description language. For instance, when dealing with first-order logic expressions ϕ and ψ the Δ-expression between them is defined by $\Delta_{\phi,\psi} = (\phi \wedge \neg\psi) \vee (\neg\phi \wedge \psi)$; the definition is analogue for functional descriptions that use description logics or logic programming with respect their semantics.

3.2 Using Δ-Relations for Gaining Efficiency

After identifying the potential of teleological level mediation and definition of Δ-relations as the main mediation technique, the following exposes the benefits attainable for Semantic Web service technologies.

As outlined introductory, the main merit of teleological level mediation is to increase efficiency in Semantic Web service technologies. With respect to this, we distinguish two functional purposes for beneficially utilization: (1) support for *problem and functionality specification by reuse and refinement*, and (2) *creation of element ontologies with additional information on the teleological level*. While the former purpose mainly refers to support for creating the semantic description of goals and Web services, the latter facilitates efficiency in mechanisms for automated discovery and composition of Semantic Web services. We explain this in more detail.

For illustration purpose, let's consider the following example. A goal G_1 defines *buy product*, and another goal G_2 defines *buy ticket*, whereby *ticket* is sub-class of *product* in the used domain ontology. Considering the capability specification of G_1 to be ϕ, and the one of G_2 to be ψ, then there is a subsume situation $\phi \supset \psi$ so that $\Delta = \phi \backslash \psi$. For the first usage scenario, imagine that G_1 already exists and some user wants to define G_2. As G_2 is a teleological refinement of G_1 (means: both goals have the same structure, but the object of interest in G_2 is narrower than the one of G_1), we can use a mediator $\mathcal{M}_{G_1,G_2}$ that contains Δ_{G_1,G_2} for automatically deriving the specification of G_2 as it holds: $G_2 = G_1 \backslash \Delta_{G_1,G_2}$. Similar, we can create the capability specifications of interrelated Web services. Following the concept of weakening and strengthening for describing Problem Solving Methods [10], this simplifies the creation of problem and functionality descriptions.

Besides, we attain additional information on the teleological relationship between elements that are interconnected by mediators with Δ-relations. Considering such an element collection as a graph, the goals and Web services represent the nodes and the mediators with Δ-relations denote the arcs that explicitly define the teleological relationship between the goals and Web services. We refer to such collections of semantically interlinked elements as *teleological element ontologies* that provide additional teleological level information and can be used for improving efficiency of central reasoning tasks for Semantic Web services.

For instance, referring to the above example, imagine that from previous runs of a Web Service discovery engine we have determined a set of Web Services that are applicable for resolving goal G_1: $WS_{G_1} = (WS_1, WS_2, ..., WS_n)$. Because of the situation $G_1 \supset G_2$ we know that the set of applicable Web Service for resolving G_2 can only be equal or a subset of those applicable for G_1: $WS_{G_2} \subseteq WS_{G_1}$. Hence, we can derive WS_{G_2} as those Web services out of WS_{G_1} that satisfy Δ_{G_1,G_2} in the mediator $\mathcal{M}_{G_1,G_2}$ outlined above as it holds: $WS \in WS_{G_2} \leftarrow WS \in WS_{G_1} \wedge \ satisfy(WS, \Delta_{G_1,G_2})$. Although becoming more complicated when taking Δ-relations between the goals and Web services into account (see Section 4), this shows that we can omit invocation of a discoverer for determining Web services satisfying G_2 - which most presumably is more

expensive than checking this simple inference. Hence, teleological level mediation significantly decreases the reasoning effort in Semantic Web service technology by following the approach of gaining efficiency for automated problem solving by additional constraints between resource descriptions as presented in [9].

3.3 Integrating Teleological Level Mediation in WSMO

Completing teleological level mediation techniques, the following incorporates the outlined usage of Δ-relations into the WSMO mediators architecture in order to attain an integrated mediation model for Semantic Web services [20].

As shown in Figure 1, WSMO distinguishes four mediator types: *OO Mediators* that connect ontologies and provide data level mediation facilities. *GG, WG*, and *WW Mediators* connect goals and Web services. Each mediator connects source and target components denoted by the denomination prefix, and applies respective mediation techniques in order to resolve and handle the heterogeneities that can potentially arise between the source and target.

As mediation facilities, *GG, WG*, and *WW Mediators* can use OO Mediators for handling data level heterogeneities and may contain Δ-relations as the teleological level mediation definition. In addition, *WG Mediators* and *WW Mediators* can use a process mediator for resolving behavioral mismatches in communication or cooperation. We consider this mediation framework to be complete for Semantic Web services as it defines architectural components that apply appropriate mediation facilities for all heterogeneity types that can appear between the core elements of Semantic Web service systems.

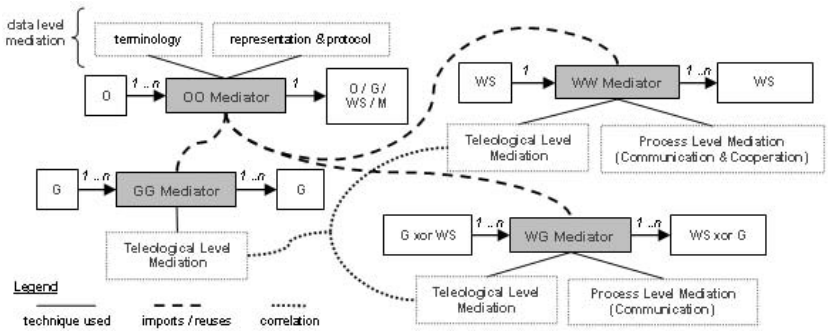


Fig. 1. WSMO Mediator Topology

There are two aspects to be mentioned for teleological level mediation in *GG Mediators, WG Mediators*, and *WW Mediators*. At first, we need to define the correlation between Δ and the source and target components of the mediator. As discussed above, WSMO capabilities are self-contained logical statements so that Δ-relations refer to complete capability. In consequence, we define the semantics of Δ-relations in WSMO mediators as follows:

- Δ defines the explicit logical relationship as the difference between the capabilities of the source S and the target T of a mediator $\mathcal{M}_{S,T}$ as defined in formula 1
- the Δ-situation is defined from the source component of a mediator to its target component (e.g. $plugin(S,T)$ denotes that $S \subset T$); as this information is needed for prosperous reasoning on teleological element ontologies, it is denoted in a non-functional type property of the Δ-relation definition
- in case of a proper generalization $plugin(S,T)$ or specialization $subsume(S,T)$, we can automatically attain $\mathcal{T}_{cap}$ from $\mathcal{S}_{cap}$ via Δ and vice versa:
  1. if $\mathcal{S}_{cap} \supset \mathcal{T}_{cap}$, then $\mathcal{T}_{cap} = \mathcal{S}_{cap} \backslash \Delta_{S,T}$ and $\mathcal{S}_{cap} = \mathcal{T}_{cap} \cup \Delta_{S,T}$
  2. if $\mathcal{S}_{cap} \subset \mathcal{T}_{cap}$, then $\mathcal{T}_{cap} = \mathcal{S}_{cap} \cup \Delta_{S,T}$ and $\mathcal{S}_{cap} = \mathcal{T}_{cap} \backslash \Delta_{S,T}$
  3. otherwise, T can not be attained directly from S via $\Delta_{S,T}$ or vice versa.

Secondly, there is a correlation between Δ-relations in *GG, WG Mediators* and *WW Mediators* denoted by the doted lines in Figure 1. In case the same goals and Web services are connected by respective mediators, we can derive new Δ-relations out of existing ones. For instance, referring to the introductory example of finding restaurants in Innsbruck, it holds that $\Delta_{R,WS_1} = \Delta_{R_O,R} \cup \Delta_{R_O,WS_1}$. Such correlations depend on the Δ-situations in the respective mediators and require further investigation that is out of the scope of this paper. However, this property allows learning mechanisms for Δ-relations for incrementally increasing the efficiency of Semantic Web service technology.

4 Evaluation by Example

This section demonstrates the usage and benefits of teleological level within exemplary scenarios in order to verify the above theoretical explorations. We first depict a simple scenario for improving the efficiency of Web service discovery along with exhibiting the modeling of Δ-relations in WSMO mediators, After that, we discuss more complex scenarios.

The following exemplifies the benefits of teleological level mediation for improving efficiency in Web service discovery as a core reasoning tasks within Semantic Web services. Discovery is concerned with detecting appropriate Web services for a given request or application scenario [14]. Therefore, we re-consider the introductory example of finding restaurant information in Innsbruck with the following goals and Web services involved:

1. G_O: a goal for finding a restaurant in Innsbruck
2. G_1: a goal for finding an Italian restaurants in Innsbruck
3. WS_1: a Web service 'Innsbruck Hotel and Restaurant Guide' that provides information on all hotels and restaurants in Innsbruck
4. WS_2: a Web service 'Tyrol Restaurant Guide' that covers restaurants in Tyrol (the state of Austria where Innsbruck is located in)
5. WS_3: a Web service 'Traditional Cuisine in Innsbruck' that provides information on Tyrolean restaurants in Innsbruck

Obviously, these goals and Web services are related to each other. For the goals, G_1 appears to be a specialization of G_0, and the Web services seem to be applicable for resolving these goals. If we would have no additional teleological information, we would have to create each goal separately and run a complete, most likely complex discovery process for determining which Web services can be used for resolving the goals. We assume the following situation for demonstrating how the efficiency improvement by teleological mediation: G_0 already exists, and we know from a previous discovery run that all Web services WS_1, WS_2, WS_3 can be used for resolving G_0. Now, a user wants to create G_1 and find usable Web services. Therefore, we define a teleological element ontology that consists of the goals, the Web services, GG and WG Mediators. Figure 2 shows this, including the Δ-situation and the direction of the mediators.

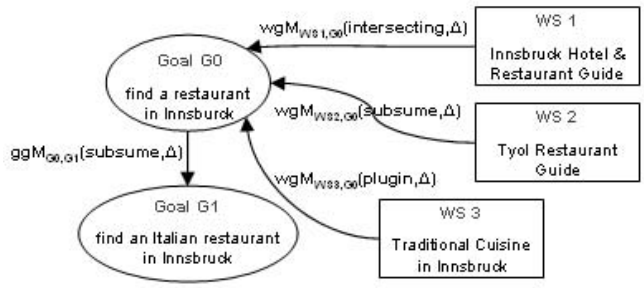


Fig. 2. Teleological Element Ontology Example

The following specifies the goals, Web services, and mediators as WSMO elements using the Web Service Modeling Language WSML [5], and exemplifies the modeling of Δ-relations. The listing below shows the ontology used as the terminology in this example, the goals G_0 and G_1, and the GG Mediator ggM_{G_0,G_1} (all elements are intended for academic demonstration purpose only and hence are very limited).

```
// Ontology used as terminology in example
namespace { _"http://www.wsmo.org/ontologies/mediate05example#",
  dc _"http:// purl .org/dc/elements/1.1#",
  loc _"http::// www.wsmo.org/ontologies/location#" }
ontology _"http://www.wsmo.org/ontologies/mediate05example"
 importsOntology _"http://www.wsmo.org/ontologies/location#"
 // an ontology for locations and addresses
 concept restaurant
    name ofType _string
    type ofType restaurantType
    address ofType loc#address
 // pre−defined instances
 austria memberOf loc#country
 tyrol memberOf loc#state
    name hasValue "Tyrol"
    inCountry hasValue austria
 innsbruck memberOf loc#city
```

```
    name hasValue "Innsbruck"
    inState hasValue tyrol

// Goal definitions (only postconditions modeled here)
namespace { _"http://www.wsmo.org/mediate05/G0#",
  o _"http://www.wsmo.org/ontologies/mediate05example#" }
goal _"http://www.wsmo.org/mediate05/G0"
// G0 – find a restaurant in Innsbruck
 importsOntology _"http://www.wsmo.org/mediate05/G0#"
 capability
  postcondition
    definedBy
     ?x[address.inCity hasValue = innsbruck] memberOf o#restaurant .

namespace { _"http://www.wsmo.org/mediate05/G1#",
  o _"http://www.wsmo.org/ontologies/mediate05example#" }
goal _"http://www.wsmo.org/mediate05/G1"
// G1 – find Italian restaurants in Innsbruck
 importsOntology _"http://www.wsmo.org/mediate05/G0#"
 capability
  postcondition
    definedBy
     ?x[type hasValue italian ,
        address.inCity hasValue innsbruck]
      memberOf o#restaurant .

// GG Mediator between G0 and G1
namespace { _"http://www.wsmo.org/mediate05/ggm#",
  o _"http://www.wsmo.org/ontologies/mediate05example#" }
ggMediator _"http://www.wsmo.org/mediate05/ggm"
 importsOntology _"http://www.wsmo.org/mediate05/G0#"
 source _"http://www.wsmo.org/mediate05/G0"
 target _"http://www.wsmo.org/mediate05/G1"
 Δ–relation
  nonFunctionalProperties
   dc#type hasValue subsume
  endNonFunctionalProperties
  definedBy
     ?x [type hasValue ?type,
     address.city hasValue innsbruck
     ] memberOf o#restaurant and
     not(?type = italian ).
```

The GG Mediator ggM_{G_0,G_1} has the source goal G_0 and the target goal G_1. The difference between them is that G_0 defines all restaurants as the desired information, while G_1 only desires Italian restaurants in Innsbruck. Hence, as $G_0 \supset G_1$, the Δ-relation is a subsumption from the source to the target goal.

The following shows the capabilities of the three Web services and the Δ-relations of the respective WG Mediators with a Web service as the source and G_0 as the target component. Each WG Mediator carries a Δ-relation that denotes the explicit difference between the capabilities of the respective Web service and G_O. The Δ-situation as defined in Figure 2 become obvious when considering the capabilities of G_O and the respective Web services. We here omit structural WSML definitions due to length limitations and only model Web service capability postconditions as this is sufficient for demonstration purpose.

```
// WS1 'Innsbruck Hotel and Restaurant Guide' capability postcondition
 capability
   postcondition
    ?x[address.inCity hasValue innsbruck] and
    (?x memberOf o#restaurant or ?x memberOf o#hotel).
```

```
// WS2 'Tyrol Restaurant Guide' capability postcondition
 capability
   postcondition
    ?x[address.inState hasValue tyrol] memberOf o#restaurant.

// WS3 'Traditional Cuisine in Innsbruck' capability postcondition
 capability
   postcondition
    ?x[class hasValue traditional, address.inCity hasValue innsbruck] memberOf o#restaurant.

// WG Mediator between WS1 and G0
wgMediator _"http://www.wsmo.org/mediate05/wgm1"
 source _"http://www.wsmo.org/mediate05/WS1"
 target _"http://www.wsmo.org/mediate05/G0"
 Δ−relation
  nonFunctionalProperties
   dc#type hasValue intersecting
  endNonFunctionalProperties
  definedBy
   ?x[address.inCity hasValue innsbruck] memberOf o#hotel.

// WG Mediator between WS2 and G0
wgMediator _"http://www.wsmo.org/mediate05/wgm2"
 source _"http://www.wsmo.org/mediate05/WS2"
 target _"http://www.wsmo.org/mediate05/G0"
 Δ−relation
  nonFunctionalProperties
   dc#type hasValue subsume
  endNonFunctionalProperties
  definedBy
   ?x[address.inCity hasValue ?city] memberOf o#restaurant and not(?city = innsbruck).

// WG Mediator between WS3 and G0
wgMediator _"http://www.wsmo.org/mediate05/wgm3"
 source _"http://www.wsmo.org/mediate05/WS3"
 target _"http://www.wsmo.org/mediate05/G0"
 Δ−relation
  nonFunctionalProperties
   dc#type hasValue plugin
  endNonFunctionalProperties
  definedBy
   ?x[type hasValue ?type] memberOf o#restaurant and not(?type = traditional).
```

Now we can discuss how the additional teleological mediation information can be beneficially utilized for gaining efficiency in reasoning tasks for Semantic Web services. As a major one, discovery is concerned with determining appropriate Web services for resolving a given goal. Therefore, semantic techniques are applied that determine logical relationship between functional service and goal descriptions in order to increase the accuracy of discovery results. As several aspects like valid pre-state and post-state detection need to be taken into account, adequate discovery engines for Semantic Web services consists of complex reasoning procedures (see [17], [13], [15], [24]).

For explaining how the need for such expensive discovery procedures can be omitted, we assume that all elements of the teleological element ontology shown in Figure 2 are given (i.e. all goals, Web services, and mediators). Also, we assume to know from a previous discovery run that WS_1, WS_2, WS_3 are usable for resolving G_O; it holds that only these or a subset can be usable for resolving G_1 because the subsume situation between the capabilities of G_O and G_1. For a Web service to be usable for the G_1 it has to satisfy its object definition that is strengthened, i.e narrowed in comparison to the one of G_O. This

can be determined via the Δ defined in the GG Mediator ggM_{G_O,G_1}. If a Web service that is in the discovery result of G_O satisfies Δ_{G_O,G_1}, then it is usable for resolving G_1 so that: $G_O \supset G_1 \wedge (usable(WS, G_1) \leftarrow usable(WS, G_O) \wedge satisfied(WS, \Delta_{G_O,G_1}))$. Evaluating this rule determines WS_1, WS_2 to be usable for resolving G_1 while WS_3 does not satisfy the Δ_{G_O,G_1}.

This example has discussed the simplest setting of efficient Web service discovery on basis of Δ-relations. However, the procedure can get more complex in case that a different Δ-relation exists between the source and target goal in a GG Mediator. While there can not exist any Web service that is usable for resolving G_y but not for G_x if the Δ-situation is $subsume(G_x, G_y)$, this can be the case for different Δ-situations. In such cases, the relationship between the Δ-relations in GG Mediators and those in WG Mediators needs to be taken into consideration. If there is a concatenation of subsumption Δ-relations between goals and Web services we do not even need to evaluate the Δ-relations as $subsume(WS, G_y) \leftarrow subsume(G_x, G_y) \wedge subsume(WS, G_x)$; on the other hand, in case of a concatenation of intersecting Δ-situations we possibly need to use a discoverer as the teleological mediation information are not sufficient for ensuring correctness of the discovery results. Hence, beneficial usage of Δ-relations for efficient discovery require more complex algorithms with respect to all possible combinations of Δ-situations that can occur in GG and WG Mediators. We do not discuss this any further as it exceeds the aim and scope of this paper.

We have demonstrated efficiency improvement for discovery as one main reasoning tasks for Semantic Web services. However, we can follow the same approach for improving efficiency within other mechanisms that are concerned with teleological level information like service composition. Therefore, we can define GG Mediators that establish a collection of sub-goals $G_{sub1}(G_x), G_{sub2}(G_x), ...$ for some complex goal G_x in the sense of a functional decomposition. If we do not discover any Web service that is capable of resolving G_x but some services for its sub-goals, we have determined the input required by a Web service composition engine for dynamically constructing a suitable execution model of the services usable for the sub-goals. Such application scenarios of teleological mediation need to deal with more complex relationships of Δ-relations that we consider to be future work.

5 Related Work

We are not aware of any other approach that identifies the need for mediation on the teleological level for Semantic Web services or provides support for this. Nevertheless, the following outlines work that has inspired the approach presented in this paper.

The need for efficient resource management has been revealed throughout our work on reasoning mechanisms for Semantic Web services with respect large-scale applicability and the performance problem of complex reasoning systems. Existing approaches like [11], [24] address this by defining classifications or architectural constraints as the basis for layered architectures that subsequently

narrow the search-space, i.e. reducing the number of elements that needs to be inspected in complex reasoning mechanisms. However, these techniques do not explicitly express the teleological relationship between resources and thus do not adequately support reasoning on additional teleological information in a way comparable to the one we have presented.

Our approach for teleological mediation has been inspired by the concept of refinement in the UPML framework for describing Problem Solving methods [10]. Therein, so-called *Refiners* define additional constraints referred to as Δ that bridge the teleological gap between goals or tasks to be achieved, the problem solving method that specifies the reasoning process, and the domain knowledge used for achieving the task [2]. This significantly decreases the required coverage for functionally describing a problem solving method as several aspects can are eliminated by Δs, hence allows gaining efficiency in the reasoning process for automatically resolving a goal [9]. This work has served as a basis for our definition of Δ-relations and their usage for improving efficiency in reasoning mechanisms for Semantic Web Services.

The idea of using information on the difference between resources for increasing efficiency in handling them is also applied in other technologies. For instance, video compression techniques like MPEG use so-called *delta frames* that only specify the changes between consecutive pictures; these are significantly smaller with respect to the amount of data required for specification and hence reduce the file size of videos [3]. This coincides with the approach of additional teleological information for reducing the reasoning effort for Semantic Web service techniques as we have presented here.

6 Conclusions and Future Work

This paper has introduced the teleological level as a novel aspect of mediation for Semantic Web services. This level deals with heterogeneities that arise between functional descriptions of Web services and related elements. We have defined Δ-relations that explicitly denote the teleological difference between functional element descriptions, integrated them into the mediation framework of the Web Service Modeling Ontology WSMO, and outlined how these additional information can be beneficially utilized for improving the efficiency of reasoning mechanisms for Semantic Web services.

Teleological level mediation as presented here is different from data and process level mediation. While the latter are concerned with techniques for establishing interoperability if this is not given a priori by resolving mismatches, teleological level mediation is concerned with improving the efficiency of Semantic Web service technologies. The elements that are connected via mediators in a teleological element ontology can reside in a functional manner without the additional teleological information. In the example on efficiency improvement for discovery we can accomplish the same correct discovery result by invoking a discovery engine instead of evaluating the Δ-relations. However, efficiency of core technologies for handling Semantic Web services is a crucial issue with re-

spect to large-scale, industrial strength applicability. As teleological mediation with Δ-relations can significantly improve efficiency, we consider this to be a beneficial mediation technique for Semantic Web services.

While this paper presents the foundation of teleological level mediation, future efforts will be concerned with integrating this technique into functional components for discovery and composition of Semantic Web services as well as elaboration of advanced algorithms for enhanced reasoning on teleological element ontologies. In a longer term, we will also consider techniques for automatically learning Δ-relations within Semantic Web service environments that enable dynamic improvement of a system's efficiency during its life time.

Acknowledgements

This material is based upon work funded by the EU under the DIP project (FP6 - 507483) and by the Science Foundation Ireland under Grant No. SFI/02/CE1/I131. The authors would like to thank the members of the WSMO working group (www.wsmo.org) and dedicate special thanks to Uwe Keller for fruitful advice and input to the presented work.

References

1. V. Alexiev, M. Breu, J. de Bruijn, D. Fensel, R. Lara, and H. Lausen. *Information Integration with Ontologies.* Wiley, West Sussex, UK, 2005.
2. V. R. Benjamin, D. Fensel, and R Straatman. Assumptions of Problem-Solving Methods and their Role in Knowledge Engineering. In *Proceedings of the European Conference on Artifical Inteliigence (ECAI 1996), Budapest, Hungary*, 1996.
3. A. C. Bovik (ed.). *Handbook of Image and Video Processing.* Academic Press, 2000.
4. E. Cimpian and A. Mocan. WSMX Process Mediation Based on Choreographies. In *Proceedings of the 1st International Workshop on Web Service Choreography and Orchestration for Business Process Management at the BPM 2005, Nancy, France*, 2005.
5. J. de Bruijn (ed.). The Web Service Modeling Language WSML. WSML Deliverable D16.1 final version 0.2, 2005. available from http://www.wsmo.org/TR/d16/d16.1/v0.2/.
6. J. B. Domingue, D. Roman, and M. Stollberg (eds.). Web Service Modeling Ontology (WSMO) - An Ontology for Semantic Web Services. Position Paper at the W3C Workshop on Frameworks for Semantics in Web Services, June 9-10, 2005, Innsbruck, Austria, 2005.
7. Lausen (ed.). Functional Description of Web Services. WSML Deliverable D28.1, 2005. Most recent version available at: http://www.wsmo.org/TR/d28/d28.1/.
8. D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2), 2002.
9. D. Fensel and R. Straatman. The Essence of Problem-Solving Methods: Making Assumptions to Gain Efficiency. *International Journal of Human-Computer Studies*, 48(2):181–215, 1998.

10. D. Fensel et al. The Unified Problem Solving Method Development Language UPML. *Knowledge and Information Systems Journal (KAIS)*, 5(1), 2003.
11. F. Giunchiglia, M. Yatskevich, and E. Giunchiglia. Efficient Semantic Matching. In *Proceedings of the 2nd European Semantic Web Conference (ESWC 2005), Crete, Greece*, 2005.
12. A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-Oriented Architecture. In *Proceedings of the International Conference on Web Service (ICWS 2005), Orlando, Florida*, 2005.
13. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic Location of Services. In *Proceedings of the 2nd European Semantic Web Conference (ESWC 2005), Crete, Greece*, 2005.
14. U. Keller, R. Lara, and A. Polleres (eds.). WSMO Web Service Discovery. Deliverable D5.1, 2004. available at: http://www.wsmo.org/TR/d5/d5.1/.
15. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *Proc. of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan, Nov. 2004*, 2004.
16. H. Lausen, A. Polleres, and D. Roman (eds.). Web Service Modeling Ontology (WSMO). W3C Member Submission 3 June 2005, 2005. online: http://www.w3.org/Submission/WSMO/.
17. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *Proceedings of the 12th International Conference on the World Wide Web, Budapest, Hungary*, 2003.
18. D. Martin (ed.). OWL-S: Semantic Markup for Web Services. W3C Member Submission 22 November 2004, 2004. online: http://www.w3.org/Submission/OWL-S.
19. S. McIlraith, T. Cao Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems, Special Issue on the Semantic Web*, 16(2):46–53, 2001.
20. A. Mocan, E. Cimpian, and M. Stollberg (eds.). WSMO Mediators. Deliverable D29, 2005. Most recent version available at: http://www.wsmo.org/TR/d29/.
21. A. Mocan (ed.). WSMX Data Mediation. WSMX Working Draft D13.3, 2005. available at: http://www.wsmo.org/TR/d13/d13.3/v0.2/.
22. M. Moran and A. Mocan. Towards Translating between XML and WSML based on mappings between XML Schema and an equivalent WSMO Ontology. In *Proceedings of the WIW 2005 Workshop on WSMO Implementations, Innsbruck, Austria*, 2005.
23. M. Paolucci, N. Srinivasan, and K. Sycara. Expressing WSMO Mediators in OWL-S. In *Proceedings of the workshop on Semantic Web Services: Preparing to Meet the World of Business Applications held at the 3rd International Semantic Web Conference (ISWC 2004) , Hiroshima, Japan*, 2004.
24. M. Stollberg, U. Keller, and D. Fensel. Partner and Service Discovery for Collaboration Establishment on the Semantic Web. In *Proceedings of the Third International Conference on Web Services, Orlando, Florida*, 2005.
25. G. Wiederhold. Mediators in the architecture of the future information systems. *Computer*, 25(3):38–49, 1994.

Mediation and Enterprise Service Bus
A position paper

Colombe Hérault, Gaël Thomas, and Philippe Lalanda

Université Joseph Fourier,
Laboratoire Logiciels Systèmes Réseaux, Équipe Adele
F-38041 Grenoble Cedex 9, France
firstname.name@imag.fr

Abstract. Enterprise Service Buses (ESB) are becoming standard to allow communication between Web Services. Different techniques and tools have been proposed to implement and to deploy mediators within ESBs. It turns out however that current solutions are very technology-oriented and beyond the scope of most programmers. In this position paper, we present an approach that clearly separates the specification of the mediation operations and their execution on an ESB. This work is made within the European-funded S4ALL project (Services For All).

1 Introduction

The integration of business activities is a long standing problem that has been tackled with different approaches (asynchronous middleware, Enterprise Integration application, etc.). The integration issue is perceived today through a new angle with the need to integrate distant applications available on the Internet. This raises challenging new problems related to communication over a public network, security and of course interoperability.

Service-oriented architectures constitute a very promising approach to integrate Internet applications: they actually provide the level of flexibility and scalability required to build industrial e-applications. However, service-oriented computing is today essentially technology-driven. Most available platforms focus on the technology allowing to publish and compose services and to make them communicate (i.e. SOAP [1], WSDL [2], UDDI [3], etc.). Different services may manipulate similar data or services under very different formats. The problem of data and interfaces heterogeneity is not directly addressed and left to the e-applications programmers.

As a remedy to this issue, several research areas are explored, including work about the semantic Web and ontologies[1] [4]. In this paper, we focus on an emerging middleware called Enterprise Service Bus (ESB [5] [6]). ESBs are providing technological solutions to intercept messages between Web Services and to translate or route them to help the integration of business applications.

[1] http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

In this paper, we argue that this technological solutions must come with models and tools allowing developers to describe the mediation operations at a higher level of abstraction. This work is carried out within the European ITEA project S4ALL (see www.itea-office.org), which aims at creating tools and models to build services using data from many heterogeneous sources from IT servers to industrial devices.

The paper is organized as it follows. The next section describes the notion of mediation and generalizes this notion to different non-functional behaviors. The section 3 presents a toy application as part of the European S4ALL project. The section 4 presents the global vision that we plan to implement. Finally, the section 5 concludes this paper.

2 From mediation to ESB

The concept of mediation is primarily an answer to the lack of interoperability between clients and data sources in Information Systems. Early mediation solutions evolved in order to also enhance the global quality of service provided by large scope of database systems. More recently the ESB (Enterprise Service Bus) concept has emerged in the context of B2B (Business to Business [7]). The ESB incorporates the concept of mediation to facilitate the design of application based on Webservices.

2.1 Mediation

Definitions As defined by G. Wiederhold in [8, 9], mediation is *"a layer of intelligent middleware services in information systems, linking data resources and application programs"*. the integration of various data resources (databases, Webservices or devices) and application programs (Webservices, enterprise applications, etc) raise a number of issues, essentially due to heterogeneity (see fig. 1).

The mediation layer is made of many mediators that are light weight components (e.g. independent black boxes that can be composed) and are composed to form mediation chains between client applications and data sources. G. Wiederhold defines a mediator as *"software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications. It should be small and simple, so that it can be maintained by one expert or, at most, a small and coherent group of experts"* [8].

Mediators capabilities Mediation includes tasks such as transformation and synthesis of data that can go from basic format translation in order to match a particular standard, to more sophisticated analysis using ontology or expert knowledge, adding new values to the data. Synthesis may be done over multiple data sources, having heterogeneous data type. For example, one may have to apply a currency conversion between services from different countries or may want to extract the global evolution of the activity of a plan over a week period

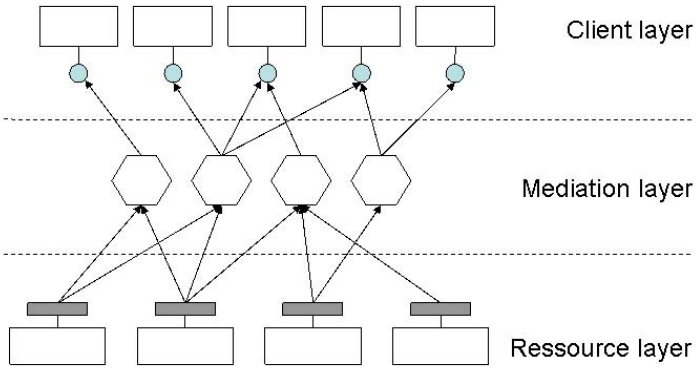


Fig. 1. Mediation layer

from a database containing daily information.
The aim of mediation is thus to abstract data and extract the domain knowledge [8] in order to ease the decision making by providing operations such as :

- selection upon different data;
- transformation from a data type to an other;
- integration of different data sources;
- selection of data source when there are many data sources available;
- resolution of inconsistent data.

Mediation added value The mediation code can be seen as code that is particular to a specific exchange between a client and a data source. Motivations to dissociate mediation code and business code are obviously to increase the separation of concerns and to decrease the coupling between client and data sources. Mediators are seen as elements which are not really part of the client, nor part of the data source but much more of the "binding" between them. Mediation improves:

- reusability : you may reuse a particular mediator in several mediation chains, for example an XML validation service.
- evolution of code : when a new client or a new data source appears, or even when there is a simple modification of one of them, it is not necessary to modify the code its interlocutors; only a new mediation element is added to the mediation chain.
- scalability : as mediation allows to integrate new clients and data sources progressively.

Using mediation concepts makes it easier to integrate code that enterprises do not want to modify, because of the cost (legacy code). It also avoids to product a client or a data source that would be able to interact with only one specific interlocutor.

Mediator patterns In the service based application life cycle, a new business arises : the mediator chain designer. Its task consists in finding appropriate mediators and detail their sequence. From [10] and [6], a basic mediator pattern list can be established :

- *examiners* modify the content of the request (e.g. validation, authentication, authorization, or monitoring);
- *transformation mediators* modify the content of the request (e.g. data type mapping and enhancement);
- *transcoder mediators* modify the format of the request but not its content. They allow requests to go through different transport protocol to interact;
- *cache mediators* stock results of already executed requests in order to save time and resources;
- *routers* chose the service they are giving the request to, depending on the content of the request. *Discovery mediators* do the same thing using besides a trader to dynamically chose the right service.
- *operator mediators* (e.g comparator, union, intersection, combination or aggregation);
- *clone mediators* dispatch a unique request to several services.

Related works Most interesting solutions in data mediation field are based on ontologies mapping. Indeed the heterogeneity of data leads to the need for semantic information. In order to deal with it, applications have been enhanced with ontologies. But it leads to the multiplication of ontologies in information systems regrouping many applications. Solutions such as WSMX[2] and TSIMMIS[3] [11], FOAM[4] or [12] provide abstractions and tools to generate mediators implementing the mappings between ontologies.

2.2 Mediation in ESB

Initially used for the integration of heterogeneous data store (databases, files, etc), the concept of mediation takes a new breath with SOA and Webservices. Mediation has thus become an essential part of ESBs (Enterprise Service Buses) (see fig. 2 inspired from [5] and IBM/SOA[5]).
An ESB is actually a middleware providing integration facilities built on top of industrial standards such as XML, SOAP, WSDL, WS-Addressing, WS-Policy, WS-Security and WS-ReliableMessaging, J2EE Connector Architecture [13]. Besides mediation functionalities, the ESB provides:

- a trading service in order to find appropriate services;
- communication service (mostly asynchronous with MOM and publish/subscribe);
- orchestration service (based on BPEL [14]).

[2] http://www.wsmo.org/TR/d13/d13.3/v0.2/
[3] http://www-db.stanford.edu/tsimmis/tsimmis.html
[4] http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/
[5] http://www-306.ibm.com/software/info1/websphere/index.jsp?tab=landings/esb

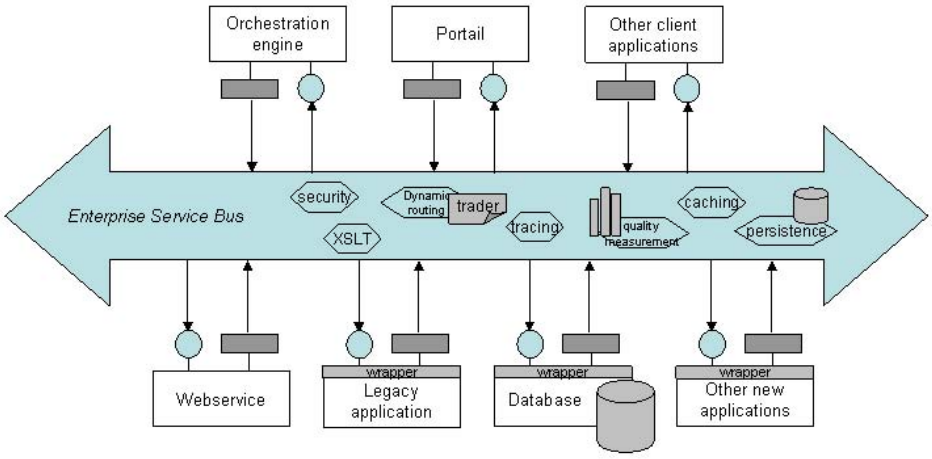


Fig. 2. Enterprise Service Bus

In ESB, the term mediation has a larger acception. In addition to transformation functionalities, mediation also includes :

- Security (e.g. cryptography, authorization, etc) which is a major preoccupation when different companies, using heterogeneous security systems need to interact.
- Dynamic routing and dispatch of requests potentially to multiple receivers in order to perform load balancing or to respond to failure of a data source for instance.
- Other non-functional actions related to QoS management such as incomplete data management, quality measurement, tracing, caching, or failure detection and recovery.

In the context of the ESBs, there are two different approaches to implement mediators. Mediators can be *ad hoc* pieces of code that intercept requests and process them. Mediators are dispatched over the network, some elements being closer to the client side (such as transformation to proprietary data format) other to data source side (such as synthesis, reducing network congestion). The second approach is to implement mediators as Webservices. Then, they are integrated to the service-based application as any other service. In any case, their requirements are to be available, reliable and easily maintainable. In the architecture, there are no constraints on whom the mediation components belong to or on their implementation.

2.3 limits of ESB

Despite these new approaches, there is no unique definition of ESB. ESB solutions are closer to commercial products packaging ad-hoc tools, than to a structured architectural layer. First ESBs were proprietary tools (Sonic ESB[6], Fiorano ESB[7], Cape Clear[8], PolarLake Integration Suite[9], etc). They were using proprietary solutions that were managed only at implementation level, such as the mediations in WebSphere [15]. Now these companies make an effort to capitalize there work into projects such as :

- projects directly working on mediation solution such as Apache Synapse. Synapse is a really recent project that emphasizes the role of mediation in SOA solutions;
- projects focusing on a larger scope that mediation such as new Open-source ESBs (e.g. Celtix[10] and Petals[11] from Objectweb, Mule[12] and ServiceMix[13]) from CodeHaus, OpenESB[14] from Sun). These projects provide code and tools (container, communication service, trader, etc) to build specific well-suited ESB. For the moment, they do not really focus on modeling the mediation. In the ObjectWeb consortium, researchers working on ESBs capitalize their results the ESB Initiative (ESBi[15]).
- a project that does not relate directly to mediation seems to overcome the ESB domain at the moment : the Sun Java Enterprise Service Bus API[16] (JBI). This API mostly defines a standardized container for services. It may have an impact on mediation because it also standardizes the exchanges between services (the sequence and format).

ESBs do not provide sufficient software engineering abstractions to give a high level comprehension of the architecture and the interactions of this layer. It focuses on development and administration and leads to lack of maintainability, whether it is a dynamic or a statical administration of the system.

3 Projet S4ALL

3.1 Project summary

The S4ALL project has been active since July 2005. It is funded by the European Community and brings together major industrial actors interested in delivering

[6] http://www.sonicsoftware.com/products/sonic_esb/
[7] http://www.fiorano.com/products/esb_key_for_bca.htm
[8] http://www.capeclear.com/products/ccESB4ws.shtml
[9] http://www.polarlake.com/en/html/resources/esb/
[10] http://celtix.objectweb.org/
[11] http://petals.objectweb.org/
[12] http://mule.codehaus.org/
[13] http://servicemix.org/
[14] https://open-esb.dev.java.net/
[15] https://wiki.objectweb.org/ESBi/
[16] http://www.jcp.org/aboutJava/communityprocess/edr/jsr208/

new services to their customers, including Alcatel, Nokia, Schneider Electric. The high level objectives of S4ALL are the following :

- To study the process of service creation, taking into account end-users, manufacturers and service providers requirements and to implement service creation and customization tools for different audience and supporting environments, that are the professionals, the end-users on PC and the end-users on mobile devices,
- To specify and implement the appropriate service execution infrastructure. In particular, the partners will focuse on the delivery of open sources OSGi platforms, a J2EE server and an Enterprise Service Bus.
- To demonstrate selected vertical applications in the telco and industrial fields illustrating all aspects developed within the project.

Applications that have been provided by the telco (Alcatel and Nokia) and by Schneider Electric (a world leader in power distribution) cannot be disclosed for the moment. In the following section we thus present a toy example (dealing with supermarkets and bakeries) that exhibits the main characteristics of the industrial applications studied in the project.

3.2 An example to illustrate the problem

The purpose of this section is to present a simple example and show how mediation techniques can be used in the Webservice context.
Figure 3 presents a simple scenario where a cybermarket provides a web service to buy different kinds of products, including bread. The cybermarket actually buy its bread from two different bakeries. Relationships between the cybermarket and the bakeries are implemented with Webservices. As it can be expected, data exchanged and interfaces used by these two bakeries aren't the same. In this example, the cybermarket wants to find the n least expensive bread sorts. Bakery 1 can directly answer to this question, but bakery 2 can only send the list of all its prices.

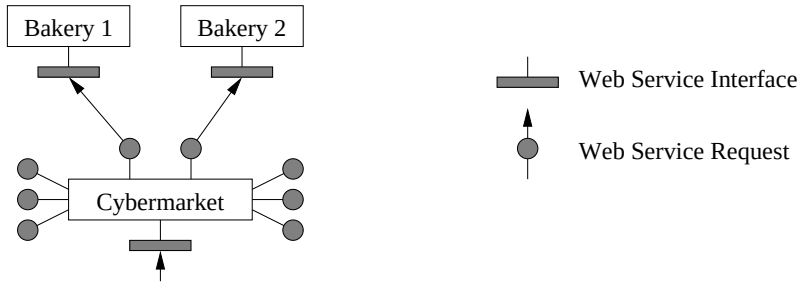


Fig. 3. A simple scenario

Using an ESB approach, the two bakeries are integrated using proxies that act as mediation elements. Two proxies are used in the cybermarket to encapsulate the calls to the bakeries and the cybermarket. With workflow languages, we can model our application but we can't separate the proxies from the application model: the proxy are inserted in the application model either like Web Services or by using language extensions.None of these two solutions is satisfying: by mixing at the same level the mediation and the process, an application designer can't understand the logical of the application (the model of the application) because there is no difference between the high level description of the application (what the application does) and the implementation of this description (how this application is implemented). The consequence of this lack of separation between the model and the implementation are principally:

- A lack of reusability of the application. Indeed, if a bakery server changes the designer should change the functional description of the application, although the model of the application doesn't change. In the same way, the mediation chain to access a bakery can't be reused in another application because it is mixed in the functional description of the application.
- A difficult design of the application because the application designer should simultaneously design the model of the application and the way it is implemented.

To improve these two aspects, our first goal is to provide a complete separation between the application model and the implementation view of this application. The second requirement of our work is to reuse the ESB tools : indeed, our purpose is not to re-engineer the already existing execution models, but the separation between these models and implementation aspects like mediation. By generalizing the notion of mediation to other non functional aspects of the application, like the quality of service, we are also improving the separation between the application model and its implementation.

3.3 An solution based on mediation

Figure 4 presents an architecture including mediation. A first mediator is used to aggregate the different bakeries. The aggregator gives the n lowest bread prices by mixing the data received from the two bakeries. Then, two mediators are used to hide the heterogeneity between the bakeries. Bakery 1 translator sends a request to find the n least expensive breads from bakery 1 to save bandwidth. Bakery 2 translator sends a request to find all the prices and gives only the n least expensive prices.

On the cybermarket side, a unique data format and a unique interface is then used to question the bakeries. By using mediation, the server (the bakeries) and the client (the cybermarket) don't have to be modified to incorporate new non-functional properties.

This solution based on mediation shows the complexity of the mediation chain. Indeed, we are only managing two bakeries and the chain remains simple.

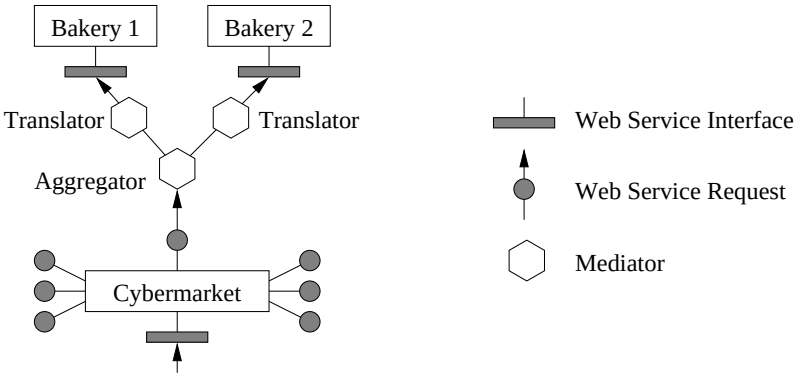


Fig. 4. A simple scenario with mediation

To really construct mediation chains, we should separate what does a mediation element from how it is connected: this separation between the chain and the implementation of mediation elements improve also the reusability. A mediation element can be reused in other application and we are separating the work of designing a mediation element from the work of composing these mediation elements. The architecture of the mediation chain also allows the insertion of other non functional element, like monitoring, security or verification elements: as explained before, these elements shouldn't appear in the application model, but in the mediation chain.

One of the possibility introduced by modeling the mediation chain between an executable model and the services that we plan to implement is the distribution of such mediation element in a network of hosts. Indeed, for load balancing, proximity or bandwidth reasons, it is more efficient to execute the mediation elements in a network of machine instead of using a single machine. A language to deploy the chain in a real network (a distributed ESB in our case) is thus necessary.

The last requirement of our mediation architecture is the dynamic adaptation of a mediation chain. Indeed, by separating the executable model of the application from its implementation, we doesn't have to stop the executable model to change how a service call is made. In our bakery scenario, dynamic adaptation allow, for example, the insertion of new bakeries to provide bread in the cybermarket without interrupting it. This possibility avoids the interruption of service time during a reconfiguration.

4 Our approach

4.1 A first experience

The purpose of this position paper is to promote the idea that it is important to specify the mediation operations in an abstract fashion, decoupled from the

execution environment (the ESBs in our case). To do so, our goal is to extend a mediation tool developed by the Scalagent company [16]. This tool allows to :

- describe mediation chains with an ADL (Architecture Definition languages) where mediation operations are performed by software components
- describe the execution environment and the way the mediation components have to be deployed on it
- automate the deployment and administration of the code installed on the network

This tool has been used successfully to develop (and industrialize) e-services in the domain of power distribution [17]. In this context, mediation is used between applications run on a J2EE infrastructure and OSGi-based gateways (see www.osgi.org). We are now exploring, in the S4All project, the way to extend this tool to deal with the connection of applications and Web services through ESBs and to automate the code generation and to allow dynamic reconfiguration.
The reminder of this section describes our new proposition. It presents the architectural and deployment views that we plan to model and implement.

4.2 Decoupling mediation

Figure 5 gives an overview of our model. First, we have defined a Platform Independent Model (PIM)[17] [18][19] in order to specify mediation chains independently of the ESBs and of the execution process: the model of the application and the implementation are separated, and the implementation of the mediation chain between the execution process and the services is modeled in a high level language. Our purpose is to increase understandability and reusability. This model includes the following elements:

- Service: any data source (Webservice, equipment, etc);
- Client: any application (Webservice or other) requesting a Service;
- Mediators: a component that is able to receive 1 to n pieces of data and send 1 to n pieces of data; it can be seen as a binding between Clients (1..n) and Services (1..n);
- Mediator component binding: link between Mediator components.

Through a dedicated tool, it is possible to assemble these elements to form mediation chains. The implementation of mediation elements themselves aren't modeled. Only the links between these elements are described in a high level language. A Mediation chain is close to the concept of "parternlink" in BPEL [14] but there are two main differences that come from the lack of integration of mediation in BPEL:

[17] http://www.omg.org/mda/

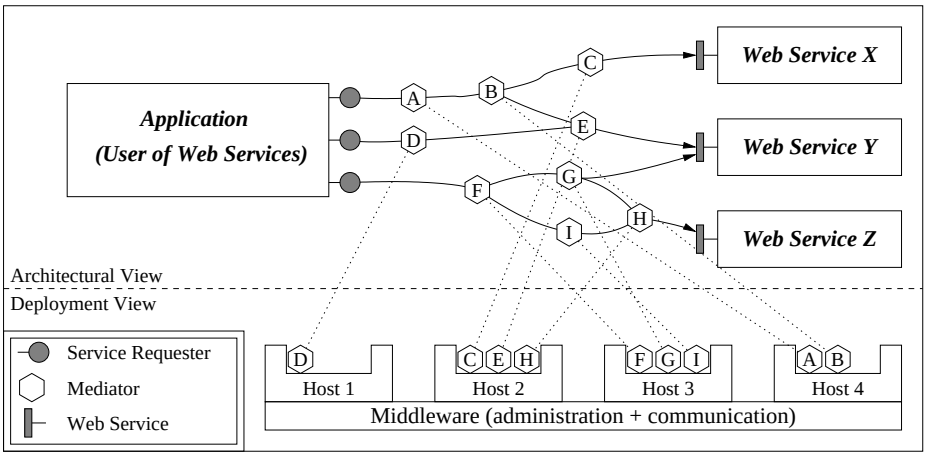


Fig. 5. Global architecture

- the data type is not necessarily the same on the client side of the link and on the server side because transformations are done on data;
- the Mediator chain is much more complex than the parterlink; it allows to describe processings done on data and the binding (e.g . synchronous, asynchronous, etc).

Another model is used to describe the execution environment, that is the ESB. In the current version of the mediation suite, the execution environment is provided by Scalagent under the form of an adapted version of the open source Joram middleware[18]. The challenge of the current project is to allow the execution of the mediation chains on an ESB. To meet the needs of the mediation suite, the ESB has to able to load dynamically new mediators and to take decision during a reconfiguration. This execution environment should also provide a middleware layer to interconnect the different execution environment. Figure 6 summarizes this execution environment. The middleware will be used to manage the monitoring, but also to interconnect the different mediators. It is important to note that the ESBs presented in the previous sections meet these technical requirements.
To automatically deploy and manage the set of mediators on ESBs, a specific language has to be developed. This language gives information about the location of mediators. This language will only describe where are physically located the mediators. This language has to remain very simple: it simply does the correspondence between a symbolic name of a mediator (used also in the ADL) and a host. We believe that is should be based on XML because. Through an extensible XML Schema, it allows to construct structured files to describe the mediation chain, that may be shared between the mediation actors, using a com-

[18] www.objectweb.org

mon vocabulary and grammar.
We also plan to extend ESBs functions in order to improve dynamism. To do

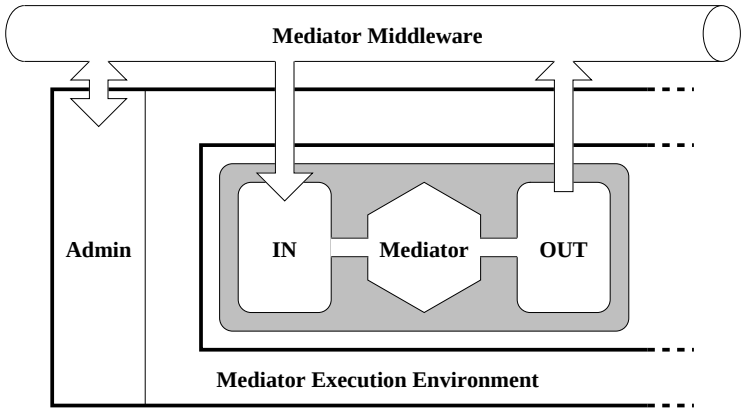


Fig. 6. The Mediation Middleware

so, we will use OSGi which provides a good solution to dynamically load and unload code. OSGi provides also a good solution to dynamically link the mediator with the execution environment. Inside the execution environment, a remotely administrable part should be planned. This part will communicate through the middleware.

5 Conclusion

Today's solutions for Web Service orchestration do not integrate heterogeneity of data type. At runtime, mediation operations have to be added to the system in order to ensure interoperability between Web Services. But mediation solutions are very technology-oriented. They do not provide abstractions that would facilitate the design and the administration of mediation chains. Dealing with distribution over heterogeneous networks and appearance of new Web Services becomes a hard task.
We propose then a higher abstraction model for mediation. Our model defines two levels: (i) a Platform Independant Model that provides an architectural view of the mediation chain. It defines the elements of the chain (Service, Client, Mediator component and Mediator component) that may be mapped automatically to the second level (ii) a framework that should provide a deployment language, a runtime environment and tools to dynamically administrate and reconfigure the platform by generating the communication code.
The architecture and the model presented in this paper meet the requirements of services integration: (i) the execution process is independent from the mediation

chain model to improve the reusability of the two elements, (ii) the mediation chain model separates the implementation of the mediation elements from their bindings, also to improve the reusability of this two elements, (iii) a mediation chain is projected on a distributed ESB to balance the load, (iv) a mediation chain can be updated during the execution of the execution process to avoid interruption of services during reconfigurations.
Our work builds on top of the mediation suite provided by the Scalagent company that has been used successfully in several device-oriented domain. The challenge is to replace the proprietary execution environment by ESBs (commercial or open source) while keeping the abstract, platform independent description of mediation chains. We are also exploring new ways to express business and mediation code. A promising approach is to use BPEL, that allows to describe Webs Services choreography, or another choreography language like APEL [20].
Another objective is to extend the ESBs execution capabilities in order to make them more dynamic regarding administration and configuration. To do so, we plan to integrate OSGi as an execution platform for the asynchronous middleware runtimes.

References

1. W3C. Simple object access protocol (soap) 1.2. Technical report, W3C, June 2003.
2. W3C. Web services description language (wsdl) 1.1. Technical report, March 2001.
3. OASIS. Uddi executive overview: Enabling service-oriented architecture. Technical report, OASIS, 2004.
4. T.R. Gruber. A translation approach to portable ontology specifications. *Academic Press*, 1993.
5. David A. Chappell. *Enterprise Service Bus*. O'reilly Media, 2004.
6. M.-T Schmidt, B. Hutchinson, P. Lambros, and R. Phippen. The enterprise service bus: Making service-oriented architecture real. *IBM System Journal*, 44(4):781, 2005.
7. C. Bussler. *B2B Integration*. Springer-Verlag, June 2003.
8. Gio Wiederhold. Mediators in the architecture of future information systems. In Michael N. Huhns and Munindar P. Singh, editors, *Readings in Agents*, pages 185–196, San Francisco, CA, USA, 1997. Morgan Kaufmann.
9. Gio Wiederhold and Michael Genesereth. The conceptual basis for mediation services. *IEEE Expert: Intelligent Systems and Their Applications*, 12(5):38–47, 1997.
10. John Todd, Christian Och, Roger King, Richard Osborne, Jr. William J. McIver, Nathan Getrich, and Brian Temple. Building mediators from components. In *DOA '99: Proceedings of the International Symposium on Distributed Objects and Applications*, page 352, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0182-6.
11. J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. Information translation, mediation, and mosaic-based browsing in the tsimmis system. In *Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data*, page 483, San Jose, California, June 1995.

12. F. Scharffe and J. de Bruijn. A language to specify mappings between ontologies. In *IEEE SITIS'05*, Yaoundé, Cameroon, November 27th - December 1st 2005.
13. Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall, March 2005.
14. Microsoft SAP AG Siebel Systems IBM BEA Systems. Business process execution language for web services version 1.1. Technical report, July 2002.
15. Rachel Reinitz and Andre Tost. Building an enterprise service bus with websphere application server v6 – part 4– http://www-128.ibm.com/developerworks/websphere/techjournal/0505_reinitz/0505_reinitz.html. IBM WebSphere Developer Technical Journal, May 2005.
16. Philippe Lalanda, Luc Bellissard, and Roland Balter. An asynchronous mediation suite to integrate business and operational processes. In *IEEE Internet Computing*, January-February 2006.
17. Philippe Lalanda. E-services infrastructure in power distribution. In *IEEE Internet Computing*, May-June 2005.
18. J. Miller and J. Mukerji. Model driven architecture (mda) omg tc document ormsc/2001-07-01. Technical report, Object Management Group (OMG), July 2001.
19. Frankel David. *Model Driven Architecture*. John Wiley and Sons Ltd, January 2004. ISBN 0471319201.
20. Jacky Estublier, S. Dami, and M. Amiour. Apel: a graphical yet executable formalism for process modeling. In *Automated Software Engineering, ASE journal*, volume 5, 1998.

Alignment infrastructure for ontology mediation and other applications

Jérôme Euzenat

INRIA Rhône-Alpes, Montbonnot, France,
Jerome.Euzenat@inrialpes.fr

Abstract. Web services are not the only application requiring ontology matching and mediation. Agent communication, peer-to-peer systems, etc. also need to find relationships between ontologies. However, they do not necessarily require the same kind of mediation as web services. In order to maximise the utility of the semantic web infrastructure, it seems reasonable to share the mediation services among these applications. To that extent we propose an infrastructure based on the reified notion of alignments and show how it can be used in these various cases.

1 Introduction

Like the web, the semantic web will have to be distributed and heterogeneous. Its main problem is the integration of the resources that compose it. For contributing solving this problem, data is expressed in the framework of ontologies. However, ontologies themselves can be heterogeneous and some work has to be done to achieve interoperability.

Web services and semantic web services suffer from the same problems. There is no reason to think that the resources required, provided and consumed by services are described in the same ontology by services. Requiring this would be a threat to innovation in the domain. In consequence, before comparing or composing services, it is necessary to find correspondences between the heterogeneous ontologies that are used for describing them.

Semantic interoperability can be grounded on ontology reconciliation: finding relationships between concepts belonging to different ontologies.

We call this process "ontology matching". The ontology matching problem may be described in one sentence: given two ontologies each describing a set of discrete entities (which can be classes, properties, rules, predicates, or even formulas), find the correspondences (e.g., equivalence or subsumption) holding between these entities. This set of correspondences is called an alignment. Since the terminology is not shared between everyone, here are the definitions of the various terms used in this paper:

matching is the task of comparing two ontologies and finding the relationships between them;

alignment is the result of the matching task: it is a set of correspondences;

correspondence the relation holding (or supposed to hold according to a particular matching algorithm or individual) between two entities of different ontologies. These entities can be as different as classes, individuals, properties or formulas. Some authors use the term "mapping" or "mapping rule" that will not be used here.

transformation a program that transforms an ontology from one ontology expression language to another;

translation a program that transforms formulas with regard to some ontology into formulas with regard to another ontology (translation can be implemented by a set of translation rules, an XSLT stylesheet or a more classical program).

bridge axioms are formulas in an ontology language that expresses the relations as assertions on the related entities. They are used when merging ontologies.

mediator in web services a mediator is a translation of an information stream, in query application it is a dual pair of translations that transforms the query from one ontology to another and that translate the answer back.

This has been identified in some frameworks as ontology mediation [4]. Ontology mediation consists of taking advantage of the relations between ontologies for composing services. The semantic web service concept depends on the availability of mediators anywhere and anytime.

So it seems that the solution is in sharing these alignments. We pretend that semantic web services should benefit from a wider infrastructure than ontology mediation tools built for semantic web services only. There are several reasons for this:

Each application can benefit from more algorithms Many different applications have needs similar to those of web services. It is thus appropriate to share the solutions to these problems. This is especially true as alignments are quite difficult to provide.

Each algorithm can be used in more applications Alignments can be used for different purposes and must be expressed in a more general way than web service mediators so that they could be used in other contexts.

Each individual alignment can be reused by different applications There is no magic algorithm for quickly providing a useful alignment. Once high quality alignments have been established – either automatically or manually –, it is very important to be able to store, share and reuse them.

We present in the next section various contexts in which aligning ontologies are necessary (§2) and what are the needs of these applications (§3). We then describe an alignment infrastructure for the semantic web (§4) that can be used, in particular, for ontology mediation in semantic web services.

2 Ontology heterogeneity problems on the semantic web

In this section we introduce a number of examples of semantic heterogeneity on the semantic web and the benefit they can have from some alignment services. This section can be skipped by those already convinced of the ubiquity of this problem and that alignments can be applied to solve it.

2.1 Editing

The first place where ontology heterogeneity can be found is while designing an application. If it makes heterogeneous resources interoperate, finding the correspondences

between ontologies in order to include some data transformation in the applications is an option. In this simple case, the application developer can find the correspondences by hand and design the corresponding transformations.

Some tools provide support for finding the correspondences, like Protégé through the Prompt suite of tools [15]. This is a first step towards the integration of mediators. However, if the alignments between widely accepted ontologies are required, there are chances that they will have to be found over and over again. An infrastructure capable of storing the alignments and/or transformations and of providing them on demand to other users would be useful.

Newer ontology development environments whose goal is to take into account, from the beginning, networked ontologies take this step. These systems provide together with the ontologies, the mediators for having their data interoperating, both ontologies and mediators being distributed on the web.

This first kind of application is simple because it is static: ontologies are encountered at design time and mediators can be built at that moment. However, in an evolving world, it is better to design adaptive applications that can dynamically take advantage of non expected resources and ontologies.

2.2 Semantic web services

Web services have clearly been designed for being independent and replaceable. So, web service processors are open to incorporate new services in their workflow. For that purpose, they must be able to compare the description of these services (in order to know if they can be used and for what) and to route the knowledge they process (in order to compose different services together).

However, in case of semantic web services, which can be described with regard to different ontologies, imposing a central common ontology does not seem realistic and would freeze the possibility of such services. So ontologies used in descriptions must be matched and mediators must be able to translate the output of one service in a suitable input for another service [4]. This task can be carried out by hand or by a programme, online or off-line.

It is thus necessary to generate and store these mediators. Although this can be done within some static web service workflows, dynamic composition of services requires a more open infrastructure in which the mediator is defined and involved at run time. We think that it is preferable to be able to take advantage of what can be provided by the environment. For that purpose, the environment must store this mediator or some independent representation from which a mediator can be generated on demand.

2.3 Meaning negotiation in multi agent systems

Agents communicate by exchanging messages. The content of these messages is expressed in some language, very often with regard to some ontology accessible to the agent. The FIPA Agent Communication Language [10, 9] makes provision for declaring within messages, the ontology in which their content is expressed.

In a society of independently developed agents, it may happen that agents using different ontologies have to communicate. Several proposals have been made to address

this situation [1, 17, 16] and we recently proposed ours [8]. Their goal is to assess the correspondences between the terms of the ontologies. However, these correspondences, once established (through negotiation or other means), are not stored and cannot be used in further dialogues or by other similar agents.

In order to share the benefits of the agreed alignments, it is useful to have an infrastructure able to find and deliver these alignments. To avoid having all agents embedding heavy matching methods, it is more convenient to have agents and services able to share stored alignments and matching capabilities. Moreover, having a specific representation of alignments enables agents to negotiate them through argumentation before using them [8].

2.4 Matching contexts in ambient computing

In ambient computing, applications take advantage of the environment for providing services to users. This environment changes, e.g., with regard to the user location, and applications must always keep track of the changing environment involving new devices and sensors. By doing so, they will provide context aware solutions. If one wants to design flexible and smart ambient computing applications, it is necessary to take advantage of ontologies of these various devices, sensors and their capabilities [5]. Like in web service descriptions, these ontologies will provide description of the devices (even abstract devices like a temperature service) and the way to interact with them.

Again, it is expected that device providers will develop different ontologies adapted to their products or will extend some standard ontologies. Moreover, since applications evolve in ever changing environments in which devices fail and new ones are added, there is no way to freeze once and for all the considered ontologies.

In order to properly operate ambient computing environments, applications have to be expressed in terms of generic features that are matched against the actual environment. This matching process can take advantage of ontology matching. Because the same devices will be met by similar applications, providing a service for reconciling the various ontologies and storing the result will help these application to share established alignments.

2.5 Peer to peer mediation

Peer-to-peer systems, when used for sharing discrete resources (files containing images, music, texts, etc.), are organised independently by each peer. Currently, their descriptions correspond to a hierarchy in which the files are stored. However, improving the search on these systems requires a finer description of items. Anyone wanting to share their pictures with their family wants to index them by the name of depicted people, the place where it is, the sights in the pictures, etc. This tagging scheme will benefit from using some ontological description (for retrieving the pictures of “one of your daughters on a horse” by opposition to “Jenny on Tornado”).

Individuals have no or little reasons to commit to the same ontology (like they do not do in current peer-to-peer systems) and they cannot be prevented from modifying and refining their current lightweight ontologies [3].

So the use of peer-to-peer systems consists of querying for information and resources to a network of peers, which are all described in autonomous ontologies; in this situation the queries (and sometimes the answers) must be translated from peer to peer.

This is up to the peer-to-peer systems (with the help of their users) to find the correspondences between ontologies that enable answering queries. This need was explicit in projects like Edutella [14].

For that purpose, it is useful to be able to propose similar ontologies to users if they feel like adopting some standard ontologies. It is also useful to provide them with alignments when they want to issue some query towards other peers and to keep track of these alignments in order not to bother users again and again. These alignments can be used in the reverse direction and shared among the peers.

2.6 Emergent semantics between users

In most of the mentioned applications, nodes have their own ontologies and share alignments between these ontologies. Because individuals and organizations can have different needs and different standpoints, it may be uncomfortable to commit to a single common ontology. However, the social action of constantly aligning ontologies can be the occasion to confront and revise these ontologies. So, in an even more dynamic way, users may want to establish more consensual ontologies from this confrontation [18].

There are several ways in which this can be helped by some alignment infrastructure:

- Producing alignments contributes to solve the gap between ontologies. The alignments are a basis from which some discussion can start (like agent protocol for arguing about correspondences, see § 2.3);
- Alignment algorithms are very often able to compute a distance between ontologies. This is useful when one wants to find the closest ontology.
- By building some network of ontologies and alignments in which the alignments measure the distance between ontologies, it is possible to find out the proximity between people and agents with the help of social network analysis techniques. They will provide indication that can help customising the query process, the matchmaking process between people and even the consensus building.

These results will help users and community consolidating their ontologies by first achieving consensus among the more similar representations. This can also be used for selecting the most central ontology (in social network analysis terms).

2.7 Safe composition infrastructure

Last, but not least, we proposed in [6] a transformation infrastructure for the semantic web. Its goal is to provide transformations between heterogeneous ontologies that guarantee the properties satisfied by the transformations. These properties can be semantic properties like model or consequence preservation, or the preservation of some types of formulas or that only some types of entities are aligned. An organisation can also certify the alignment.

In a web context, transformations, their properties and their proofs can be made available on the network. Knowing the semantics of languages and the proofs of the transformation enables the application of the "proof-carrying code" idea [13] to knowledge exchange. Consequently, the user can be sure that the result of the transformation is the transcription of the initial knowledge with the desired properties.

In applications like semantic web services, it is critical, before launching some complex workflow to know that it satisfies some properties. In particular, it is important to assess the level of correctness of mediators (i.e., that an entity is not translated into another entity that will lead the web service to perform inadequately): the correspondences between ontologies cannot be assumed correct. Proving them would be a must.

3 Towards a common solution

It is clear, from the above examples, that matching ontologies is a major issue on the semantic web. It is not circumscribed to semantic web services, but applies to any application that communicates through ontologies.

3.1 Global needs...

As heterogeneous ontologies are a global problem of the semantic web, this calls for an infrastructure able to help these different applications to deal with it. In such a way, the effort of interoperating ontologies does not need to be solved in each kind of use made of this semantic web.

Moreover, given the difficulty of the matching tasks, there are a few algorithms available and when good alignments are available, it would be very useful to share them among applications. Hence a global solution requires sharing among various instances of applications and not only within each kind of application (web services, agents, etc.).

Given that the task (finding correspondences between ontologies) is very basically the same, it seems possible to use the same tools for that purpose.

3.2 ...but heterogeneous needs

However, if all these applications require ontology matching, they require it for different purposes:

- edition requires the ability to transform some ontology in order to integrate it or to generate a set of bridge axioms that will help identify corresponding concepts (the transformations act at the ontological level);
- agent communication requires translators for messages sent from one agent to another (they act at the data level); similarly, semantic web services require one-way data translations for composing services;
- peer-to-peer systems and more generally query systems require bidirectional mediators able to translate queries (ontological level) and translate back answers (data level).

So, sharing between the applications mentioned here is difficult at the level of the particular reconciliation services they require. For these reasons, it is certainly more convenient to share the matching results themselves (i.e., the alignments) and to ask the infrastructure to be able to generate the appropriate mediators.

3.3 Requirements

This infrastructure should be able to store and retrieve alignments as well as providing them on the fly. We call it an infrastructure because it will be shared by the applications using ontologies on the semantic web. However, it may be seen as a directory or a service by web services, as an agent by agents, as a library in ambient computing applications, etc.

Services that are necessary in such an infrastructure are:

- The ability to store alignments, whether they are provided by automatic means or by hand;
- Their proper annotation in order for the clients to evaluate the opportunity to use one of them or to start from it (this starts with the information about the matching algorithms, the justifications for correspondences that can be used in agent argumentation, as well as properties of the alignment, see § 2.7);
- The ability to produce alignments on the fly through various algorithms that can be extended and parameterized;
- The ability to generate knowledge processors such as mediators, transformations, translators, rules as well as to process these processors if necessary;
- The possibility to find similar ontologies and to contact other such services in order to ask them for operations that the current service cannot provide by itself.

In addition, it is necessary that services be able to exchange between them the alignments they found and select them on various criteria.

4 Architecture proposal for an alignment infrastructure

We argue below that alignments are the necessary structure for supporting this infrastructure. We propose an infrastructure made of a network of services that can be invoked with some particular commands. These services will manipulate alignments through an embedded implementation of our Alignment API.

4.1 Alignments

[2] tried to provide some strict definition of the alignment structure so as to be able to use and reuse it in various situations. Given two ontologies O and O', alignments are made of a set of correspondences (called mappings when the relation is oriented) between pairs of (simple or complex) entities $\langle e, e' \rangle$ belonging to O and O' respectively.

A correspondence is described as a quadruple:

$$\langle e, e', R, n \rangle$$

where:

- e and e' are the entities (e.g., formulas, terms, classes, individuals) between which a relation is asserted by the correspondence;
- R is the relation holding between e and e', asserted by the correspondence. For instance, this relation can be a simple set-theoretic relation (applied to entities seen as sets or their interpretation seen as sets), a fuzzy relation, a probabilistic distribution over a complete set of relations, a similarity measure, etc.
- n is a degree of confidence in that correspondence (this degree does not refer to the relation R, it is rather a measure of the trust in the fact that the correspondence is appropriate – "I trust 70% the fact that the correspondence is correct/reliable/..." – and can be compared with the certainty measures provided by meteorological agencies). The trust degree can be computed in many ways, including users' feedback or log analysis.

So, the simplest kind of correspondence (level 0) is:

$$URI1 = URI2$$

while a more elaborate one could be:

$$URI1(x, y, z) \Leftarrow_{.85} URI2(x, w) \wedge URI3(z, concat(y, w))$$

The first one express the equivalence (=) of what is denoted by two URIs (with full confidence), while the second one is a Horn-clause expressing that if there exists a w such that $URI2(x, w)$ and $URI3(w, concat(y, z))$ is true in one ontology then $URI1(x, y, z)$ must be true in the other one (and the confidence is here quantified with a .85 degree).

As can be observed from these two examples, alignments in themselves are not tied to a particular language. But in order to use complex alignments like the second one, systems must be able to understand the language in which formulas and relations are expressed. This is supported through the definition of a particular subtype of alignment (the first example resorting to the level zero of alignment).

We claim that alignments are more intelligible than transformations: they only express correspondences between ontology entities, not the way they must be used. This can be the basis for studying their properties (moreover, these properties can also be inferred from the methods used for generating alignments).

In order to help developing applications based on alignments, we designed the Alignment API [7]. It has been developed in the aim of manipulating a standard alignment format for sharing among matching systems. But it provided the features required for sharing them more widely. The API is a JAVA description of tools for accessing alignments in the format presented above.

The Alignment API as been implemented on top of the OWL API (other implementations could be based on totally different languages). This implementation offers the following services:

- Storing, finding, and sharing alignments;
- Piping alignments algorithms (for improving an existing alignment);
- Manipulating (trimming and hardening) and combining (merging, composing) alignments;

- Generating "mediators" (transformations, axioms, rules in format such as XSLT, SWRL, OWL, C-OWL, WSML);
- Comparing alignments (like computing precision and recall or a symmetric distance with regard to a particular reference alignment).

The API also provides the ability to compose matching algorithms and manipulating alignments through programming. Part of the interface of the API is presented in Table 1. The API can be used for producing transformations, rules or bridge axioms independently from the algorithm that produced the alignment. Since its definition, several matching systems have been developed within this API (OLA, OMAP) and more of them are able to generate its format (FOAM, Prompt, Falcon, etc.).

4.2 Alignment services

Our architecture is based on *Alignment services*. These services are able to perform a number of alignment tasks and offer them to the other agents or services. These tasks are summarised in Table 1.

Service	Syntax
Finding a similar ontology	$O' \Leftarrow Match(O, T)$
Align two ontologies	$A' \Leftarrow Align(O, O', A, P)$
Thresholding	$A' \Leftarrow Threshold(A, V)$
Generating code	$P \Leftarrow Render(A, language)$
Translating a message	$m' \Leftarrow Translate(m, A)$
Storing alignment	$n \Leftarrow Store(A, O, O')$
Suppressing alignment	$Delete(n)$
Finding (stored) alignments	$\{n\} \Leftarrow Find(O, O')$
Retrieving alignment:	$\langle O, O', A\rangle \Leftarrow Retrieve(n)$

Table 1. Services provided by the alignment service and corresponding API primitives (O denotes an ontology, A an alignment, P parameters, n an index denoting an alignment, P a programme realising the alignment and T and m some expressions).

Most of these services correspond to what is provided by any implementation of the Alignment API. They are exposed to clients through various communication channels (FIPA ACL, SOAP messages) so that all clients can effectively share the infrastructure.

The alignments are indexed by ontology pairs and by surrogates allowing fast retrieving. To one surrogate corresponds only one alignment while for an ontology pair, there can be several such alignments. There is no constraint that the alignments are computed online or off-line (i.e., they are stored in the alignment store) or that they are processed by hand or automatically. This kind of information can however be stored together with the alignment in order for the client to be able to discriminate among them.

The main principle of the Alignment API is that it can always be extended. In particular, it is possible to add new matching algorithms and mediator generators that will be

accessible through the API. They will also be accessible through the alignment services. Services can thus be extended to new needs without breaking the infrastructure.

Moreover, the kind of annotations put on alignments is also extensible. So far, alignments contain information about:

- the kind of alignment it is (1:1 or n:m for instance);
- the algorithm that provided it (or if it has been provided by hand);
- the language level used in the alignment (level 0 for the first example, level 2Horn for the second one);
- the confidence in each correspondence.

Other valuable information that may be added to the alignment format are:

- the parameters passed to the generating algorithm;
- the properties satisfied by the correspondences (and their proof if necessary);
- the certificate from a issuing source;
- the limitations of the use of the alignment;
- the arguments in favour or against a correspondence [8].

4.3 End-user support

The main purpose of the alignment infrastructure is to be invoked by applications that use the alignments for themselves and generally hide the alignments away from users.

However, in some cases, it is necessary that users have access to these alignments and the alignment services. This is particularly the case in the editor application in which it may be useful to allow users to display and modify the alignments by hand. More generally, it is useful for any application in which alignments can be computed and reviewed off-line. Edition facilities must enable to provide high quality reviewed alignments to the infrastructure.

This is also useful in the emergent semantics application when it involves users, because inspecting alignments can help providing arguments for reaching consensus.

Again, using alignments by opposition to more operational transformation or mediators opens the opportunity to share the same tools all over the infrastructure (which does not exclude, in addition, to use some more specific editors). It will thus be useful to provide standard tools for ontology editing that can be shared among these various applications. The VisOn[1] tool developed by University of Montréal is such a tool that can be used for editing alignments in the Alignment API format. Other tools such as the WSML Mapping language editor [11] could be adapted.

4.4 Inter-service communication

Alignment services must be found on the semantic web. For that purpose they can be registered by service directories (e.g., UDDI for web services). Services or other agents should also be able to subscribe some particular results of interest by these services.

[1] http://www.iro.umontreal.ca/~owlola/visualization.html

These directories are useful for other web services, agents, peers to find the alignment services. They are even more useful for alignment services to basically outsource some of their tasks. In particular, it may happen that:

- they cannot render an alignment in a particular format;
- they cannot process a particular matching method;
- they cannot access a particular ontology;
- a particular alignment is already stored by another service.

In these events, the concerned alignment service will be able to call other alignment services. This is especially useful when the client is not happy with the alignments provided by the current service, it is then possible to either deliver alignments provided by other services or to redirect the client to these services.

Moreover, this opens the door to value-added alignment services which use the results of other services as a pre-processing for their own treatments or which aggregates the results of other services in order to deliver a better alignment.

Such an organisation takes full advantage of the goal assigned first to the Alignment API, namely, the ability to compose matching algorithms, here under the form of alignment services.

5 Example

We provide below an example of the use of this alignment infrastructure in the context of agent communication. All actors are agents which communicate through messages using a small set of FIPA Agent Communication Language message types. The communication rules obey a precise protocol that has been defined in [8].

The scenario presented here involves four agents: two agents a and b are communicating but agent a uses ontology O while agent b uses O'. b will call two alignment services c and d with c being a powerful aligner with a restricted access to b environment (it cannot access O') and d having a broader access. Lines beginning with "//" provide explanations for the dialogue moves.

// Agent a is looking for a book and asks agent b
a−query-ref(:ontology O
:language RDQL
:content "SELECT x WHERE x O:autobiography http://www.bertrandrussell.com"
:reply-with 1)→ b
// Agent b does not understand ontology O and asks service c to align it with O'
b−request(:content align(O,O',$\emptyset$,$\emptyset$) :reply-with 1)→ c
// Service c cannot reach ontology O'
b ←failure(:in-reply-to 1 :content unreachable(O'))−c
// Agent b asks d to find a similar ontology
b−request(:content find(O',m) :reply-with 2)→ d
O'' ⇐Match(O',T)
// Service d found O''
b ←inform(:in-reply-to 2, :content O'')−d

// Agent b asks service d to align O' with O''
$b-$request(:content is-align(O',O'') :reply-with 3)$\rightarrow d$
$s \Leftarrow$ Find(O',O'',$\emptyset$,$\emptyset$)
// Service d had already stored such an alignment and returns it
$b \leftarrow$inform(:in-reply-to 3, :content s')$-d$
// Agent b asks service c to align O with O''
$b-$request(:content align(O,O'',$\emptyset$,$\emptyset$) :reply-with 4)$\rightarrow c$
$A \Leftarrow$ Align(O,O'',$\emptyset$,$\emptyset$)
$s \Leftarrow$ Store(O,O'', A)
// Service c computes the alignment
$b \leftarrow$inform(:in-reply-to 4, :content s)$-c$
// Agent b asks service c to translate the message with the found alignment
$b-$request(:content translate(m, s) :reply-with 5)$\rightarrow c$
$\langle O, O'', A\rangle \Leftarrow$ Retrieve(s)
$m \Leftarrow$ Translate(m, A)
$b \leftarrow$inform(:in-reply-to 5
:content "SELECT x
WHERE x O':biography http://www.bertrandrussell.com.
x O':author http://www.bertrandrussell.com.")$-c$
// Agent b asks service d to translate the result with the O' to O'' alignment
$b-$request(:content translate(m', s') :reply-with 6)$\rightarrow d$
$\langle O', O'', A'\rangle \Leftarrow$ Retrieve(s')
$m'' \Leftarrow$ Translate(m', A')
$b \leftarrow$inform(:in-reply-to 6
:content "SELECT x
WHERE x rdf:type O':biografia.
x dc:subject http://www.bertrandrussell.com.
x dc:creator http://www.bertrandrussell.com.")$-d$
// The returned query is evaluated by agent b
QueryResult(m'') $\Rightarrow$ x=http://isbn.org/2-436-4428-1
// which returns the answer to agent a
$a \leftarrow$reply-ref(:content "x=http://isbn.org/2-436-4428-1" :in-reply-to 1)$-b$
// a is satisfied and wants to know the publisher of the book
$a-$request-ref(:content "http://isbn.org/2-436-4428-1 O:publisher x" :reply-with 2)$\rightarrow b$
// b had not recorded the alignment surrogate and asks it to c
$b-$request(:content align(O,O",$\emptyset$,$\emptyset$) :reply-with 7)$\rightarrow c$
// which only have to retrieve it in its store
$s \Leftarrow$ Find(O,O",$\emptyset$,$\emptyset$)
$b \leftarrow$inform(:content s :in-reply-to 7)$-c$
// b asks c for a program in order to translate the messages by itself
$b-$request(:content render(s, C-OWL) :reply-with 8)$\rightarrow c$
// but c cannot deliver this format
$b \leftarrow$failure(:content unsupported(C-OWL) :in-reply-to 8)$-c$
// so b ask for another one
$b-$request(:content render(s, XSLT) :reply-with 9)$\rightarrow c$

$\langle O, O'', A\rangle \Leftarrow$ Retrieve(s)
$P \Leftarrow$ Render(A, XSLT)
$b \leftarrow$inform(:content P :language XSLT :in-reply-to 9)$-c$
// which is delivered and used by b to translate the message
$m' \Leftarrow P(m)$
// The translation goes once again through d
$b-$request(:content translate(m', s') :reply-with 10)$\rightarrow d$
$\langle O', O'', A'\rangle \Leftarrow$ Retrieve(s')
$m'' \Leftarrow$ Translate(m', A')
$b \leftarrow$inform(:in-reply-to 10
:content "SELECT x
WHERE http://isbn.org/2-436-4428-1 dc:publisher x")$-d$
// and the query is processed by b
QueryResult(m'') $\Rightarrow$ "x=http://www.example.com/#Routledge"
// which returns the result
$a \leftarrow$reply-ref(:content "x=http://www.example.com/#Routledge" :in-reply-to 2)$-b$

We have not described the use of these services for semantic web services. However, any application for composing web services may use exactly the same services for generating mediators.

6 Related work

Most of the work on general organisation of alignments is tied to some kind of application (e.g., C-OWL for peer-to-peer applications, WSMX for web services, Edutella for emerging semantics). The work most similar to the one presented here is that on MAFRA [12]. MAFRA proposes an architecture for dealing with "semantic bridges" that offers many functions such as creation and storing of such bridges. The dimension that distinguishes both works is the insistence of MAFRA to have a transformation associated with bridges. Although the transformations can take very different forms, this prevents the use of the same alignment for different purposes: the very benefit of the proposed architecture.

7 Conclusion

In the semantic web, ontologies are used by different kinds of applications and they all suffer from ontology heterogeneity. Henceforth, we have considered the problem of generating mediators for semantic web services as a global problem for the semantic web rather than a specific web service problem. Doing so enables semantic web applications to share ontology alignments instead of developing concurrent pieces of infrastructure.

A common infrastructure have to rely on alignments instead of mediators because alignments can be used by any application. We proposed an alignment infrastructure

based on our Alignment API. The API already implements most of the functions required for the alignment service. It has been enhanced by an agent interaction protocol that enables communicating with it and we are developing a web service interface. The API has been used by various groups, in particular for plugging in their matching algorithms. This API is extensible and can deliver alignments for many different languages.

It is too early to consider the scalability of this approach. On the one hand, aligning on the fly and using many different ontologies seems a threat to scalability. On the other hand this may be regarded as a lightweight process with regard to the cost of some heavy standardisation process. We think that the factor that will help this alignment infrastructure to scale is its flexible nature.

Of course, the proposed architecture, beside the alignment format and the interaction primitives, is open. This means that any other implementation than the current alignment API can be provided as one service and interact with the other ones.

The same service can be shared by all in a natural way (for agent, the alignment service is another agent, for services, it is a web service). The fact that the same service is used throughout the semantic web multiplies the chances that required alignments are already available and prevents the waste of resources.

We hope to have been convincing that such an infrastructure is worthwhile and would contribute to the growth of the semantic web and semantic web services.

Acknowledgements: This work has been partly supported by the European network of excellence Knowledge Web (IST-2004-507482).

References

1. Sidney Bailin and Walt Truszkowski. Ontology negotiation: How agents can really get to know each other, 2002.
2. Paolo Bouquet, Jérôme Euzenat, Enrico Franconi, Luciano Serafini, Giorgos Stamou, and Sergio Tessaris. Specification of a common framework for characterizing alignment. deliverable D2.2.1, Knowledge web NoE, 2004.
3. Paolo Bouquet, Luciano Serafini, and Stefano Zanobini. Semantic coordination: A new approach and an application. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proceedings of the 2nd International Semantic Web Conference (ISWC)*, volume 2870 of *Lecture Notes in Computer Science*, pages 130–145, Sanibel Island (FL, USA), October 2003. Springer Verlag.
4. Chris Bussler, Dieter Fensel, and Alexander Mädche. A conceptual architecture for semantic web enabled web services. *SIGMOD Records*, 31(4):24–29, 2002.
5. Joelle Coutaz, James Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.
6. Jérôme Euzenat. An infrastructure for formally ensuring interoperability in a heterogeneous semantic web. In Isabel Cruz, Stefan Decker, Jérôme Euzenat, and Deborah McGuinness, editors, *The emerging semantic web*, pages 245–260. IOS press, Amsterdam (NL), 2002.
7. Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd international semantic web conference, Hiroshima (JP)*, pages 698–712, 2004.
8. Jérôme Euzenat, Loredana Laera, Valentina Tamma, and Alexandre Viollet. Negotiation/argumentation techniques among agents complying to different ontologies. deliverable D2.3.7, Knowledge web NoE, 2005.

9. FIPA ACL communicative act library specification. Technical report, FIPA, 2002. http://www.fipa.org/specs/fipa00037.
10. FIPA ACL message structure specification. Technical report, FIPA, 2002. http://www.fipa.org/specs/fipa00061.
11. Armin Haller, Emilia Cimpian, Adrian Mocan, Eyal Oren, and Chris Bussler. WSMX – a semantic service-oriented architecture. In *Proceedings International Conference on Web Services (ICWS 2005), Orlando (FL US)*, 2005.
12. Alexander Mädche, Boris Motik, Nuno Silva, and Raphael Volz. MAFRA – a mapping framework for distributed ontologies. In *Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW)*, pages 235–250, 2002.
13. George Necula and Peter Lee. Efficient representation and validation of proofs. In *Proceedings of the 13th symposium on "logic in computer science", Indianapolis (IN US)*, pages 93–104, 1998.
14. Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. EDUTELLA: A P2P networking infrastructure based on RDF. In *Proceedings WWW Conference, Hawaii (HA US)*, 2002.
15. Natasha Noy and Mark Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proc. 17th AAAI, Austin (TX US)*, pages 450–455, 2000. http://citeseer.nj.nec.com/528663.html.
16. Jun Wang and Les Gasser. Mutual online ontology alignment. In *AAMAS OAS workshop*, 2002.
17. F. Wiesman, N. Roos, and P. Vogt. Automatic ontology mapping for agent communication. Research memorandum, MERIT-Infonomics, Maastricht (NL), 2001.
18. Anna Zhdanova, Reto Krummenacher, Jan Henke, and Dieter Fensel. Community-driven ontology management: DERI case study. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, Compiegne (FR)*, 2005.

Mappings Creation Using a View Based Approach*

Adrian Mocan, Emilia Cimpian

Digital Enterprise Research Institute (DERI)
National University of Ireland, Galway, Ireland
adrian.mocan@deri.org, emilia.cimpian@deri.org

Abstract. Solving the heterogeneity problems between semantically enriched data can only be done by having accurate alignments between the underlying ontologies. To obtain 100% accuracy of these alignments the human user (i.e. the domain expert) has to be kept in the loop in order to validate the mappings part of these alignments. Since creating mappings between ontologies in a manual fashion can be an error prone and time consuming task, our aim is to provide semi-automatic mechanisms that reduce the human effort to simple validations and choices. Furthermore we propose a mechanism for transforming domain experts inputs placed in graphical interface in formal representations of the semantic relationships between ontologies. The domain expert can choose between different views, each of them displaying certain relationships and entities in the ontologies, used for generating different types of mappings. At each step suggestion algorithms propose possible solutions for creating new mappings.

1 Introduction

Ontology mappings have become a prerequisite in solving data heterogeneity problems in the context of Semantic Web and Semantic Web Services. Manual, semi-automatic or automatic approaches have as output a set of so called *mappings*, expression of the semantic relationships existing between the analyzed ontologies. Accordingly, the mappings might be 100% accurate (generated by manual and semi-automatic methods) or could have a lower accuracy (usually generated by automatic methods).

In this paper we present a semi-automatic way of deriving the semantic relationships existing between two ontologies (i.e. mappings) and expressing these relationships in a formalized form. We especially emphasize the step in the mapping process that transforms the domain expert inputs provided by using a graphical interface, in formal representation of mappings. By this, the human user is abstracted from the peculiarities of a specific formalism and they can fully focus on the problem to be solved. The proposed tool offers a set of features meant to reduce the human user efforts during the mapping process from a laborious and error prone task to simple choices and validations. This is done by including a set of mechanisms and strategies to guide the domain expert during the entire mapping process and to suggest potential relationships between the two ontologies. Different views on the ontologies to be mapped can be activated, each of

* This material is based upon works supported by the Science Foundation Ireland under Grant No. SFI/02/CE1/I131.

them focusing on certain ontology entities and their relationships. By switching views different types of mappings can be created using the same principles.

The data mediation prototype we propose is part of Web Service Execution Environment (WSMX) [7], a framework capable of dynamic discovery, selection, mediation and invocation of Semantic Web Services. WSMX has as conceptual model Web Service Modeling Ontology (WSMO) [3] and as a consequence, in our approach, the ontologies to mediate between are WSMO ontologies (i.e. ontologies that conform to WSMO conceptual model for ontologies).

This paper is structured as follows: Section 2 gives a short motivation and describes the context in which this mediator was developed; Section 3 presents the strategies and mechanisms behind the design-time tool we propose; Section 4 presents an example that shows how our approach can be applied and how the generated mapping rules look like; Section 5 briefly describes two of the existing frameworks that relate to our work and Section 6 concludes the paper and indicates some of the future work.

2 Motivation

In the Semantic Web Services context it is mandatory for the exchanged data to be semantically described by using ontologies. Furthermore, alignments between different ontologies used for modelling the same domain have to be provided in order to enable the inter-operation of various parties using these ontologies. Such an alignment has to contain a set of (bidirectional) mappings that can be applied on the input data (source data) to produce the corresponding data in terms of the ontologies used by the target party. If the interchanged data are part of a business process the necessity of 100% accuracy[1] of mappings is obvious. This leads us to the immediate consequence that the domain expert has to be kept in the loop to validate these alignments and to assure the 100% accuracy.

The ontology mapping tool developed in WSMX is able to offer support to the domain experts in their work, to reduce the amount of effort required and to create a formal representation of the semantic relationships captured between the source and target ontologies. The semantic relationships are expressed as mappings in the abstract mapping language proposed by [1] and stored in a persistent storage. These mappings are to be used during the run-time when the actual transformations of the interchanged data is performed. As the abstract mapping language doesn't associate any formal semantics to mappings, a grounding to a concrete language has to be provided. More details can be found in [5].

3 Mappings Creation - A View Based Approach

The mapping creation process represents one of the most important phases in a mediator system. It is a design time process and it is well known that in order to obtain

[1] By the *accuracy* we don't necessary understand in this work the correctness of the mappings. Therefore, we consider that the mappings are accurate if they match the human user inputs, i.e. domain experts inputs are 100% accurate.

high accuracy of the mappings the human user has to be present in this process. We believe that by offering a set of strategies and methodologies for creating these mappings, we can reduce this error prone and laborious process from a manual task to truly semiautomatic one.

In our view, the mapping process (i.e. the design time phase of the mediation process) basically requires three types of actions from the domain expert:

Browse the ontologies The domain expert has to discover the ontology elements they are interested in. This step involves different views on the input ontologies, and strategies for reducing the amount of information to be processed by the human user (e.g. contexts based browsing).

Identify the similarities This step involves the identification of semantic relationships between the entities that are of interest in the two ontologies. For doing this the human user can make use of the suggestions offered by a set of lexical and structural algorithms for determining semantic relationships.

Create the mappings This last step involves the capturing of the semantic relationships by mappings. This means that the domain expert has to take the proper actions in order to capture the semantic relationships in the mapping language statements or maybe in predefined mapping patterns [2].

We propose a way of tackling the existing gap between the identified semantic relationships of the two ontologies and the mapping language (in our case a logical language) statements that capture these relationships. Mapping patterns can fill this gap only partially, the necessary steps from the semantic relationships identified in graphical tools to these patterns remaining uncovered.

Our approach describes how the input ontologies can be browsed by using different views, how the same ontological entities can play different roles in different views and how certain algorithms can be applied on these roles. We identified a set of views that can play an important role in the mapping creation process: *PartOf* view, *InstanceOf* view, *RelatedBy* view. Each of them will be described in details in this paper.

3.1 Terminology and General Strategies

We identify a set of general strategies that can be applied no matter what views are used for browsing the ontologies. Before describing these strategies we have to define several notions that will be used from now on:

Views A View presents a subset of the ontology entities (e.g. concepts, attributes, relations, instances) and the relationships existing between them. The views can be seen as vertical subsets of the ontology – all the entities and relationships of a specific type (dictated by each particular view) from the ontology are taken in consideration. Usually the view used for browsing the source ontology (source view) and the view used for browsing the target ontology (target view) have the same type but there could be cases when different view types are used for source and target.

Roles In each of the views there is a predefined number of roles the ontology entities can have. In general, particular roles are fulfilled by different ontology entities in different views and in each of the strategies and algorithms described we refer to

roles rather than ontology entities. The roles that can be identified in a view are: *Compound Item*, *Primitive Item*, *Description Item*.

Compound Item A *Compound Item* contains at least one description associated with it. For example in the *PartOf* view a compound item would be a concept that has at least one attribute.

Primitive Item A *Primitive Item* doesn't have any description associated. For example in the *PartOf* view all data types play the role of a primitive item.

Description Item A *Description Item* links a Compound Item with other Compound or Primitive Items. By this we can define as *Successor* of a Description Item, the Compound or Primitive Item it links to (indicated by → in Figure 1).

Figure 1 presents an abstract representation of a view and the main elements it consists of.

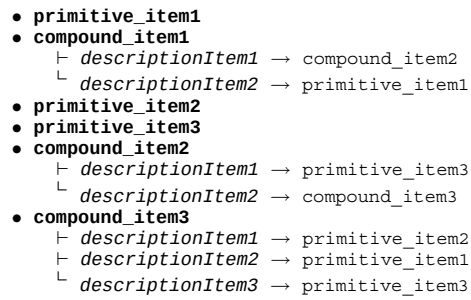


Fig. 1. Abstract View

All the algorithms and the strategies we propose are designed to be applicable to any view that meets these abstract specifications. By associating different ontological entities with the roles presented above, different views are obtained, the results being interpreted differently in respect with each particular view. One of the most important advantages of this approach is that these algorithms are immediately reflected in the graphical interface as unique mechanisms that ca be applied no matter of the view used.

Decomposition Algorithm The decomposition algorithm is one one of the most important algorithms in our approach and it is used to offer guidance to the domain expert in the mapping process and to compute the structural factor as part of the suggestions algorithms (described later on in this section). By decomposition we expose the descriptions of a compound item and make them available to the mapping process. That is, the decomposition algorithm can be applied on description items and it returns the description items (if any) for the successors of that particular description items. An overview of this algorithm is presented below: The implementation of *isCompound*, *getDescriptions*, *getSuccessor*, and *createLoop* differ from one view to another – for example, the cases when loops are encountered (i.e. the algorithm will not terminate) have to be addressed for each view in particular.

Table 1. Decomposition Algorithm

```
decompose(Collection collectionOfItems){
  Collection result;
  for each item in collectionOfItems do {
    if isCompound(item)
      Collection itemsDescriptions = getDescriptions(item);
        for each description in itemsDescriptions {
          Item successorItem = getSuccessor(description);
          if (not createLoop(succesorItem))
            result = result + successorItem;
        }
  }
  return result;
}
```

Mapping Contexts During the mediation process not all the information available in the ontology is of interest for each particular phase of the mapping process. A *mapping context* represents a subset of a view and presents only the relevant information for the current step of the mapping process. The notion of mapping context goes hand in hand with the decomposition algorithm as a mapping context is updated by applying this algorithm on a set of items. Thereby, by applying it recursively and updating the corresponding mapping context, the domain expert is guided through the mapping process until all the items from the initial context are mapped. A mapping context can be seen as a horizontal subset of an ontology[2].

Please note that when updating mapping contexts the input of the human user has to be taken in consideration: the domain expert has to choose the source and the target items on which the decomposition process has to be applied. Of course, this choice can be done in a semi-automatic manner, the system suggesting the most probable source-target combinations to be further explored. Depending on the results returned by applying the decomposition algorithm on the source and on the target items respectively, four situations might me encountered:

- *Both sides decomposition.* For both the source and the target items the decomposition algorithm returned a non empty set of items. As a consequence both the source and the target mapping contexts are updated.
- *One side decomposition - Source decomposition.* Only for the source items the decomposition returned a non empty set of items. This means that only the source mapping context is updated while the target mapping context remains unchanged.
- *One side decomposition - Target decomposition.* This is symmetric with the previous case. Only the target mapping context is updated, the source mapping context remaining unchanged.
- *No decomposition.* Successors were found neither for the source nor for the target, so no mapping contexts can be updated. Usually this ends the decomposition and the mapping process for the current branches in the source and target views.

[2] As the ontologies are browsed using views, the context becomes a horizontal subset of a view

Suggestion Algorithms The suggestion algorithms are used for helping the domain expert in taking decisions during the mapping process, regarding the possible semantic relationships between source and target items in the current mapping context. We propose a combination of two types of such algorithms: the first one being the lexical based algorithms while the second type being the structural algorithms that consider the description items in their computations.

As a result, for each pair of items we compute a so called *eligibility factor* (EF), which indicates the degree of similarity between the two items: the smallest value (0) means that the two items are completely different, while the greatest value (1) indicates that the two items are similar. For dealing with the values between 0 and 1 a threshold value is used: the values lower than this value indicate different items and values greater than this value indicate similar elements. Setting a lower threshold assures a greater number of suggestions, while a higher value for the threshold restricts the number of suggestion to a smaller subset. The EF is computed as an weighted average between a *structural factor* (SF), referring to the structural properties and a *lexical factor* (LF), referring to the lexical relationships determined for a given pair of items. The SF of two items is recursively determined by calculating the EF for their descriptions and for the successors of their descriptions. As mapped items have the EF equal with 1, is interesting to observe that the suggestions become more accurate the further we advance in the mapping process.

The weights can be chosen based on the characteristics of the ontologies to be mapped. For example when mapping between ontologies developed in different languages the weight of LF should be close to 0 in contrast with the case when mapping between ontology developed in the same working groups or institutions (the usage of similar names for related terms is more likely to happen) .

Even if the structural factor is computed using the decomposition algorithm, the actual heuristics used are dependent on the specific views where it is applied. In a similar manner the current views determine the weight for the structural and lexical factors as well as the exact features of the items to be used in computations.

Bottom-up vs Top-Down Approach Considering the algorithms and methods described above two possible approaches regarding ontology mapping can be differentiated: bottom-up and top-down approaches.

The bottom-up approach means that the mapping process starts with the mappings of the primitive items (if possible) and than continues with items having more and more complex descriptions. By this the pairs of primitive items act like a minimal, agreed upon set of mappings between the two ontologies, and starting from this minimal set more complex relationships could be gradually discovered. This approach is useful when a complete alignment of the two ontologies is desired.

The top-down approach implies that the mapping process starts directly with mappings of compound items and it is usually adopted when a concrete heterogeneity problem has to be resolved. That means that the domain expert is interested only in resolving particular items mismatches and not in fully aligning the input ontologies. The decomposition algorithm and the mapping contexts it updates will help the user to identify

all the relationships that can be captured by using a specific type of view and that are relevant to the problems to be solved.

In the same way as for the other algorithms, the applicability and advantages/disadvantages of each of these approaches depends on the type of view used.

Abstract Mapping Language The scope of the design-time environment presented in this paper is to produce formal representations of the semantic relationships identified/validated by the domain expert using a graphical tool. We chose to express these relationships as mappings in the language proposed in [1]. It is an abstract mapping language which does not commit to any existing ontology representation languages, thereby a formal semantic has to be associated with it and to ground it to a concrete language. Part of our work was to provide a grounding to Flora2[3] but for space reasons we are not discussing it in this paper (a full description can be found in [5]). From the same reasons we provide only a brief listing of some of the abstract mapping language statements:

- *classMapping* - By using this statement, mappings between classes in the source and the target ontologies are specified. Such a statement can be conditioned by class conditions (*attributeValueConditions*, *attribuiteTypeConditions*, *attributeOccurenceConditions*).
- *attributeMapping* - Specifies mappings between attributes in the source and target ontologies. This statements usually appears together with classMappings and can be conditioned by attribute conditions (*valueConditions*, *typeConditions*)
- *classAttributeMapping* - It specifies mappings between a class or an attribute (or the other way around) and it can be conditioned by both class conditions and attribute conditions.
- *instanceMapping* - It states a mapping between two individuals, one from the source and the other from the target.

In the next sections we illustrate how these mapping language statements are generated during design time by using our view based approach.

3.2 *PartOf* View

The *PartOf* is probably the most popular view on the ontologies to be aligned. The roles of *Primitive items* are taken by the *primitive concepts* (i.e. data types) while the roles of *Compound items* are taken by *concepts* described by at least one attribute. The *descriptions* are represented by *attributes* and naturally, the *successor* of a description is the *range* of that attribute. As shown in Figure 2 the successor of a description in this view (i.e. the range of an attribute) can be either a primitive concept or a compound concept.

Using this view we can create the following set of mappings:

Primitive Concept to Primitive Concept mapping. Such a mapping generates a *classMapping* statement in the abstract mapping language.

[3] Available at http://flora.sourceforge.net

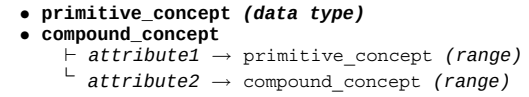


Fig. 2. Elements of *PartOf* View

Compound Concept to Compound Concept mapping. This mapping generates a *classMapping* statement in the abstract mapping language corresponding to the two compound concepts and triggers the decomposition mechanism, followed by a set of mappings between the attributes of these compound concepts, respectively. Such mappings between attributes are described below.

Attribute to Attribute mapping. There are four cases that can be encountered in this situations, generated by the two types of concepts an attribute can have as range: primitive concept or compound concept (i.e. primitive range or compound range).

- *Primitive range* on the source and *primitive range* on the target.
  An *attributeMapping* is generated in the abstract mapping language followed by a mapping between two primitive concepts.
- *Primitive range* on the source and *compound range* on the target.
  This case generates in the abstract mapping language a *classAttributeMapping* between the owner of the source attribute and the target attribute, followed by a mapping between two compound concepts: the owner of the source attribute and the range of the target attribute.
- *Compound range* on the source and *primitive range* on the target.
  This case is symmetric with the one presented above and it generates a *classAttributeMapping* in the abstract mapping language (actually this is an *attributeClassMapping* but there is only one statement in the language for both situations) and leads to a compound concept to compound concept mapping as well.
- *Compound range* on the source and *compound range* on the target.
  An *attributeMapping* is generated in the abstract mapping language followed by a mapping between the two compound concepts.

Primitive Concept to Compound Concept mapping. The *PartOf* view does not allow this type of mappings. Such mappings that might seem necessary initially, are covered by considering a compound concept from the source that has (or inherits) an attribute pointing to the primitive concept and mapping it with the compound target concept.

Compound Concept to Primitive Concept mapping. This is a situation similar to the one above and the *PartOf* view does not allow this type of mapping. The rational behind these restrictions is that such combinations would generate artificial mappings (with no semantics) between primitive concepts and all the compound concepts that refer, by means of their attributes, to these primitive concepts. For example, any compound concept that has an attribute with the range *String* could be mapped with *String*.

3.3 *InstanceOf* View

During the modeling process a set of instances can be used to properly capture some of the features of the domain. This is the case for enumeration sets, containing for example geographical locations, categories or even the allowed values for certain data-types (e.g. true and false for boolean). In the *PartOf* view these instances are not visible, however in the *InstanceOf* view the primitive items' role is taken by such instances (we call them *primitive instances*). By using these primitive instances more complex instances (*compound instances*) could be created, that is, instances of compound concepts whose attributes have ranges for which primitive or compound instances already exist or can be created. The compound instances play the role of compound items in this view (see Figure 3). The descriptions for the compound instances are represented by the attributes and attribute values corresponding to the compound instances. The attribute values are in fact the successors of compound items descriptions.

Additionally in this view we have to consider the rest of the concepts, for which no compound instances can be created. They might be either primitive concepts (they have no attributes at all) or compound concepts (none of their attributes has a range for which a primitive instance exists or a compound instance can be created) as identified in the *PartOf* view. They will play in this view the role of primitive items and all of them will be called *primitive concepts*.

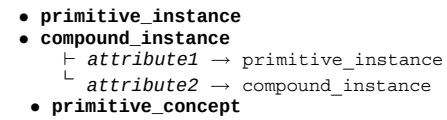


Fig. 3. Elements of *InstanceOf* View

InstanceOf view is used for creating conditional mappings and almost all the cases presented in the *PartOf* view occur in this view as well but with the difference that conditions are associated to mappings:

Primitive Instance to Primitive Instance mapping. This mapping generates an *instanceMapping* statement in the abstract mapping language.

Primitive Instance to Compound Instance mapping. Mappings between a primitive and a compound instance are not allowed in the *InstanceOf* view from similar reasons as for *Primitive Concept to Compound Concept mapping*in the *PartOf* view.

Compound Instance to Primitive Instance mapping. The same restriction applies as above.

Compound Instance to Compound Instance mapping. This mapping generates a *classMapping* statement in the abstract mapping language corresponding to the two compound concepts that are instantiated by the compound instances and triggers the decomposition mechanism, followed by a set of mappings between the attribute values of these compound instances, respectively. Such mappings between attributes' values are described below.

Attribute value to Attribute value mapping. There are four cases that can be encountered in this situation, generated by the two types of instances an attribute can have as value: primitive instance or compound instance (i.e. primitive instance range or compound instance range).

- *Primitive instance range* on source and *primitive instance range* on target.
  An *attributeMapping* is generated in the abstract mapping language conditioned by two *attributeValueConditions* - one for the source and the other one for the target attribute.
- *Primitive instance range* on source and *compound instance range* on target.
  This case generates in the abstract mapping language a *classAttributeMapping* between the owners of the source attribute and target attribute followed by a mapping between two compound instances: the owner of the source attribute and the range of the target attribute. In addition a *typeCondition* on the target attribute is applied.
- *Compound instance range* on source and *primitive instance range* on target.
  This case is symmetric with the one presented above and it generates a *classAttributeMapping* in the abstract mapping language and leads to a compound instance to compound instance mapping as well. In addition a *typeCondition* on the source attribute is applied.
- *Compound instance range* on source and *compound instance range* on target.
  An *attributeMapping* and two *typeConditions* one for the source and the other one for the target attribute, are generated in the abstract mapping language followed by a mapping between two compound instances.

In this view, no decomposition process can be applied on the primitive concepts and they can participate in the following types of mappings:

Primitive Instance to Primitive Concept mapping. This type of mapping is not allowed as the scope of primitive instances is to set conditions when mapping between other ontological entities. In this case, most probably a Compound Instance (created based on the initial primitive instance) with a Primitive Concept mapping should be considered.

Primitive Concept to Primitive Instance. The above restriction applies here as well.

Primitive Concept to Compound Instance mapping. This type of mapping will trigger decomposition on the right side and will generate a *classMapping* statement in the abstract mapping language. After decomposition, a set of mappings between the primitive concept and the attribute values of the compound instances can follow.

Primitive Concept to Attribute value mapping. Depending on the range of a particular attribute value (primitive or compound instance) we have two cases:

- *Primitive concept* on source and *primitive instance range* on target.
  An *attributeValueCondition* is added to the mapping between the primitive concept and the concept that owns the attribute (used in the above attribute value).
- *Primitive concept* on source and *compound instance range* on target.
  A *typeCondition* is added to the mapping between the primitive concept and the concept that owns the attribute (used in the above attribute value). After this *primitive concept to attribute value* mappings are to be considered.

3.4 *RelatedBy* View

Another interesting view to consider is the one where *attributes* play the roles of items. Each item has two fixed descriptions, one called *hasDomain* and the other called *hasRange*, having as successor the domain and the range of the attribute, respectively. Because of these two fixed descriptions, all the items are *compound items*. In addition, applying the decomposition algorithm to the descriptions and their successors will trigger a change of the view type. As the successors are concepts, by decomposition the view will switch from *RelatedBy* view (see Figure 4) to the *PartOf* view.

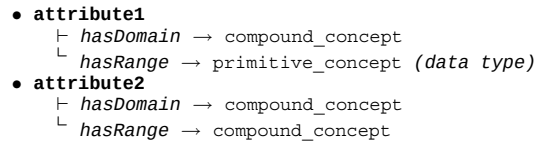


Fig. 4. Elements of *RelatedBy* View

We have two interesting types of mappings that can be done using this view: mappings between two *RelatedBy* views, and mappings between a *RelatedBy* view and a *PartOf* view (for the source and target respectively). For the first case we simply have:

– **(Compound) Attribute to (Compound) Attribute Mapping.** This is a classical mapping between two attributes, which will be followed after applying the decomposition algorithm by mappings between their domains and ranges. Such mappings can be done using the *PartOf* view as well, with only one difference: when created using the *RelatedBy* view, inverse attributes can be mapped, simply by mapping *hasDomain* from the source with the *hasRange* description from the target and *hasRange* from the source with *hasDomain* from the target. One has to keep in mind that the decomposition will trigger a view switching (from *RelatedBy* to *PartOf* view) so these mappings are affected by the restriction of the *PartOf* view (i.e. no mappings between primitive and compound concepts or vice versa are allowed). This type of mappings generates an *attributeMapping* and a pair of *classMappings* statements in mapping language.

When mapping between *RelatedBy* and *PartOf* view (and vice versa) we can have the following types of mappings:

– **(Compound) Attribute** (from *RelatedBy*) **to Compound Concept** (from *PartOf*). This mapping can be followed by any mappings between *hasDomain* and *hasRange* descriptions and descriptions of the compound concept from the target (i.e. attributes) as long as the mappings between successors conform to the restrictions on mappings in the *PartOf* view. A *classAttribute* mapping will be generated, followed by one or more pairs of *classAttributeMapping* and *classMapping*.
– **Compound Concept** (from *PartOf*) **to (Compound) Attribute** (from *RelatedBy*). This is a similar type of mapping as the one presented above. A *classAttribute* mapping is created, followed by one or more pairs of *classAttributeMapping* and *classMapping*.

In the next section we exemplify how a small ontology fragment is captured through the three types of views presented above.

4 Example

Let's consider the two fragments of ontology in Table 2 and try creating different types of mappings by using different types of views. For each pair of views we will present the abstract mappings generated from the graphical tool together with the grounding from this abstract representation to Flora-2.

Throughout the example we will represent the attributes in the following form in order to make the mappings self-explanatory:

```
[(attribute_owner) attribute_owner.attribute name => attribute_range]
```

The concepts will be referred simply by their names. The mapping process can

Table 2. Source and a target ontology fragments to be mapped

```
concept person
     name ofType xsd:string
     age ofType xsd:integer
     hasGender ofType gender
     hasChild ofType person
     marriedTo ofType person

concept gender
     range ofType xsd:string

instance male memberOf gender
     range hasValue "male"

instance female memberOf gender
     range hasValue "female"
```

```
concept human
     name ofType xsd:string
     age ofType xsd:integer
     noOfChildren ofType xsd:integer

concept marriage
     hasParticipant ofType human
     date ofType xsd:date

concept man subConceptOf human

concept women subConceptOf human
```

start by creating mappings using the *PartOf* view for both ontology fragments (see Figure 5). The mappings generate the abstract mapping language statements illustrated in Table 3. In order to use these mappings in concrete mediation scenarios a grounding mechanism is necessary to associate formal semantics to them. Table 4 shows how these mappings can be grounded to Flora-2. The dotted lines in Figure 5 shows mappings that are automatically created due to the inheritance relation existing between *human* and *man* and *human* and *woman* concepts.

Another set of mappings can be created by using *InstanceOf* views (see Figure 6). These mappings create a set of mapping rules and associate conditions to them, meaning that a particular mapping is valid only if the associated conditions hold.

Table 5 shows these mappings in the abstract mapping language while Table 6 contains the same mappings grounded to Flora-2.

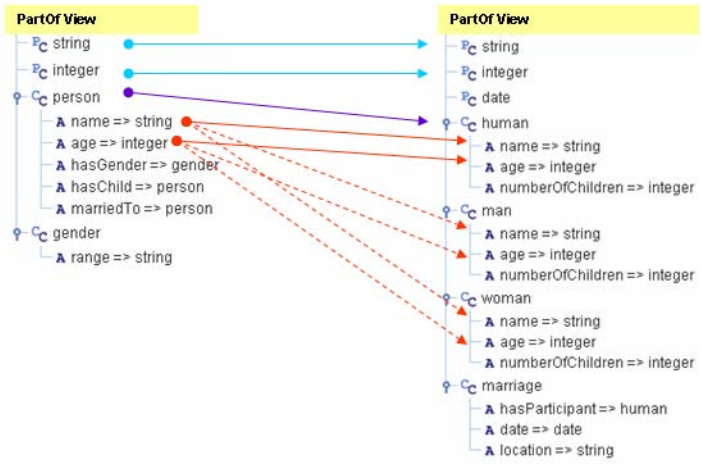


Fig. 5. Example of mappings between two *PartOf* Views

Table 3. Abstract mapping language statements generated by using *PartOf* views

```
Mapping(id000001
  classMapping( one-way
    string  string))
Mapping(id000002
  classMapping( one-way
    integer integer))
Mapping(id000003
  classMapping( one-way
    person  human))
Mapping(id000004
  attributeMapping( one-way
    [(person) person.name => string]    [(human) human.name => string]))
Mapping(id000005
  attributeMapping( one-way
    [(person) person.age => integer]    [(human) human.age => integer]))
```

Table 4. Flora-2 statements generated from the abstract mappings in Table 3

```
mediated(X_2, string):string :- X_2:string.
mediated(X_3, integer):integer :- X_3:integer.
mediated(X_4, human):human :- X_4:person.
mediated(X_5, human)[human.name -> Y_6] :- X_5[person.name -> Y_6].
mediated(X_7, human)[human.age -> Y_8] :- X_7[person.age -> Y_8].
```

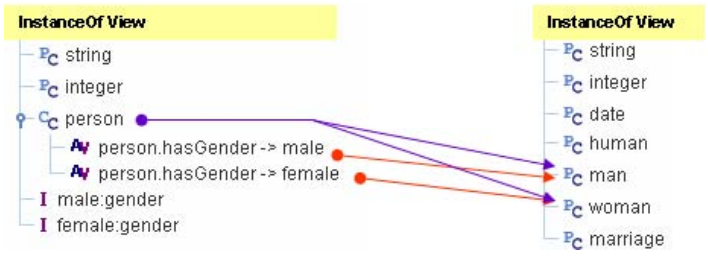


Fig. 6. Example of mappings between two *InstanceOf* Views

Table 5. Abstract mapping language statements generated by using *InstanceOf* views

```
Mapping(id000011
  classMapping( one-way
    person  man)
    attributeValueCondition(person [(person) person.hasGender => string] male)
Mapping(id000012
  classMapping( one-way
    person  women)
    attributeValueCondition(person [(person) person.hasGender => string] women)
```

Table 6. Flora-2 statements generated from the abstract mappings in Table 5

```
mediated(X_9, man):man :- X_9:person, X_9[hasGender -> male].
mediated(X_10, woman):woman :- X_10:person, X_10[hasGender -> female].
```

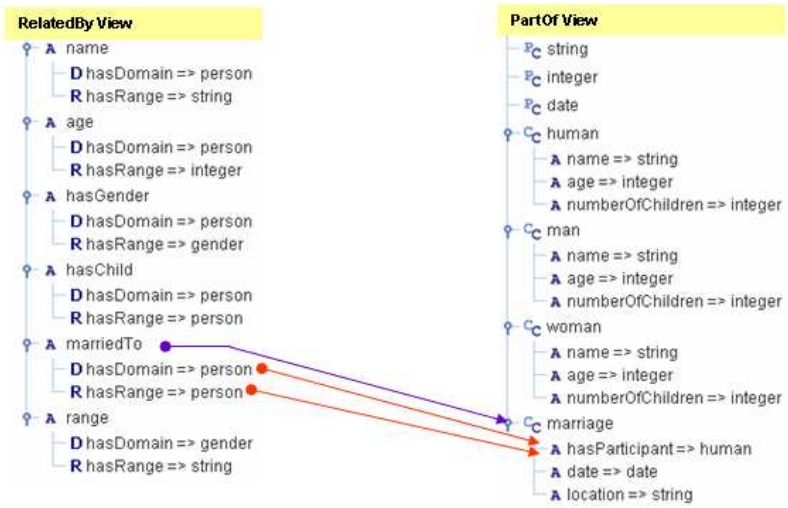


Fig. 7. Example of mappings between two *RelatedBy* Views

Another set of mappings can be created by using *RelatedBy* and *PartOf* views as depicted in Figure 7, for the *marriedTo* relation.

These mappings generate two one *classAttributeMapping* and one *attributeClassMapping* denoted in the abstract mapping language by the same statement, *classAttributeMapping* (see Table 7). Table 8 depicts the grounding of the abstract mappings in Flora-2.

Table 7. Abstract mapping language statements generated by using *RelatedBy* and *InstanceOf* view

```
Mapping(id000021
  classAttributeMapping( one-way
    [(person) person.marriedTo => person]  marriage))
Mapping(id000022
  classAttributeMapping( one-way
    person  [(marriage) marriage.hasParticipant => human]))
```

Table 8. Flora-2 statements generated from the abstract mappings in Table 7

```
mediated(X_25, marriage):marriage :- Y_26[person.marriedTo -> Z_27],
                                     mediated(X_25, human):human,
                                     Z_27:human, Y_26:person.
mediated(X_24, marriage)[marriage.hasParticipant -> mediated(Y_23, human)] :-
                                     Y_23:person,
                                     mediated(X_24, marriage):marriage.
```

5 Related Work

PROMPT is an algorithm and a tool proposed by Noy and Musen [6], which allows semi-automated ontology merging and alignment. It takes as inputs two ontologies and guides the user through an iterative process for obtaining a merged ontology as an output. This process starts with the identification of the classes with similar names and provides a list with initial matches. Then the following steps are repeated several times: the user selects an action (by choosing a suggestion or by editing the merged ontology directly) and the tool computes new suggestions and determines the eventual conflicts.

MAFRA [4] is a Mapping Framework for Distributed Ontologies, designed to offer support at all stages of the ontology mapping life-cycle. The framework is organized in two dimensions: it contains horizontal and vertical modules. The horizontal modules (*Lift & Normalization*, *Similarity*, *Semantic Bridging*, *Execution*, *Post-processing*) describe fundamental and distinct phases in the mapping process, while the vertical modules (*Evolution*, *Domain Constraints & Background Knowledge*, *Cooperative Consensus Building*, *Graphical User Interface*) run along the entire mapping process interacting with the horizontal modules.

Our approach offers a set of strategies that guides the domain expert through the whole mapping process. By maintaining proper mapping contexts and applying the decomposition process we can ensure that a complete set of mappings are derived for

that specific problem and give a precise meaning to user's actions. Further more, alternating different views on the ontologies to be mapped we can change the meaning of these actions without changing the mapping process. Each of the views we identified can be used to address different types of ontologies mismatches in a natural way and to abstract the human expert from the burdensome of the underlying mappings representation. And finally, we generate a representation of the identified semantic relationships as mappings in an abstract mapping language offering a great flexibility with respect to the mapping context in which these mappings are to be used.

The ideas presented in this paper have been implemented as part of WSMX Ontology Mapping Tool, delivered as plug-in in the Web Service Modeling Toolkit[4].

6 Conclusions and Future Work

Our approach proposes a methodology for mapping creation based on the inputs and validations of a domain expert. The resulting mappings have a 100% accuracy with respect to the domain expert inputs. They are represented as statements in an abstract mapping language, leaving open the possibility to create the most suitable grounding for a specific application.

Our future plans include refinements of the strategies and methodologies we introduced as: the identification of more views that can have a relevant role in mapping creations, improvements of the suggestion algorithms (including tuning mechanism) and of course enhancements of the design-time tool towards a more intuitive and user friendly graphical interface.

References

1. J. de Bruijn, D. Foxvog, and K. Zimmerman. Ontology mediation patterns. Technical report, SEKT Deliverable D4.3.1, 2004.
2. Jos de Bruijn, Douglas Foxvog, and Kerstin Zimmerman. Ontology mediation patterns library. SEKT Project Deliverable D4.3.1, Digital Enterprise Research Institute, University of Innsbruck, 2004.
3. C. Feier, A. Polleres, R. Dumitru, J. Domingue, M. Stollberg, and D. Fensel. Towards intelligent web services: The web service modeling ontology (WSMO). *International Conference on Intelligent Computing (ICIC)*, 2005.
4. A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra - a mapping framework for distributed ontologies. *Proceedings of the 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW)*, September 2002.
5. A. Mocan and E. Cimpian. WSMX Data Mediation. Technical report, WSMX Working Draft, `http://www.wsmo.org/TR/d13/d13.3/v0.2/`, March 2005.
6. D. Noy and M. Munsen. Prompt: Algorithm and tool for automated ontology merging and alignment. *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2000.
7. K. Verma, A. Mocan, M. Zaremba, A. Sheth, and J. A. Miller. Linking semantic web service efforts. *Second International Workshop on Semantic and Dynamic Web Processes (SDWP)*, 2005.

[4] Fore more information see `http://sourceforge.net/projects/wsmx/`

Lifting XML Schemas to Ontologies - The concept finder algorithm

Philipp Kunfermann, Christian Drumm

SAP Research Center CEC Karlsruhe
SAP AG
philipp.kunfermann@gmail.com
christian.drumm@sap.com

Abstract. In this paper we will present the Concept Finder algorithm. This algorithm is capable of creating mappings between the elements of a XML Schema and the concepts of an existing ontology. Furthermore we present results of a preliminary evaluation where real world schemas from the area of B2B communication were mapped to different ontologies using this algorithm.

1 Introduction

The data mediation problem in the context of web services is concerned with the transformation of a source message M_S which adheres to a source message schema S_S into a target message M_T that adheres to a target message schema S_T. To solve a given mediation problem a mapping needs to be created based on the source and the target message schema. In general the creation of such a mapping in very complex, making the task of developing such transformations very strenuous and error prone [1].

Semantic Web Service (SWS) are seen as the next evolutionary step after Web Service. SWS use ontologies to annotate the used data formats. Mappings between different message formats are in this context created on the semantic rather than on the syntactic level. In order to enable data mediation on the semantic level for existing web service and to bootstrap the semantic annotation of web services we will present an algorithm to relate XML schemas [2] to an existing ontology.

The remaining of the paper is organized as follows. First we briefly introduce the lifting problem. After that we present the algorithm we developed and explain its functionality using a simple example. Finally we present the result achieved by our approach when applied to real world schemas and ontologies.

2 Lifting

As mentioned in the introduction there are different problem areas in the context of SWSs that require the linking between purely syntactical data representation and ontologies. In the remaining of this paper we will call the process of relating syntactical data representations to ontologies *Lifting*.

Definition 1 (Lifting) *Lifting is the process of semantically annotating a source schema S_S with an ontology O.*

Note that this definition doesn't restrict the nature of the input schemas used for the lifting process. Furthermore based on the previous definition 3 different approaches to the lifting problem are possible (see Fig. 1):

- The most basic approach is to create a new ontology based on the source schema and use that ontology for the annotation
- If an existing ontology should be used for the annotation, a mapping between the source schema and the target ontology needs to be created
- Finally a combination of the first two approaches could be used by creating a new ontology from the source schema and by than mapping this ontology to an already existing one. For this approach existing ontology mapping techniques could be reused.

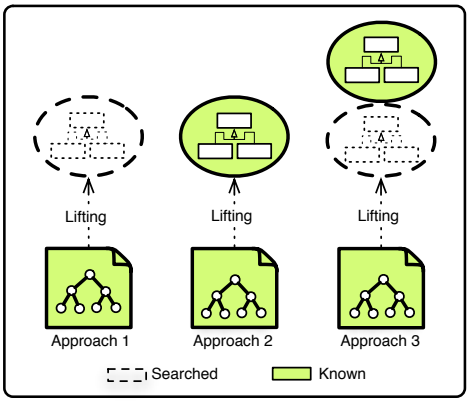


Fig. 1. Different possible approaches to the lifting problem.

In this paper we will present a solution for the second approach describe above. Our solution, the *Concept Finder* algorithm, is capable of identifying mappings between elements of S_S and concepts in O_T. We have chosen that approach for two reasons. Firstly we want to annotate existing WS in order to enable their usage in a SWS environment. For this purpose we need the possibility to easily relate the XML Schemas describing their in and output messages to the domain ontologies. Secondly our approach is capable of integrating different schemas by relating them to a given ontology. This simplifies the integration of existing services using the ontology as intermediate connector.

3 The Concept Finder Algorithm

For the discussion of the Concept Finder algorithm presented in this paper we have chosen to restrict the supported input data formats to XML for the source schemas S_S and to OWL [3] for the target ontology O_T. However the Concept Finder algorithm can easily be extended to use other input formats for both, S_S and O_T.

3.1 Overview

Figure 2 shows a high level overview the Concept Finder algorithm. Creating a mapping between a source schema and a target ontology consists of three steps:

1. In the first step the user needs to select the source schema S_S and the target ontology O_T. The Concept Finder algorithm parses the two files and creates an internal representation of them.
2. In the next step the user is presented with a graphical representation of both, S_S and O_T. The user now needs to select a *seed node* in S_S and a matching *seed concept* in O_T. This information is necessary in order to give the Concept Finder algorithm a starting point for exploring the ontology.
3. In the last step the algorithm computes a list of mappings between S_S and O_T. Details on how this is done will be given in the subsequent sections.

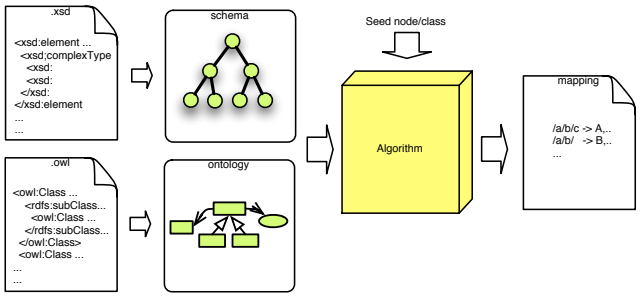


Fig. 2. High level overview of the Concept Finder algorithm

3.2 Details

Starting with the seed node and the seed concept, the algorithm compares the elements of the schema with relationships in the ontology. In order to do so, it navigates through the schema and the ontology in different ways. The structure

of the schema is traversed depth first while the ontology is traversed based on the matches that were found with the compared schema elements.

The seed node and seed concept are a schema respectively ontology element that have to be semantically corresponding. They represent the first match of our algorithm and define the context in which the algorithm operates in the schema as well as in the ontology. In order to find the next match, the Concept Finder algorithm compares every child of the seed node[1] with the relationships of the last matching concept (in the first step this is the seed concept). Firstly, all subclasses of the seed class (find subclass match) are compared. If no match is found, the algorithm compares all properties of the seed class (find property match). If still no match is found for the current element, the child nodes of the current element are explored in order test if a match can be derived (derived match).

A *similarity test* is used to determine if an elements of the S_S and an element of the O_T are matching based on their names. It is processed in three steps. In the first step a normalized levenshtein distance is calculated between the two names and if it is above a given threshold parameter, the elements are accepted as matching. If this is not the case, the name of the schema element is in a second step levenshtein compared with synonyms of the ontology element. In order to find these synonyms *WordNet*[4] is used as an external knowledge source. If still no match is found, the names of the compared elements are tokenized in a third step. These tokens are compared as in the first step and a new metric is calculated based on the number of existing and matching tokens. Again a threshold is applied in order to determine if two elements are matching. If the elements do not match the algorithm passes to the next schema element that will be compared with the relationships in the ontology.

After investigating possible mismatches between schema and ontology models, we discovered that the most important mismatches originate from the developer's freedom on how and what he is modeling using a concrete formalism. Because of the importance of these human caused mismatches, similarity measures have to be concentrated on features that are intuitive for human beings. While humans do hardly agree on the data-type, restrictions or even how many relationships are relevant for a certain concept, names contain important semantics, understandable to every person speaking the same language. Even though the developer may use different terminology, he will use a name that specifies the semantic of what he wants to represent. By using synonyms, we test the elements against a wide range of semantically identical concepts and therefore find correct match, if existing, with high probability. Nevertheless, from natural language processing (NLP) it is known that single words may cause ambiguities. This is prevented by looking at the context of the word. While NLP tries to interpret the sentence in which a word is presented, the context of the names in our case is found in the structure of the used models (e.g. branch of the

[1] In the following we will call the element that is compared by the algorithm current element.

schema tree). Therefore, Concept Finder navigates through the two models and compares only elements that are in the same context.

For every current schema element the algorithm does first try to *find a subclass match*. Therefore all subclasses - not only the direct ones - of the last matched concept are used. This last matched concept semantically corresponds to the parent element of the current schema element. If the current element is matching one of the compared subclasses, the relationship with the parent element is identified as an inheritance relationship. If no such match is found, the relationships of the last matching concept will be compared.

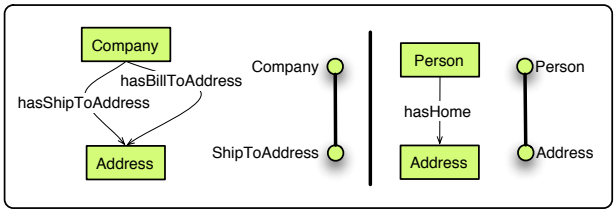


Fig. 3. The object property conflict.

Trying to *find property matches*, the Concept Finder algorithm compares all relationships, data-type properties and object properties, of the last matching concept with the current element. This comparison is more difficult for object properties than for datatype properties, as with object properties both, the relationship itself as well as the range concept, have a semantics carrying name and therefore both of them have to be taken into consideration for the comparison. Figure 3 illustrates that if only the range would be tested, the similarity for $(p_{hasShipToAddress}, c_{Address})$[2] and $(p_{hasBillToAddress}, c_{Address})$ would be the same with respect to the used similarity test while only the first match is semantically correct. If in the opposite case only the property would be tested, $(p_{hasHome}, c_{Address})$ would not be accepted as a match even though it correspond perfectly. In order to prevent that, the similarity test compares both and calculates the similarity as a mixed and weighted metric.

If no property match was detected, the algorithm tries to capitalize on the knowledge of inheritance contained in the ontology and *derive a match* by continuing to explore the children of the current element. They are still compared with the relationships of the last matching concept, knowing that subclasses inherit all relationships of its parents. If a certain number children of the current

[2] When describing the algorithm using examples we will use the following notation: c_{name} and p_{name} denote a concept or a property in the ontology with a given name; e_{name} denotes a element in the schema with a given name.

element match with relationships of the last matching concept, the probability for the current element to be corresponding to a subclass is high.

The algorithm navigates in this way through the whole schema tree while it visits only the relevant concepts of the ontology matching with the schema elements. This context based navigation makes sense because of the ontologies nature. As ontologies are used to define concepts and not only to structure information, we can suppose that if a relationship between two concepts exists in the real world, they are also represented in the ontology and therefore its context is explicitly defined. As in the ontology development no general rules exist on how elaborated a ontology has to be, the algorithm requires to have a meaningful and sufficiently elaborated ontology as input.

4 Example

In order to illustrate how the algorithm works in more detail, we will apply it to the schema and ontology excerpt shown in figure 4 and describe the different execution steps that match these two.

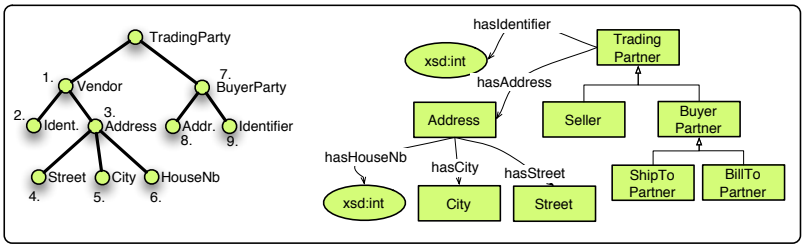


Fig. 4. Applying the algorithm.

- Starting with the first match chosen by the user (seed node and seed class) $e_{TradingParty} \rightarrow c_{TradingPartner}$, $e_{\boldsymbol{TradingParty}}$'s children, e_{Vendor} and $e_{BuyerParty}$ are explored in order to match them to the ontology[3].
- In the first step a match the algorithm tries to find a match for $e_{\boldsymbol{Vendor}}$. Comparing the subclasses and properties of $c_{TradingPartner}$ with e_{Vendor} would not result in any match[4].

[3] Figure 5 demonstrates the algorithm's execution. The current seed elements are shown in blue/dark-grey, the ones that were already successfully compared are shown squared and the unsuccessfully compared ones are show yellow/light-grey.

[4] This is the case if WordNet is not used or does not contain "Vendor" as a synonym for "Seller". We will assume this match is not found for illustrating the further execution steps

– Therefore,Concept Finder starts to explore the e_{Vendor}'s child $e_{\boldsymbol{Identifier}}$. No subclass match is found but a correspondence with the trading partner's identifier $e_{Identifier} \rightarrow p_{hasIdentifier}$.
– Because of the match $e_{\boldsymbol{Address}} \rightarrow (p_{hasAddress}, c_{Address})$, the exploration continues to compare the children of $e_{Address}$ with the range class of the match ($c_{Address}$).
– $e_{\boldsymbol{Street}} \rightarrow (p_{hasStreet}, c_{Street})$, $e_{\boldsymbol{City}} \rightarrow (p_{hasCity}, c_{City})$ and $e_{\boldsymbol{HouseNb}} \rightarrow p_{hasHouseNb}$ are found but cannot be further explored since they are leaf nodes.
– The algorithm selects e_{Vendor} and based on the matches found for its child node he derives that e_{Vendor} could correspond to a subclass of $c_{TradingPartner}$. A match $e_{Vendor} \rightarrow c_{TradingPartner}$ is stored with the special remark that this is a derived subclass match[5].
– Now the algorithm selects $e_{\boldsymbol{BuyerParty}}$,the next child node of $e_{TradingPartner}$ where the match $e_{BuyerParty} \rightarrow c_{BuyerPartner}$ is stored.

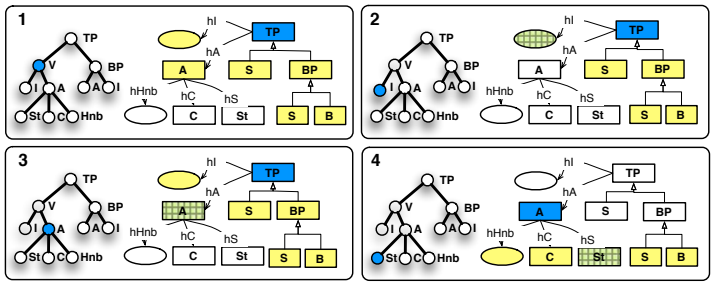


Fig. 5. The beginning proceeding of the algorithm.

In continuation "Address" and "Identifier" are matched after the same procedure until all elements of the schema tree have been explored.

5 Evaluation

We evaluated the Concept Finder algorithm using four scenarios based on different real world schemas and different ontologies. However, as these evaluations using large real world schemas and large ontologies are very time consuming and

[5] Note that it would easily be possible to test for all subclasses contained in the ontology, if children of e_{Vendor} match to properties of them. Hereby the exact subclass match could be found automatically. As this demands much more computational effort it was not implemented in the prototype and therefore the user has to assign the exact subclasses manually at the moment.

strenuous, we were not able to perform a comprehensive evaluation. Instead we created three evaluation scenarios based on the following real world schemas:

- `sap-order.xsd`
- `catalog.xsd`

The `sap-order` schema is a schema developed by SAP describing a purchase order in the well known order to invoice process [5]. The second schema, the `catalog.xsd` describes a product catalog and is part of the BMECat [6] standard. In addition to the schemas two different ontologies where used in the evaluation scenarios. The first one, which will in the following be called *Lifting Ontology (LO)* was manually developed after studying the SAP order to invoice process. The second one, the *Business Data Ontology (BDO)* [7] is an ontology developed as part of the DIP project [8]. It is based on the UBL [9] standard and describes the domain of business-to-business communication.

Based on these schemas and ontologies the following scenarios where developed for the evaluation:

- **Scenario 1:** *sap-order.xsd* → *LO*. Our first scenario uses similar inputs as the LO is based on the set of schemas which `sap-order.xsd` is part of.
- **Scenario 2:** *sap-order.xsd* → *BDO*. The BDO covers a bigger domain and is much more complete in sense of number of concepts than the LO ontology. In this scenario the only dependency between the schema and the ontology is the domain of interest. They have been created independently and for different purposes.
- **Scenario 3:** *catalog.xsd* → *LO*. For the third scenario the schema originates from a different domain than the used ontology, namely the exchange of product catalogues.
- **Scenario 4:** *catalog.xsd* → *BDO*. This scenario is similar to the third scenario. Only the used target ontology differs.

Using the metrics used in [1] we achieved the results presented in table 1. The table shows for each of the four scenarios described above two results. This is due to the fact that we evaluated the Concept Finder algorithm using two operation modes. The first row of values show the results achieved when running the algorithm in fully automatic mode whereas the second row shows the results when running the algorithm semi-automatically.

5.1 Discussion

The results of the scenarios 1,3 and 4 show, that the Concept Finder algorithm generally achieves very good results. Even for schemas that only partly overlap with the domain of the ontology, algorithm achieves and overall result between 0.75 and 0.8 in automatic mode and between 0.85 and 0.92 in semi-automatic mode.

In the second scenario, the algorithm achieves only poor results running automatically. The results in this scenario significantly improve if the algorithm

Table 1. The results of the evaluation of the Concept Finder algorithm

	Precision	**Recall**	**Overall**
Scenario 1*a*	0.746	0.842	0.898
Scenario 1*b*	0.831	0.858	0.970
Scenario 2*a*	-0.248	0.050	0.143
Scenario 2*b*	0.375	0.594	0.731
Scenario 3*a*	0.643	0.857	0.800
Scenario 3*b*	0.786	0.857	0.923
Scenario 4*a*	0.400	0.600	0.750
Scenario 4*b*	0.500	0.600	0.857

is run semi-automatically. The reason for the poor results in automatic mode is, that if the ontology is not complete enough, it is possible that by not finding a matching concept for the current element, the algorithm is mislead. Thus, if the comparison point (current class) in the ontology does not anymore correspond to what the element in the schema represents, the algorithm may not recover and will only produce false matches, leading to poor results.

Therefore the overall results of our experiments show are twofold. On the one hand, even in the full automatic mode, the algorithm performs very well for most inputs. On the other hand it is possible that the Concept Finder algorithm is mislead and may perform bad if the structures of the source schema and the target ontology are too different.

An important observation of the evaluation is that a semi-automatic lifting in general seems to perform very good in terms of precision, recall and manual effort and seems to be a very promising approach.

6 Summary and Outlook

In this paper we have presented a new algorithm for lifting existing XML Schema to ontologies. The first evaluation of this algorithm using real world schemas originating from the area of B2B communication show very promising results.

The current implementation of the algorithm is based on a very simple linguistic algorithm to identify possible correspondences between the source schema and the target ontology. If a set of matchers, similar to the ideas presented in [1] would be used, accuracy of the created mappings could be further improved. Furthermore $1-n$ and $m-1$ correspondences between schema elements and ontology concepts need to be taken into account. Finally a improvement of the user interface for the semi-automatic creation of liftings could improve the quality of the results and minimize the necessary user effort.

References

1. Do, H.H., Rahm, E.: COMA - a system for flexible combination of schema matching approaches. In: Proc. 28th VLDB Conference. (2002)

2. W3C: XML schema. Online (2001) http://www.w3.org/XML/Schema.
3. McGuinness, D., van Harmelen, F.: Owl web ontology language overview. http://www.w3.org/TR/owl-features/#s3.4 (2004) [Online; accessed 6-Sept-2005].
4. Princeton University: Wordnet - a lexical database for the english language. http://wordnet.princeton.edu/index.shtml (2005) [Online; accessed 3-Jul-2005].
5. SAP AG: Order schema. http://sap.com/xi/EBP (2002)
6. eBusiness Standardization Commitee: Bmecat. http://www.bmecat.org (2005) [Online; accessed 25-Aug-2005].
7. Nagypal, G., Lemcke, J.: D3.3 a business data ontology, wp3: Service ontologies and service description. - **D3.3** (2005) 141
8. DIP: Data, information, and process integration with semantic web services. online (2004) http://dip.semanticweb.org/.
9. OASIS: UBL. Online (2003) http://www.oasis-open.org/committees/ubl/.