Adapting I/O Parameters of Web Services to Enhance Composition*

Peter Bartalos and Mária Bieliková
Institute of Informatics and Software Engineering, Faculty of Informatics
and Information Technologies, Slovak University of Technology in Bratislava
{bartalos,bielik}@fiit.stuba.sk

Abstract

Web service chaining forms one of the basic operation during web service composition. To decide whether services can be chained, semantic matching between the I/O is usually evaluated. In this paper we propose an approach considering possible adaptation of output data before it is consumed as input by other service. We also define the corresponding rules based on which it is evaluated if the chaining is possible or not. Moreover, we deal also with the affect on workflow and service description, and their execution.

1 Introduction

One of the most studied topic in the field of web services is the composition – a process when web services are arranged into one complex service to satisfy a user goal for which there is no single service. The base of this process is service chaining. It happens when one service produces data consumed by another service. It is obvious that to do this, data level compatibility of the services is necessary.

Current approaches dealing with service chaining are usually based on semantic web technologies. They deal with the compatibility of services at the semantic level. This is possible because of available semantic description of web services. The aim of the semantic description is to add meaning to different elements of web services. The description includes also binding of the I/O of services to domain terms described as concepts. Based on this, the compatibility problem is transformed to a decision problem if some concept is semantically the same (or close) to the other.

In this paper we deal with web service chaining considering the possibility of data adaptation. Our approach goes beyond the idea that two services can be chained if the first produces data which can be directly consumed be the second service. It takes into account the possible adjusting of the produced output data before they are used as inputs. This way we enhance the possibilities to chain services. The data adjustment is done by performing data adaptation operations during the execution. The possible operations are: *decomposition*, *listing*, *composition*, and *list arrangement*.

This feature affects different aspects of the semantic web service composition. The semantic description of the services must follow some rules. The possible adaptation brings new opportunities to arrange web services. The workflow representation must be able to capture them. The reasoning required to decide the compatibility of web services must also consider the possible data adaptation. Finally, the execution engine must perform the adaptation operation over the data based on the workflow description. We deal in this paper with each of these issues and focus mainly on web service annotation and chaining.

Our work is motivated by the *SEMCO-WS* research project [11]. This narrowly focused project aims at automated composition of grid services workflows using semantics and comfortable user interfaces for complex grid middleware. The domain of *SEMCO-WS* is crisis management. The workflows we work with are oriented to simulations of fictive emergency scenarios. Some example services from this domain are presented in Fig. 1. Depicted service chaining problem serves also as an example in this paper.

In our example, service *S1* computes a simulation of radiation spread based on a defined scenario and region. The output consists of a 3D grid model containing the radiation dose rates and a list of risky areas. Service *S2* computes the radiation deposition based on the dose rates from the 3D model. Service *S3* creates the safety precautions for a given region and risk type. As we can see from the simplified implementation examples of the I/O parameters in Fig. 1, there is no exact match between services *S1* and *S2*, *S3*. However, there is a noticeable interest to chain them. The problem is that *S2* expects as input only a part of the output of *S1*. This is true also for *S3*. Moreover, *S3* expects only one item from the list of risky areas produced by *S1*.

In this paper we propose a solution for the chaining problem example depicted in Fig. 1. Using the adaptation oper-

*This work was partially supported by the Slovak Research and Development Agency under the contract No. APVV-0391-06 and by the Scientific Grant Agency of Slovak Republic, grant No. VG1/0508/09.

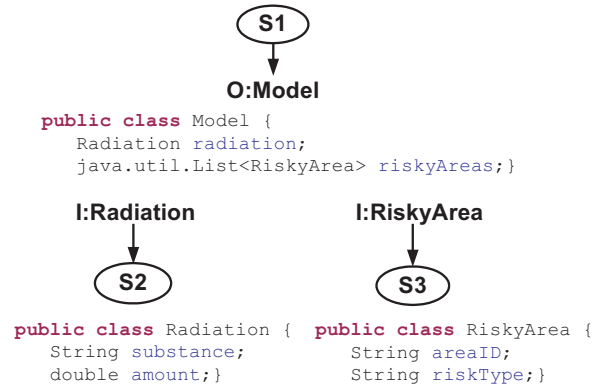

Figure 1. Examples of web services.

ations it is possible to chain services *S1* and *S2*, *S3*. Hence, the composition of these services is enhanced.

The rest of the paper is structured as follows. Section 2 deals with problem overview and presents related work. It focuses on web service description and chaining at both syntactic and semantic level. The chaining problem is discussed from semantic content and pre/postcondition point of view. In section 3 we present notes and recommendations regarding web service annotation which are important to effectively exploit the possibility of data adaptation. The web service compatibility in the context of data adaptation is discussed in section 4. We present devised new rules based on which the decision if services can be chained is done. These formally described rules enhance the possibility of service chaining in the case when data adaptation is performed. For each adaptation operation we also discuss the basic affects related to the workflow representation and execution engine. Finally, section 5 concludes our approach and presents future direction of our work.

To formalize our ideas in this paper, we utilize set algebra, first order and description logic. The following notation is used in the paper to formalize our statements. Symbols from set algebra and logic are used as usual. Symbol O denotes an output of a service. Symbol I denotes an input of a service. Symbols *rdf:List* refers to the class defined in RDF (Resource Description Framework[1]) as a list. Symbols *rdf:first*, *rdf:rest* refer to RDF properties expressing the first element of a list and the sublist without the first element.

2 Problem Overview and Related Work

2.1 Web service description

The basic document describing web services is WSDL (Web Service Description Language). The aim of WSDL is to provide information necessary to invoke the web service. For us the most important issue is that it defines the structure of the messages sent between the service invoker and provider. These messages describe data corresponding to the services' I/O parameters serialized into XML. The message consists of one or more *parts* each corresponding to one parameter. These have defined type which can be simple (string, integer) or complex. This corresponds to the primitive data types and classes in an object-oriented (OO) language. WSDL is considered to be a syntactic description of the service.

Semantic web services are web services enhanced by semantic description, i.e. they are semantically annotated. The most known languages for this purpose are OWL-S[2], WSDL-S[3] and WSML[4]. They differ in the complexity and expressivity of their construction elements. In each case the semantic annotation binds the elements of the web service to domain terms described by ontologies. From all of the elements it focuses on the I/O of the web service and the condition under which the web service can be invoked and the condition which will hold after its execution. The domain ontology is usually defined using OWL (Ontology Web Language) [10] based on description logic [1].

The semantic annotation of the I/O describes the semantics of the attributes defined within the WSDL message. This way we define also the meaning of the attribute, not only its data type.

The pre/postconditions of services are usually expressed as logical statements. In [7], the authors define pre/postconditions as a list of predicates which hold before/after execution. There is an implicit conjunction between them, i.e. all the predicates must hold. The predicates cannot be combined using other logical operations such as disjunction. The same approach is used in [12].

In our previous work [3] we proposed an approach to describe complex pre/postconditions. It allows to combine simple statements using first order logic operators such as negation, conjunction, disjunction. Moreover, it is also possible to use universal and existential quantification to increase the expressivity of pre/postconditions' description. The complex conditions are expressed using an extension of SWRL – Semantic Web Rule Language[5]. The SWRL expressions are then included within the *process:hasPrecondition* and *process:hasEffect* parts of the OWL-S service description which are dedicated for this purpose.

Other approaches describing complex pre/postconditions are presented in [5, 8]. The ap-

[1] http://www.w3.org/RDF/

[2] http://www.daml.org/services/owl-s/1.0/owl-s.html

[3] http://www.w3.org/Submission/WSDL-S/

[4] http://www.wsmo.org/wsml/

[5] http://www.daml.org/2004/11/fol/proposal

proach in [8] uses USDL language to define the formal semantics of the service. Its disadvantage is that it allows not to use existential and universal quantification. In [5] the authors present an extension of OWL-S to describe pre/postconditions. They do not use the dedicated parts of OWL-S to express them. However, their approach seems to have the same expressivity as our.

2.2 Web service chaining

2.2.1 Semantic content point of view

Two services S_{anc} and S_{succ} can be chained, i.e. are compatible if S_{anc} produces data which can be consumed by S_{succ}. This means that the output of S_{anc} is used as one of the inputs of S_{succ}. In this paper the service producing data for other service is called *ancestor service*. On the other hand, the service consuming data produced by ancestor service is called *successor service*. The question if two services can be chained is answered at both syntactic and semantic level.

At the syntactic level the compatibility concerns the structure of the messages wrapping the service's I/O data. The output data of the ancestor service must be serialized into a XML fragment which is correctly deserialized into the successor service's input.

The approaches to semantic web service composition evaluate the matching between the I/O parameters at semantic level. They usually chain services if there is a subsumption between the I/O parameters: $O \sqsubseteq I$ (O subsumes I). In [9] the authors present four matching functions based on which services can be chained: *Exact*, *PlugIn*, *Subsume*, *Intersection*. If some of these hold for the I/O, a *Valid causal link* between them is established. If the matching function is *Exact* or *PlugIn*, a *Robust causal link* is established. This means that the output *O* can be directly consumed by the successor service. If the matching function is *Subsume* or *Intersection*, abduction and constraint relaxation is applied to make possible the service chaining.

Our attitude regarding service chaining is that it is not enough to consider only one compatibility level. To get sensible service composition both levels must be considered.

If the services are compatible only at the syntactic level, it means that the data produced by the ancestor service can be successfully used as input for the successor service. Hence, the execution of the successor service with the given input will be successful. However, it will not make sense to do that. One example is the chaining of a service producing an integer expressing the number of the *risky areas* and a service expecting as input an integer expressing the number of *endangered people*.

If the services are compatible only at the semantic level, it means that the successor service cannot consume data produced by the ancestor service. This again means that it makes no sense to chain the services because the execution of the successor service is not possible. This situation may occur when wrong annotation is created. For example we annotate two different *weather forecast models* with the same concept.

2.2.2 Considering pre/postconditions

When dealing with the compatibility question of semantic web services considering pre/postconditions we refer to our previous work. In [3] we describe how complex pre/postconditions are expressed to annotate web services. The paper presents also rules deciding if two services are compatible. We distinguish between weak and strong compatibility. It depends on the certainty of the compatibility. If the execution of the ancestor service results always in a condition matching the precondition of the successor service, there is a strong compatibility between them. If this is not assured, i.e. such condition is achieved only in some cases, there is a weak compatibility. This situation may occur by means of the disjunction operation usage. In this case the compatibility type is known only at the execution time.

The complex pre/postconditions are expressed as first order logic formulae. Predicates are combined using logical operators. At the semantic level, the predicates correspond to the properties between two concepts. The predicate parameters may refer to instances representing I/O parameters of services, other instances, owl data values or owl variables.

Our approach to decide the compatibility is based on normalization of the pre/postconditions into disjunctive normal form (DNF). Let denote the set of subformulae representing the respective conjunctions of the pre/postcondition DNF of service S as $C_{pre}(S)$ and $C_{post}(S)$. To decide if two services can be chained we present the following *compatibility statements*:

Statement 1:

Services S_{anc} and S_{succ} are not compatible if:
$\forall c_{pre} \in C_{pre}(S_{succ}), \nexists c_{post} \in C_{post}(S_{anc}) :$
$c_{post} \Rightarrow c_{pre}$ is true.

Statement 2:

Services S_{anc} and S_{succ} are weakly compatible if:
$\exists c_{pre} \in C_{pre}(S_{succ}), \exists c_{post} \in C_{post}(S_{anc}) :$
$c_{post} \Rightarrow c_{pre}$ is true.

Statement 3:

Services S_{anc} and S_{succ} are strongly compatible if:
$\forall c_{post} \in C_{post}(S_{anc}), \exists c_{pre} \in C_{pre}(S_{succ}) :$
$c_{post} \Rightarrow c_{pre}$ is true.

3 Annotating Web services

Upper part of Fig. 2 depicts the common data model in object-oriented paradigm (for simplicity we neglect the primitive data types). This model describes the structure of any data. Hence, it models also the I/O parameters of web services at the syntactic level. Each *wsdl:part* element in the WSDL document has defined the *attribute type* corresponding to some *Class*, see Fig. 2. At the semantic level we annotate *wsdl:part* element by some owl class. This class can subsume *rdf:List* to express multiplicity.

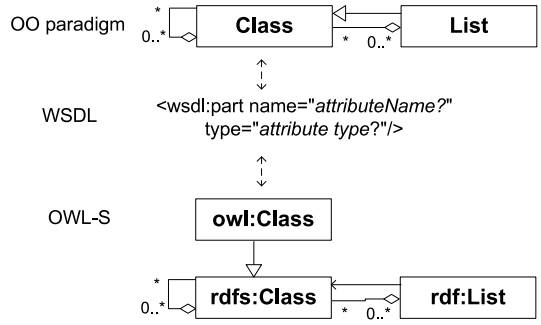

Figure 2. Common data model.

The OWL and OWL-S provide a lot of freedom when annotating web services. As languages, they allow to create also meaningless annotations which are incorrect considering the goal of the annotation. It is up to the designers creating the ontology and the services' semantic descriptions to create useful annotation. They must be aware of the goal of the annotation which is the enhancement of web service discovery or composition. Hence, the knowledge of the mechanisms behind these processes is required.

When annotating web services it is important to follow the rule that never create annotation leading to a situation that some services are not compatible at the syntactic level but at the semantic level they do match. The annotations should be created in such a way that it is always true that if some services are compatible at the semantic level, then they are compatible also at the syntactic level. Following this rule allows to omit the checking of the compatibility at the syntactic level because if it holds at semantic level, it implicitly holds also at the syntactic level.

We recommend to design the semantic description in such a way that it is in strong correspondence with the syntactic level. The designer should be aware of the knowledge behind object-ontology mapping [2, 6]. The base is that classes in OO paradigm correspond to the ontological classes. If a class has attributes, i.e. it aggregates other classes (or data types), there is a property between the ontological equivalent of this class and the ontological equivalent of the aggregated class. The property corresponds to the name of the attribute.

From the point of view of web service chaining considering data adaptation it is important to correctly express at the semantic level the cardinalities of the class attributes. This is important because from this we know which attributes must be filled in the input or output of services. If the attribute cannot be null, we define a cardinality restriction (on the property corresponding to this attribute) to value 1. If the attribute represents a collection (list, field) we annotate it with a class C defined as $C \sqsubseteq$ *rdf:List* and $C \equiv \forall$*rdf:first*.$C_{el} \sqcap \forall$*rdf:rest.C*. The element of the collection corresponds to ontological class C_{el}.

4 Data Adaptation

In the following we deal with service chaining considering also the possibility of data adaptation. The data adaptation is realized by four operations: *decomposition*, *listing*, *composition*, and *list arrangement*. The goal of the adaptation is to prepare the input data for a web service from the outputs of the ancestor services. It is applied when the data cannot be used directly but the adaptation makes them usable. To decide if services can be chained we check if the data match from the semantic content and also the pre/postcondition point of view. For each adaptation operation we propose rules deciding the compatibility from these two viewpoints.

Let $conj$ be a conjunction. $\nabla(conj, symbol)$ denotes a formula which is a conjunction of predicates constructed as follows. First, we add into it those predicates from $conj$ where the first parameter is $symbol$. In each subsequent iteration we add into it those predicates from $conj$ where the first parameter is a symbol appearing as a second parameter of some predicate which already is contained in $\nabla(conj, symbol)$. The iteration finishes when there are no more predicates which may be added.

To evaluate the compatibility at the pre/postcondition level we regard to the compatibility statements presented in section 2.2.2. Also when considering data adaptation the compatibility is evaluated as none, weak or strong. The difference is that we must consider other implication (not $c_{post} \Rightarrow c_{pre}$) or other formula C_{post} against the original definition. These facts are depicted in the following.

4.1 Decomposition

Decomposition is an operation which decomposes a complex data into its components. It is applied when a web service requires as input only a part of the complex data produced by the ancestor service.

From the semantic content point of view, two services can be chained if the output of the ancestor service has such a property R whose range subsumes the input of the successor service, i.e. $O \equiv \exists R.O_i \sqcap \geq 1R \sqcap \leq 1R$ and it holds

that $O_i \sqsubseteq I$. This is the case of services *S1* and *S2* from our example.

Let o_i to be a symbol used in the postcondition formula to denote that component of O which is required as input. Based on the compatibility statements, services can be chained considering their pre/postcondition if we consider the implication $\nabla(c_{post}, o_i) \Rightarrow c_{pre}$ (only non empty $\nabla(c_{post}, o_i)$ are considered). C_{post} is defined as originally.

In the case that decomposition is used, the workflow representation must depict which services are chained and which part of the ancestor service's output is used as input of the successor service. The execution engine must decompose the output and retrieve the required part. This is then used as input to invoke the successor service.

4.2 Listing

Listing is an operation during which we get the elements of a given list for further separated processing. This is the case of services $S1$ and $S3$ from our example. It is applied when the successor service processes each element of the output list separately.

From the semantic content point of view, two services can be chained if the output of the ancestor service is defined as $O \sqsubseteq$ *rdf:List* and $O \equiv \forall$*rdf:first*.$O_{el} \sqcap \forall$*rdf:rest.O* and it holds that $O_{el} \sqsubseteq I$.

Let o_{el} to be the symbol of the element of the list produced by the ancestor service in its postcondition formula. To evaluate the compatibility at pre/postcondition level the following implication must be considered: $\nabla(c_{post}, o_{el}) \Rightarrow c_{pre}$ (only non empty $\nabla(c_{post}, o_{el})$ are considered). C_{post} is defined as originally.

When listing is applied, the workflow representation must express which services are chained. During the execution, the successor service is invoked for each element of the list produced by the ancestor service. The respective element is used as the input parameter.

4.3 Composition

Composition is an operation which composes the required data from its parts. These parts are produced by different web services.

From the semantic content point of view, services $S_{anc1}, \ldots, S_{ancn}$ with corresponding outputs $O_1, \ldots, O_n$ and a service with input I can be chained if I is defined as $I \equiv \exists R_1.I_1 \sqcap \geq 1R_1 \sqcap \leq 1R_1 \sqcap \ldots \sqcap \exists R_n.I_n \sqcap \geq 1R_n \sqcap \leq 1R_n$ and it holds that $O_1 \sqsubseteq I_1, \ldots, O_n \sqsubseteq I_n$. $R_1, \ldots, R_n$ are symbols of any properties.

To evaluate the compatibility at the pre/postcondition level, we must construct C_{post} as formula which is a conjunction of the postconditions of $S_{anc1}, \ldots, S_{ancn}$, i.e. $C_{post} = C_{post}(S_{anc1}) \wedge \ldots \wedge C_{post}(S_{ancn})$. The implication is defined as originally, i.e. $c_{post} \Rightarrow c_{pre}$.

In the case that composition is used, the workflow representation must depict which ancestor services produces which components of the data expected by the successor service. The execution engine must compose the input data after each ancestor service returned the output. The resulting composite data are used as the input to invoke the successor service.

4.4 List arrangement

List arrangement is an operation where we arrange a list from elements. There are three cases how the elements can be provided. First, they are a result of multiple invocation of the same service. Second, they are a result of the invocations of different services. The third, most general case combines the first two with a situation that into a resulting list we include also lists of elements which are produced by some ancestor service. This means that the resulting list is arranged from elements which are a result of a single or multiple invocation of some services and lists of elements produced by some other services. The practical usefulness of this case is disputable. In the following, in each case the input of the successor service I is defined as $I \sqsubseteq$ *rdf:List* and $I \equiv \forall$*rdf:first*.$I_{el} \sqcap \forall$*rdf:rest.I*.

Multiple invocation of the same service

From the semantic content point of view, service with output O can be chained with service with input I if it holds that $O \sqsubseteq I_{el}$.

When considering compatibility from pre/postcondition point of view, both C_{post} and implication are defined as originally.

Single invocation of different services

From the semantic content point of view, services $S_{anc1}, \ldots, S_{ancn}$ with corresponding outputs $O_1, \ldots, O_n$ and a service with input I can be chained if it holds that $O_i \sqsubseteq I_{el}$ for $i = 1, \ldots, n$.

To decide the compatibility at the pre/postcondition level we must enhance the compatibility statements. First we evaluate the compatibility for each ancestor service with the successor service separately based on the rules holding in the case of multiple invocation of the same service. Based on this, services $S_{anc1}, \ldots, S_{ancn}$ are not compatible with service S_{succ} if there exist at least one service from $S_{anc1}, \ldots, S_{ancn}$ which is not compatible with S_{succ}. Services $S_{anc1}, \ldots, S_{ancn}$ are weakly compatible with service S_{succ} if each service from $S_{anc1}, \ldots, S_{ancn}$ is weakly or strongly compatible with S_{succ}. Services $S_{anc1}, \ldots, S_{ancn}$

are strongly compatible with service S_{succ} if each service from $S_{anc1}, \ldots, S_{ancn}$ is strongly compatible with S_{succ}.

Combined

In this case we combine the outputs of services producing one or more elements and arrange them into a resulting list. To decide the compatibility we must combine the rules defined for previous two cases. It is straightforward that each element produced by the ancestor services which is put in the list must subsume the element of the list corresponding to the input. From the pre/postcondition point of view, C_{post} is a conjunction of postcondition of each ancestor service contributing into a resulting list. The postconditions are normalized into DNF. In the case of a service producing a list, we apply $\nabla(c_{post}, o_{el})$ for each conjunction of the postcondition before it is put into C_{post}.

To apply list arrangement in the workflow, we must depict which services are contributing to the list. Moreover, hence some services may be invoked multiple times to produce elements, we must also express the number of invocations. It is important to note that multiple invocation of some service is possible only if listing was applied previously in the workflow. Hence, the number of invocations is equivalent to the size of the list for which listing was applied before. The execution engine must arrange the required list after each service produced the expected number of elements. In the case of single invocation, one element is expected.

5 Conclusions and Future Work

In this paper we presented an approach enhancing the possibilities to chain services. It goes beyond the simple checking of the subsumption relation between the I/O. It considers the possible adaptation of the output data before it is used as input. We propose devised new rules for web services chaining and corresponding data adaptation operations, which are necessary during the execution of the composite service. Our approach is integrated in our automatic web service composition method. This method is experimentally evaluated within a multi-user workflow editor tool [4]. It supports user collaboration during the creation and execution of workflows. The composition method is integrated into the editor to automate this process. It considers the matching rules when deciding whether the services can be chained. The workflow execution engine of the multi-user editor was extended to realize the data adaptation operations over outputs to create the required inputs.

Our common, domain independent approach showed usefulness in the context of crises management workflows we deal with in *SEMCO-WS*. Currently we deal with examination of other adaptation mechanisms and their affect on service description and execution. We also research other levels at which compatibility of services should be considered. An example in this direction of our work is a study based on a hypothesis that a chaining of some services may be not successful because the resulting QoS attributes of the chain are of poor quality.

References

[1] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003.

[2] P. Bartalos and M. Bieliková. An approach to object-ontology mapping. In *2nd IFIP Central and East European Conf. on Software Engineering Techniques CEE-SET 2007*, 2007.

[3] P. Bartalos and M. Bielikova. Fast and scalable semantic web service composition approach considering complex pre/postconditions. In *WSCA '09: Proc. of the 2009 IEEE Congress on Services, Int. Workshop on Web Service Composition and Adaptation*. IEEE CS, 2009.

[4] P. Bartalos, I. Kapustik, and V. Rozinajova. Visual support of workflow composition involving collaboration. In *GCCP '08: Proc. of the 2008 Int. Workshop on Grid Computing for Complex Problems*, pages 120–127. SAS, 2008.

[5] Y. Gamha, N. Bennacer, G. V. Naquet, B. Ayeb, and L. B. Romdhane. A framework for the semantic composition of web services handling user constraints. In *ICWS '08: Proc. of the 2008 IEEE Int. Conf. on Web Services*, pages 228–237. IEEE CS, 2008.

[6] A. Kalyanpur. Automatic mapping of OWL ontologies into Java. In *F. Maurer and G. Ruhe, Proc. of the 17th Int. Conf. on Software Engineering and Knowledge Engineering, SEKE'2004*, 2004.

[7] M. Klusch, A. Gerber, and M. Schmidt. Semantic web service composition planning with owls-xplan. In *AAAI Fall Symposium on Semantic Web and Agents*, Arlington VA, USA, 2005. AAAI Press.

[8] S. Kona, A. Bansal, and G. Gupta. Automatic composition of semantic web services. In *ICWS '07: Proc. of the 2007 IEEE Int. Conf. on Web Services*, pages 150–158. IEEE CS, 2007.

[9] F. Lecue, A. Delteil, and A. Leger. Applying abduction in semantic web service composition. In *ICWS '07: Proc. of the 2007 IEEE Int. Conf. on Web Services*, pages 94–101, Los Alamitos, CA, USA, 2007. IEEE CS.

[10] D. L. McGuinness and F. van Harmelen. *OWL Web Ontology Language Overview*. 2004. retrieved from http://www.w3.org/TR/owl-features/, 2007.

[11] M. Paralic, O. Habala, J. Paralic, and P. Bartalos. Semantic composition of web and grid services. In *Znalosti 2009*, pages 355–358, Brno, 2009.

[12] A. Sirbu and J. Hoffmann. Towards scalable web service composition with partial matches. In *ICWS '08: Proc. of the 2008 IEEE Int. Conf. on Web Services*, pages 29–36. IEEE CS, 2008.