Computing and Informatics, Vol. 30, 2011, 793–827

AUTOMATIC DYNAMIC WEB SERVICE COMPOSITION: A SURVEY AND PROBLEM FORMALIZATION

Peter Bartalos, Mária Bieliková

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
e-mail: {bartalos, bielik}@fiit.stuba.sk

Communicated by Ladislav Hluchý

Abstract. The aim of Web service composition is to arrange multiple services into workflows supplying complex user needs. Due to the huge amount of Web services and the need to supply dynamically varying user goals, it is necessary to perform the composition automatically. The objective of this article is to overview the issues of automatic dynamic Web service composition. We discuss the issues related to the semantics of services, which is important for automatic Web service composition. We propose a problem formalization contributing to the formal definition of the pre-/post-conditions, with possible value restrictions, and their relation to the semantics of services. We also provide an overview of several existing approaches dealing with the problem of Web service composition and discuss the current achievements in the field and depict some open research areas.

Keywords: Web services, automatic dynamic web service composition, semantics, pre-/post-conditions.

1 INTRODUCTION

One of the most important benefits of web services is their interoperability. This feature allows that in the case of diverse software systems, one system can exploit the functionality of the second. Moreover, we can build complex systems by integrating diverse applications independently on the platform and place in which these are running on. Integral parts of applications are exposed as web services to make accessible the functionality they perform. These web services are then combined

together to perform more complex tasks. The process of arranging the services into a complex workflow is called web service composition. The interoperability and composability capabilities of Web services promote their usability in different areas, e.g. in science in the context of scientific workflow management [13, 39, 24].

The motivation for web service composition, as just defined, is more about solving technical problems related to application integration, such as platform diversity. In the last years the vision of web service composition moved toward this. A lot of research effort has been put towards developing approaches able to provide the systems user his goal by composing web services. The user must just describe what he wants to achieve and the application takes care of automatic service composition fulfilling this goal the best. In this scenario the problem is to find relevant web services and arrange them in a meaningful, optimal manner. Web service composition is required because usually there is no single service which can fulfill the user goal completely.

Automation of the service composition is crucial for fulfilling varying composition goals. To make this possible, we cannot rely only on pure syntactic descriptions of services, since these are insufficient for machine processing and for achieving meaningful results. Thus, the Web service descriptions are enhanced with semantics. This makes the automatic composition feasible; however, semantic annotation and correct understanding of semantics still deals with challenges.

Although the basis of the semantic Web service composition problem is easy to explain, there are numerous details making the whole problem different. Even small differences can make its complexity significantly different [2]. Thus, the vague definition is insufficient and precise formalization is required.

Tremendous number of research papers had been published, focusing on different aspects of the Web service composition [10, 18, 46]. The research tends to focus on issues related to the QoS [3, 7, 8, 23, 25, 34, 29], the pre-/post-conditions [6, 8, 28, 31, 30, 12, 53, 42], the user constraints and preferences (e.g. soft constraints) [1, 20, 26, 37, 38], the consideration of the user context and complex dependencies between services [40, 57, 15], and the transactional behavior of Web service compositions [43, 21].

The objective of this article is to overview automatic dynamic Web service composition in wider context. We discuss the issues related to the semantics of services. To make the service composition problem clear, we propose its formalization. There are some works formalizing the problem already [31, 30]. Our main contribution is the formal definition of the pre-/post-conditions, with optional value restrictions, and their relation to the semantics of services. In opposite to other works, we clearly state what the pre-/post-conditions and the value restrictions are, how they are related to the semantics of services, and how they are considered during the composition process. This article also provides an overview of several existing approaches dealing with the basic problem of Web service composition. However, it is hard to strictly categorize existing approaches; we divided them into three groups. First, approaches considering only the I/O during service chaining are presented. These usually deal also with the QoS optimization and performance issues. Then, we discuss approaches considering additional metadata of services, e.g. the pre-/post-

conditions. This is followed by approaches that allow defining also soft constraints to the composition goal or used services. Finally, we discuss the achievements in the field and depict some open research areas, which we believe should obtain more research attention.

2 AUTOMATIC DYNAMIC WEB SERVICE COMPOSITION

2.1 Overview of Service Composition

The aim of Web services is to provide a functionality, which can produce required data and effects in a widely accessible, easily discoverable form (by machines). The philosophy of Web services goes beyond the technical solution of making some procedures possible to execute remotely, through the Web. The concept of Web services introduces a new paradigm affecting distributed computation and software system development [45, 44]. In the context of this new paradigm, the service composition stands as a mechanism combining multiple services to build up more complex functionalities. This enhances the potential of single services.

The composition of multiple services supplies more complex needs, which cannot be achieved using a single service. Even if there is no single service providing the desired functionality, it may be possible to design a composite service from the single services, and thus provide the desired functionality. The service composition is classified based on two main aspects [18]. First, we distinguish between static and dynamic composition. Second, we divide the composition approaches into manual, and automatic.

The first perspective concerns the time when the service is selected to be a component in the composite service. The static approaches made the selection during design time. The dynamic approaches select services during run-time, according to the needs given by the particular user goal. The second aspect deals with the fact whether the composition is made manually by a designer, or automatically by a composition system. In our case, we deal with automatic dynamic service composition. In this context, the problem is defined as follows: *Given a query describing the composition goal and providing some inputs, design automatically a composite service from the available services such that if it is executed, it produces the required goal.*

It is important that in our context, the Web service is an abstract entity providing its functionality on the Web. It consumes inputs and produces outputs. It can be also associated with pre-/post-conditions. The Web services are divided into two main classes: RPC style and RESTfull services. Although both of them are associated with the I/O and pre-/post-conditions, there are fundamental conceptual differences between them. In the case of the RPC style Web services, we consider them as remotely invocable distributed methods which we communicate by using messages. In the case of REST-full services, the focus is on distributed stateful resources over which we perform the standard HTTP operations. Considering Web service composition, the research attention is almost strictly focused on RPC style services. However, there are some works dealing with RESTful service

composition [60]. Although both classes of Web services fit our definition, our work is conceptually about composition of RPC style services, i.e. the focus is on the operation performed by the service, which is specified by the description of its I/O and the pre-/post-conditions. Note that in this context, we can consider also human tasks as services realized by humans. If the procedures performed by humans are described by the I/O and the pre-/post-conditions, we can consider them during the composition.

There is a correspondence between service composition and declarative programming. In declarative programming we describe what the program should accomplish instead of describing how to do it. It is the responsibility of the program interpreter to accomplish the described intent. Similarly, during service composition, the user describes what s/he wants to achieve and the mechanism behind the service composition takes care of accomplishing it.

From another point of view, *artificial intelligence planning* and service composition have much in common [46]. Depending on the concrete assumptions to the problem, they are more or less overlapping. The basic problem is finding a sequence of actions, whose execution leads to the achievement of the desired goal, and it is common for both. Differences may follow for example from a different perception of the time required to realize an action. Service composition usually does not design a time schedule of service executions. Another difference is that the service composition focuses more on the inputs required to execute a service and the outputs produced by it. The examination whether the outputs produced by a service can be consumed as inputs by another service has a lot of attention in the field. Artificial intelligence planning focuses more on the pre-conditions and effects of the actions. On the other side, the same techniques can be used when solving service composition, or artificial intelligence planning problem, i.e. the basis of both can be transformed to the same problem.

The parallel with the artificial intelligence planning shows an important fact determining the class of problems solvable by service composition. Note that not each real problem can be described and solved using planning. Examples are the environments where the future state cannot be determined before it is reached. For instance, the n-body problem belongs into this category. It deals with finding of motions of n bodies determined by Newton's laws of motion and Newton's law of gravity. The first contribution to the solution of the n-body problem has been done by Poincaré in 1892 and the problem was finally solved by Karl Frithiof Sundman (for $n = 3$) in 1912. The result is that, in general, it is not possible to solve the n-body problem analytically. Only numerical methods can be used to find a solution with arbitrary precision (greater than zero). Consider a planning problem that we want to fly to the Mars and land on an exactly specified point. The existence of the n-body problem proves that a plan which solves this problem cannot be found. The consequence is that also the Web service composition cannot deal with any problem; however, the set of solvable problems is wide.

The overall process of the Web service composition should be seen more widely and includes also the problem of a user goal description, acquisition of the necessary

input data from the user, design and execution of the best possible composite service, presentation of the results to the user [55]. As shown in Figure 1, the overall process incorporates two parties: the user and the composition system. The user, who may be a human or a software system, requests a composition (1). The request contains the goal description (2). The user also has to provide some input data (3). These may be defined by the user him/herself or the composition system identifies what data are required. After these steps, the workflow design and execution takes place (4). In common, the user goal can be achieved by several alternative solutions. These vary in different issues which can be divided into two groups. The first includes issues affecting the users' satisfaction degree based on the quality of the result, achieved by composite service execution, e.g. the user might have preferences. The second concerns the quality of the execution process affected by the QoS (Quality of Service) attributes which are non-functional properties of services. The aim is to find the best solution considering all these issues. Usually there is no solution being the best from each point of view. Hence, a suboptimal one, having the best total asset, is selected. The found solution is executed to produce the required result (5). If the user has requested some data, these are retrieved from the result and provided to him/her (6). The user is also informed about the resulting status of the execution, e.g. whether it was successful (7).

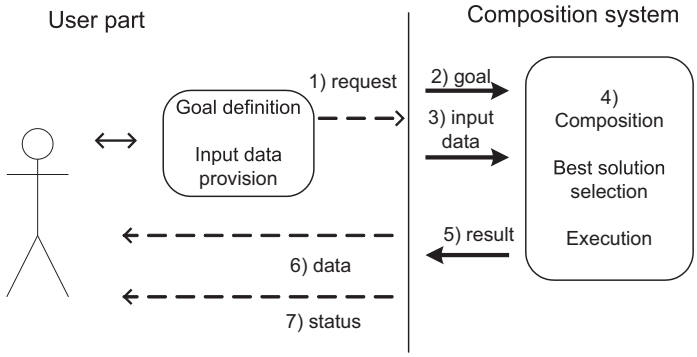


Fig. 1. Service composition process overview

The user goal has two basic essences: desired data, and effects or state (following chapter discusses different aspects of the user goal). This corresponds to what can be produced by web service execution. Usually the desired goal can not be fulfilled by executing a single service. Instead, complex workflow is designed, in which web service execution stands as an atomic activity.

The services are combined together using control constructs such as parallel, sequential execution. The combination of services is required for two reasons. First, the user goal may consist of several subgoals, each fulfilled by different service. Second, execution of certain services could be required, to produce data or achieve effects which are necessary to execute other services. The mechanism behind au-

tomatic web service composition must design a schema describing the control and data flow of the execution, i.e. it depicts the synchronization of the particular service executions, and prescribes which service produces output data used as input by other services or achieves effect required before other service can be executed.

2.2 The Need of Semantics

The Web was primarily designed for use by humans. Nevertheless, there is an effort to automate its use and bring the Web more accessible for machines. This has brought forward the need for machine processable representations of semantically rich information: a vision at the heart of the *Semantic Web* [14]. Nowadays, approaches based on giving additional information – semantics, to the content of the Web are researched. The semantic description focuses both on the content information and services available on the Web. The added metadata are used to enhance the processing of the Web resources (documents, services) and its automation. Without metadata, we explore only low level data such as HTML and WSDL files. These are originally not intended to be used for advanced automatic processing, and thus the accuracy is low in this case. The semantics is a way how more sophisticated exploitation of the Web resources could be achieved.

Web services enhanced by additional metadata expressing their semantics are called *semantic Web services* [17, 52]. Semantic Web services are the result of the Web evolution in two directions: adding dynamic elements to the Web and enhancement of the syntactic description of Web services, see Figure 2. At the beginning, the Web was only a collection of documents, i.e. it offered a static content. This had evolved into an environment offering also different functionalities via services, i.e. dynamic elements were appended. The additional semantics tends to solve the problem of hard interpretation of the syntactic description of Web resources. This causes problems when searching for relevant information or services.

Moreover, the popularity of the Web led to its enormous expansion in the amount of documents and services available. Processing of these by humans is impossible without additional machine support. Hence, the Web must be adapted to allow sophisticated processing by machines. Not only humans are considered as the users of the Web. The Semantic Web is an initiative aiming to bring the Web accessible for machines – software agents. These can use methods which are more resistant to the problem of Web expansion than humans are. The performance of the software agents is limited due to weak structuring and hard machine interpretation of the Web resources. The semantic approaches tend to overcome the above problems. The aim is to automatically process Web resources and to make it fast and with high accuracy.

In the context of service composition, the semantics helps discover and arrange services which are relevant for the given request [12, 53]. Without semantics we rely only on syntactic description of services. These do not provide enough information to recognize the functionality of the service and thus automatic service composition, satisfying dynamically varying user goals, is not feasible.

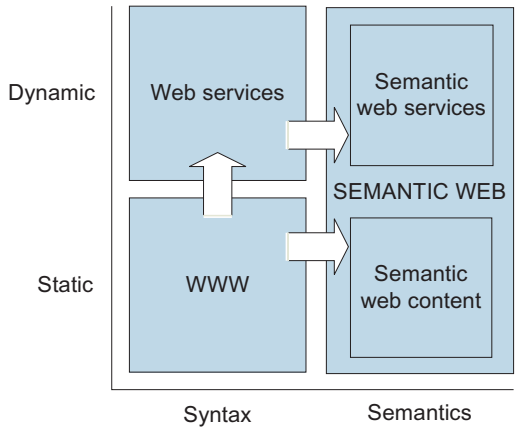


Fig. 2. Web evolution to Semantic Web services

Technically it is possible to design a composite service satisfying only the requirement that if some service produces output data used as input by another service, the output and the input must have the same data type, defined in XML schema within the WSDL description. Workflows designed in this manner are executable. However, it is meaningless without considering other aspects. The aim of automatic service composition is to design a composite service whose execution results in a predefined user goal. This means that the composite service must be meaningful, which is achieved by considering the semantics of services. This principle guaranties that the composite service is not only executable, but also leads to a desired, meaningful goal. Without semantics, we cannot disambiguate which services realize the required task.

In the context of automatic semantic Web service composition, both the query and services are described at the semantic level. From the user goal point of view, the semantics helps the user better express his/her intent. The semantics of services supports the decision if the execution of the service can result in data and effects required by the user, or other services. The required/provided data presented in the query and the I/O of services have defined meaning. To enhance the opportunities we deal also with states representing the desired effects in the goal, or condition, which must hold before service execution, or an effect made by service execution. The meaning of the service elements is provided by their being based to the ontology. The ontology presents a shared vocabulary of terms – conceptualization of the given domain. It is crucial that the semantic annotation of services guaranties that different systems understand the meaning of service elements in the same way.

2.3 Semantic Annotation of Web Services

From the technical point of view, a Web service is a collection of executable functions, available as a Web resource. Each has a name, unique location specified by its URL, and a set of operations. The operations have names and defined I/O types.

The information about these is available in the syntactic description using *Web Service Description Language* (WSDL). The WSDL description of the service includes also all the information necessary to correctly invoke it. In WSDL 2.0, these two aspects are described in different stages: abstract and concrete. If we deploy the same implementation of some functionality to several Web service servers, we get corresponding number of Web services. Each of them is a unique one and has its own WSDL description. However, they share the same abstract description.

In the semantics based approach, the service provider should create also semantic annotation of the services and provide them in the registry. The semantic description relates to the abstract definition of services which includes the I/O types. These attributes do not vary based on the place where the service is deployed. This means that the same description suits all the deployments of the same implementation. Moreover, if we implement the same functionality in different programming languages, we can use the same semantic description in each case. Hence, reusability of the descriptions is well-advised. The available semantics is then used to enhance a search considering only pure syntactic descriptions with semantics based search, to get more appropriate results. The semantics is beneficial also to define the requestor's intent. It allows more expressive definition of the user goal.

Several languages for semantic annotation of Web services are proposed. The most known ones are *OWL-S*[1], *WSML*[2], *USDL*, and *WSDL-S*[3]. They differ in complexity and expressivity of their construction elements. In each case the semantic annotation binds the elements of Web services to domain terms. It focuses on the I/O, the conditions under which the Web service can be invoked (pre-conditions) and the conditions which hold after its execution (post-conditions).

By binding the I/O of services to ontological concepts, we still have only a limited option to express the functionality of the service. A more natural way to express some behaviour is by using a *cause-effect* paradigm. Web services may have defined pre-/post-conditions to better express what they do. This kind of service modelling helps improve the way how the user expresses his/her goal, service discovery, and composition [12, 53].

2.4 Understanding the Semantics

The core of a service composition system can be seen as a black box module with a defined interface. Its aim is to: based on the query, compose the services which are made available in such a way that the execution of the composite service satisfies the goal described in the query. Such a black box requires a specific formulation of the user goal in a defined language. The results are provided in a specified form too. These specifications are defined by the interface. Any entity able to fulfil the requirements defined by the interface and to execute an invocation of the composition

[1] `http://www.w3.org/Submission/OWL-S/`
[2] `http://www.w3.org/Submission/WSML/`
[3] `http://www.w3.org/Submission/WSDL-S/`

system can utilize it. On the other side, the correct form of the composition request, followed from the interface specification, does not guarantee successful utilization of the composition system.

To achieve valuable results, both the requestor and the composition system must understand the semantics of the query the same way. Moreover, the composition system should understand the semantics of services created by the providers. Hence, there are three parties which must share some knowledge. The correct understanding of semantics of services is a complex task. It is affected by several issues such as *domain orientation*, or *openness of the services set* (i.e. if we deal with a predefined set of services, or an open service registry). These are important from the point of view of managing the ontology creation, its maintenance, and service annotation. In some cases these issues can be handled manually. However, considerable automation is required for practical usability.

The composition system requires a consistent ontology including all the terms used to annotate all the services the system uses during the composition. This ontology may consist of several ontologies depending on the set of considered services and the ontologies used to annotate them. The service provider may publish services from one, or multiple domains. In any case, we expect that the ontology used to annotate the published services is consistent, even if it covers several domains. Services from one domain may be published by multiple providers. These may use different ontologies to annotate the services. Hence, to compose services from different providers, we have to deal with ontology heterogeneity.

The examination of the correspondence between two ontologies is studied as a field of *ontology matching* [19]. Its aim is to provide methods enabling interoperability of systems using different ontologies. In the context of service composition, we exploit these methods to understand the semantics of services from different providers in the same, consistent way. Moreover, these methods are useful to correctly understand the user's desire expressed in the query. They are also used to estimate the semantic relation (distance) of two concepts associated with Web service elements [11, 54].

An interesting solution aiming at capturing the semantics of services is proposed by USDL (Universal Service-Semantics Description Language) [32]. This language has much in common with the other languages for semantic annotation of Web services. On the other side, it concentrates on the problem of creating an ontology which is understood by all the interested parties. The creation of standard domain ontologies had shown to be a hard issue. The authors of [32] propose that a universal ontology, including a sufficiently comprehensive set of concepts, is required. The ontology fulfilling this requirement is a *Lexical database for English – WordNet*.

To annotate Web services, a transformation of the WordNet into an OWL ontology is used[4]. Based on this, USDL lacks the semantic aliasing problem. Similarly to other languages, USDL also maps the parts of the WSDL service description to concepts of the ontology. In this case, the concepts are *basic concepts*, or con-

[4] `http://www.w3.org/TR/wordnet-rdf/`

cepts created by a conjunction, disjunction, and negation of basic concepts. The basic concepts stand as a contact point between USDL and the WordNet. This approach makes USDL a promising solution able to handle the problem of common understanding of semantics depicted by an ontology defined by different parties.

3 SEMANTIC WEB SERVICE COMPOSITION FORMALIZATION

3.1 Preliminaries

In our work the ontology is seen as a conceptual model of a reality, as it is usual. It provides a vocabulary of terms which are used to describe different aspects of Web services and user goals. If an artefact is annotated by an ontological element, it is expected that diverse systems understand its meaning in the same way. To make the definition of the ontology precise, several formal systems can be used. The most appropriate is the usage of description logic [4]. Description logic is a family of formal systems differing in the allowed set of operators affecting the expressivity level and the complexity of the reasoning. The reasoning is applied during service matchmaking and composition [5]. The conceptualization defined by description logic can also be expressed using first order logic [4].

We define the ontology by a set of concepts (also called classes) and relationships between them (also called properties). The world consists of elements. These are real objects (e.g. a credit card) appearing in the given domain with data attributes (e.g. credit card number) and states of these objects (e.g. the credit card is charged). Concept is a collection of elements sharing something in common. For example, the *Credit card* concept is a collection of all credit cards. The *Charged object* concept is a collection of all objects, which have been charged (not necessarily a credit card). An element may belong to several concepts, e.g. if a credit card is charged, it belongs to both *Credit card* and *Charged object* concepts. The elements may have characteristics which are seen as data attributes, e.g. the credit card has a number. This is depicted as the *hasNumber* property of the *Credit card* concept. The property defines a relationship between the *Credit card* and the *Credit card number* concepts.

To express the meaning of different aspects of Web services, or user goals, we bind them to the ontology. For example, a Web service charging a credit card is bound to *Credit card* and *Charged object* concepts. The same is true for a user goal to charge his/her credit card.

Definition 1 (Ontology)**.** Ontology O conceptualizing the world consisting from a set of elements E is a 3-tuple $\langle C, S, P \rangle$.

C is a set of unary predicates over E, depicting concepts, i.e. each $c \in C$ depicts one concept. The fact that an element $e \in E$ belongs to the concept c is denoted by the existence of $c(e), c \in C$.

S is a set of binary predicates over C depicting a subsumption relationship between two concepts, i.e. if $c_1, c_2 \in C$, $s = (c_1, c_2) \in S$ then c_1 subsumes c_2, denoted as $c_1 \sqsubseteq c_2$.

P is a set of binary predicates over E used to depict properties, i.e. each $p \in P$ depicts one property. The fact that there is a property p of element $e_1 \in E$, ranging in element $e_2 \in E$, is denoted as $p(e_1, e_2)$, $p \in P$.

A simple ontology, also presented in Figure 3, from a credit card domain may look as follows:

$$
\begin{aligned}
C &= \{\textit{Payment card}, \textit{Credit card}, \textit{Card number}, \textit{Charged object}, \textit{Charged amount}, \\
&\quad \textit{Stolen object}, \textit{Active object}, \textit{Expiration date}, \textit{Expired object}, \textit{Transaction detail}, \\
&\quad \textit{Transaction status}, \textit{Currency}\} \\
S &= \{\textit{Credit card} \sqsubseteq \textit{Payment card}\} \\
P &= \{\textit{chargedTo} \subseteq \textit{Transaction detail} \times \textit{Charged amount}, \\
&\quad \textit{isInCurrency} \subseteq \textit{Transaction detail} \times \textit{Currency}, \\
&\quad \textit{hasStatus} \subseteq \textit{Transaction detail} \times \textit{Transaction status}, \\
&\quad \textit{hasNumber} \subseteq \textit{Payment card} \times \textit{Card number}, \\
&\quad \textit{expiresAt} \subseteq \textit{Payment card} \times \textit{Expiration date}, \\
&\quad \textit{realizedBy} \subseteq \textit{Transaction detail} \times \textit{Payment card}\}
\end{aligned}
$$

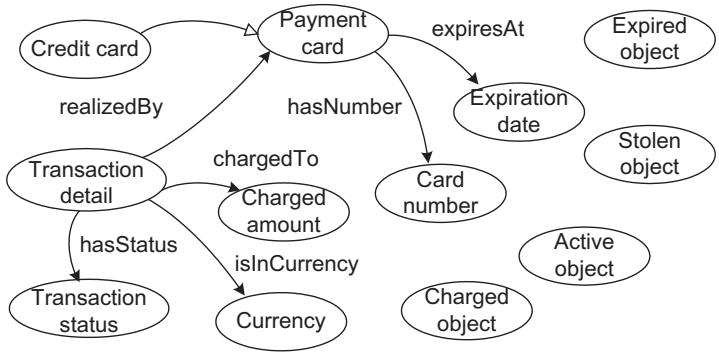


Fig. 3. Conceptualization of the credit card domain

The set C is a collection of terms of the conceptualized domain. The subsumption relation between these is meant to be reflexive, i.e. $\forall c \in C$, $c \sqsubseteq c$ and transitive, i.e. if $c_1 \sqsubseteq c_2 \wedge c_2 \sqsubseteq c_3$, then $c_1 \sqsubseteq c_3$. The subsumption is used to express the generalization/specialization relationship. The concept subsuming another one is a more specific term. The properties explicitly defined for some concept are also meant to be properties of all its subsuming concepts, i.e. if it is explicitly stated that $R \subseteq c_1 \times c_3$ and $c_2 \sqsubseteq c_1$, then R is implicitly also a property of c_2, $R \subseteq c_2 \times c_3$. When introducing a property, it is important to be aware of the meaning of the order of predicate arguments. The property is meant to hold for the first argument. The second is seen as a range of the property.

Definition 2 (Structural statement)**.** *Structural statement* over ontology $O = \langle C, S, P \rangle$ is any formula created as a composition of predicates from $C \bigcup P$, or other structural statements using logical operators $\wedge, \vee, \neg$, considering closed world assumption.

Instead of putting concrete elements to the predicate arguments, we use bound variables, representing some elements, according to the ontology. Conjunction depicts that each of the conjuncts holds. Disjunction expresses that some of the conjuncts hold. Negation is used to state that something does not hold.

Structural statement can be seen as a semantic model of certain reality in the world, focusing on its *structure*. We use it to depict the structure of a user goal, or I/O and pre-/post-condition of Web services. Note that from description point of view there is no significant difference between these.

An example structural statement is as follows: *Credit card*(*card*)$\wedge$*expiresAt*(*card*, *date*) $\wedge$ *Expiration date*(*date*) $\wedge$ *hasNumber*(*card*, *number*) $\wedge$ *Card number*(*number*). The *card* variable represents a credit card with expiration date *date* and its number is *number*. Another example is as follows: *Credit card*(*card*) $\wedge \neg$*Active object*(*card*) $\wedge$ (*Stolen object*(*card*) $\vee$ *Expired object*(*card*)). Here, the *card* represents an inactive credit card which was stolen, or has expired.

The structural statement proposes what holds for the elements which can replace the variables appearing in it. Using disjunction we define alternatives. The conjunction is used to express multiple facts about elements and their relationship, e.g. an element belongs to multiple concepts, or has several defined properties.

Next, we introduce value restrictions to the variables appearing in structural statements. The value restriction is expressed using *value restriction statement* defined as follows.

Definition 3 (Atomic value restriction statement)**.** Let us denote $\mathcal{O}$ as a set of operators. Any expression composed from variables, constants, and operators from $\mathcal{O}$, which can be evaluated as true or false after the variables are valued, is an *Atomic value restriction statement* over set $\mathcal{O}$.

An example set of operators is $\mathcal{O} = \{<, >, ==\}$. Using these we can create atomic value restriction statements depicting (in)equality such as

1. $a < b$, or
2. $a == b$,

which are evaluated as true if the value of a is

1. less,
2. the same

as the value of b.

Definition 4 (Value restriction statement)**.** Any statement composing atomic value restriction statements over set $\mathcal{O}$ using logical operators $\wedge, \vee, \neg$ is a *Value restriction statement* over set $\mathcal{O}$.

Let us denote the evaluation of the value restriction statement *Val* as $v(Val)$, i.e. $v(Val) \rightarrow \{true, false\}$. The value restriction statement *Val* is satisfied $v(Val) = true$, iff the variables are valued in such a way that the resulting statement is true. The variables appearing in the value restriction statements correspond to the variables from structural statements, or present local variables.

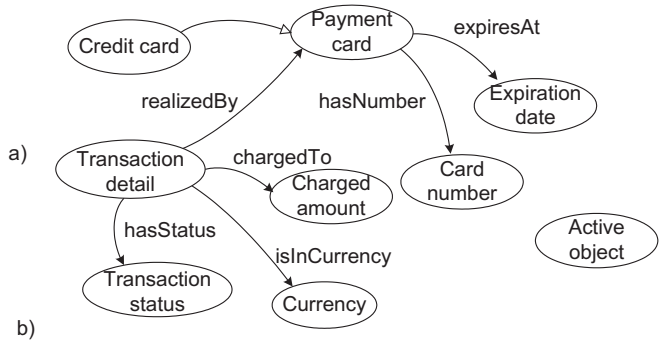
Definition 5 (Condition)**.** Condition C is a 2-tuple $C = \langle Str, Val \rangle$. *Str* is a structural statement depicting the structure of the condition. *Val* is a value restriction statement over variables appearing in *Str*.

The structural statements and value restrictions are interconnected through variables appearing in them. The variables in value restrictions are necessarily mapped to data fragments described at syntactic level, i.e. the possible values of the variables must suit the defined data type. The fact that variable v belongs to concept C and its data type is *dataType* is denoted as $C(v) \rightarrow dataType$.

An example condition $C = \langle Str, Val \rangle$ is defined as *Str*: *Transaction detail*(*transaction*) $\wedge$ *chargedTo*(*transaction*, *amount*) $\wedge$ *Charged amount*(*amount*) $\wedge$ *isInCurrency* (*transaction*, *currency*) $\wedge$ *Currency*(*currency*) $\wedge$ *hasStatus*(*transaction*, *status*) $\wedge$ *Transaction status*(*status*) $\wedge$ *realizedBy*(*transaction*, *card*) $\wedge$ *Credit card*(*card*) $\wedge$ *hasNumber* (*card*, *number*) $\wedge$ *Card number*(*number*) $\wedge$ *expiresAt*(*card*, *date*) $\wedge$ *Expiration date*(*date*) and *Val*: *amount* = 400.0 $\wedge$ *status* = *approved* $\wedge$ *date* = 2012-09-02T16:24:19.544 +02:00 $\wedge$ *number* = 5 588 320 123 456 789 $\wedge$ *currency* = *EUR*. The condition depicts that an active credit card with number 5 588 320 123 456 789 and expiration date September 2012 is successfully charged by the amount of 400.0 €.

Figure 4 depicts the correspondence between the syntactic and semantic description level of Web services in the context of our example condition. Figure 4 a) shows the ontology prescribing the possible predicates usable in the structural statements. Figure 4 b) presents the XML schema corresponding to the syntactic level defining the data types of variables appearing in the structural statements. Figure 4 c) depicts the fragment of the SOAP message with concrete data values. Considering for example the amount of money to which the credit card is charged, we see that at the semantic level it is described by concept *Charged amount*. The data type used to represent it is *xsd:double*. The value restrictions must be defined according to this data type, e.g. 400.0.

Definition 6 (Service)**.** Service S is a 4-tuple $S = \langle I, O, Pre, Post \rangle$. I/O is a list of inputs/outputs, i.e. variables based on concepts in the ontology and bound to data fragments described at syntactic level. *Pre* is a condition which must hold before service execution. *Post* is a condition which holds after service execution. Services are the elements of service repository R.




```
<complexType name="TransactionDetail">
 <sequence>
  <element name="realizedBy" type="impl:PaymentCard"/>
  <element name="chargedTo" type="xsd:double"/>
  <element name="hasStatus" type="xsd:string"/>
  <element name="isInCurrency" type="xsd:string"/>
 </sequence>
</complexType>
<complexType name="PaymentCard">
 <sequence>
  <element name="hasNumber" type="xsd:string"/>
  <element name="expiresAt" type="xsd:dateTime"/>
 </sequence>
</complexType>
<complexType name="CreditCard">
 <complexContent>
  <extension base="impl:PaymentCard">
  </extension>
 </complexContent>
</complexType>
```

c)

```
<transaction>
  <realizedBy>
      <hasNumber>5588320123456789<hasNumber/>
      <expiresAt>2012-09-02T16:24:19.544+02:00<expiresAt/>
  <realizedBy/>
  <chargedTo>400.0<chargedTo/>
  <hasStatus>approved<hasStatus/>
  <isInCurrency>EUR<isInCurrency/>
</transaction>
```

Fig. 4. Correspondence between syntactic and semantic service description

Definition 7 (Repository of Services)**.** Repository R is a set of available services. These can be used in the service composition.

Consider a simplified service charging a credit card to a defined amount:

- I: *card*, *amount* such that *Credit card*(*card*) $\rightarrow$ *type:CreditCard* and *Charged amount*(*amount*) $\rightarrow$ *type:double*
- O: *transDetail* such that *Transaction detail*(*transDetail*) $\rightarrow$ *type:Transaction-Detail*
- *Pre*: *Credit card*(*card*) $\wedge$ *hasNumber*(*card*, *number*) $\wedge$ *Active object*(*card*) $\wedge$ *Charged amount*(*amount*)
- *Post*: *Transaction detail*(*transaction*) $\wedge$ *hasStatus*(*transaction*, *status*) $\wedge$ *Transaction Status*(*status*) $\wedge$ *realizedBy*(*transaction*, *card*) $\wedge$ [(*Charged object*(*card*) $\wedge$ *chargedTo*(*transaction*, *chargedAmount*) $\wedge$ *Charged amount*(*chargedAmount*)) $\vee$ $\neg$*Charged object*(*card*)] and (*status* = *Approved* $\wedge$ *chargedAmount* = *amount*) $\vee$ *status* = *Declined*

The pre-condition prescribes what is necessary to hold before the Web service can be executed. It is an additional requirement to the inputs and their values. If these are not met, the execution cannot take place, or will result incorrectly. The post-condition prescribes what holds after the service execution. It is a true statement which does not have to be checked. Taking our example, after the service is executed, it holds that the transaction was approved and the credit card was charged by a given amount, or the transaction was declined and the credit card was not charged.

Definition 8 (Query)**.** Query Q is a 3-tuple $Q = \langle I', O', C' \rangle$. I' is a list of provided inputs. O' is a list of required outputs. Both I' and O' consists of variables ground to the concepts in the ontology. C' is a condition which is required to hold after execution of the service composition.

Definition 9 (Service composition)**.** Service composition is a directed acyclic graph $G = (V, E)$ prescribing the control-/data-flow of execution of services used in the composition. The nodes represent those services from the repository, which are meant to be executed, i.e. $\forall S \in V : S \in R$. The edges depict the control-/data-flow. If service S_{succ} should be executed directly after execution of S_{anc}, or S_{succ} takes as inputs the outputs of S_{anc}, there is an edge $e \in E$ such that $e = (S_{anc}, S_{succ})$, i.e. S_{anc} is chained with S_{succ}. Based on the complexity level of the service composition, additional requirements are applied to the composition.

Definition 10 (Final services)**.** Set of final services FS is a subset of all services in the service composition $G = (V, E)$, which do not have a successor, i.e. $FS \subseteq V$ such that $\forall S \in FS : \nexists S_{succ} \in V, \exists e = (S, S_{succ}) \in E$.

3.2 Service Compatibility During Composition

Web service composition can be stated as a problem of different complexities. The complexity is affected by several issues. It depends on the options offered to the user when expressing his/her goal and the overall objectives of the composition. These issues impose new requirements to the composition method and require different information about the services, or the ontology. At the basic level, the composition focuses on the core essences of the user goal and Web services. These are the data and states described by conditions.

The basic step during Web service composition is to decide if the service execution might result in data and effects required by the successor services, or in the user goal. The decision is made by checking the data and condition compatibility. From the data perspective, we examine *semantic type restriction*, i.e. if the service output data suit the required semantic type. To check condition compatibility, we need to verify *condition restrictions*, i.e. to check whether the service post-condition describes a state which is required to hold.

Next we formalize the problem of data and condition compatibility. Examples will be used from the domain of traveling. A simple conceptualization of the domain

is presented in Figure 5. To keep the visualization of the ontology simple, some details are not depicted.

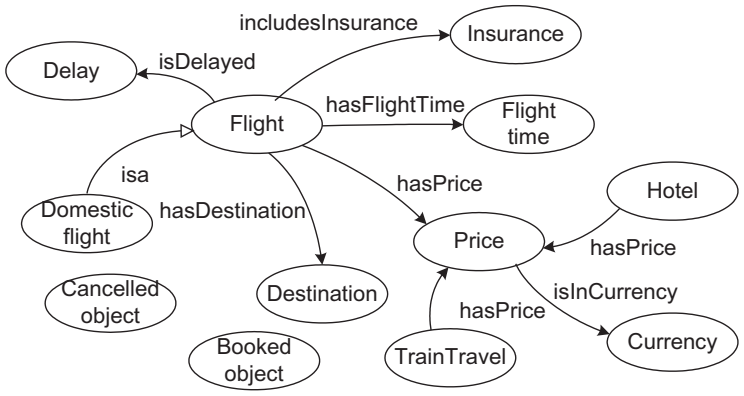


Fig. 5. Conceptualization of the traveling domain

3.2.1 Data Compatibility

At the simplest level, the semantics of the services and the user goal are expressed only by binding them to a concept defined in the ontology. No pre-/post-conditions, or goal condition are considered. In this case it is also enough to have a low level ontology defining only a set of domain terms. It is not required to propose properties of concepts. We define only a set of concepts. By binding them to the I/O of services, we define their meaning. Even the subsumption relations need not be defined. This means that the ontology is defined as $O = \langle C, S, P \rangle$, where $S = \emptyset, P = \emptyset$. For a service $S = \langle I, O, Pre, Post \rangle$, both Pre and $Post$ are empty. Analogically, the user goal is defined as a list of concepts related to provided inputs and required outputs, i.e. $Q = \langle I', O', C' \rangle$ where $C' = \emptyset$. An example goal is as follows: *The user wants a flight reservation.* Here, the user expresses that as a result s/he expects data which are of *Flight* semantic type.

Definition 11 (Type restriction aware service composition)**.** Service composition $G = (V, E)$, for a user goal depicted by query $Q = \langle I', O', C' \rangle$, $C' = \emptyset$, satisfies type restrictions if the following holds:

1. Each service $S = \langle I, O, Pre, Post \rangle$, $Pre = \emptyset, Post = \emptyset$, $S \in V$, in the service composition must have provided each input $i \in I$. The input is provided in the query, or as an output of ancestor service $S_{anc} = \langle I_{anc}, O_{anc}, Pre_{anc}, Post_{anc} \rangle$, $Pre_{anc} = \emptyset, Post_{anc} = \emptyset$. Each input i must subsume its provider ip. $\forall S \in V, \forall i \in I, \exists S_{anc} \in V, \exists ip \in O_{anc} \bigcup I'$ such that $ip \sqsubseteq i$.
2. Each required output from O' is provided by some service $S \in V$, i.e. $\forall o' \in O', \exists S, \exists o \in O$ where $o \sqsubseteq o'$.

3.2.2 Condition Compatibility

The pre-/post-condition aware service composition brings the option to describe a state the user wants to achieve. An example goal is defined as follows: *The user wants to book a flight.* In this case, the user goal is described as a condition which is required to hold. The aim of the composition is to design a composite service having a post-condition implicating the goal condition. In general, the conditions prescribe several alternatives from which some holds after the service execution. Consider two conditions C_1, C_2 for which we evaluate the implication, i.e. if C_1 implicates C_2. The condition C_2 may implicate

1. none of the alternatives of C_1,
2. some of the alternatives of C_1, or
3. each alternative of C_1.

In the first case we are sure that the execution of C_1 cannot result in a state suiting C_2. The second case is a situation when the execution may result in a state suiting C_2. In the third situation we are sure that the execution will result in a suitable state. In the next, we formalize these statements.

Definition 12 (Weak condition implication)**.** Let us denote the set of subformulae representing the respective conjunctions of the formula F, transformed to DNF, as F_{DNF}. Condition $C1 = \langle Str1, Val1\rangle$ weakly implicates condition $C2 = \langle Str2, Val2\rangle$, iff the following holds:

1. There is a conjunction $str1 \in Str1_{DNF}$ and conjunction $str2 \in Str2_{DNF}$ for which we can find a unique, one-to-one mapping between variables in $str2$ to variables in $str1$ such that we can get a formula $\widehat{str1}$ implying $str2$ as a result of
   (a) substituting the variables in $str1$ based on the mapping with the variables in $str2$ and
   (b) substituting the unary predicates in $str1$ to predicates subsumed by them, i.e.: $\exists str1 \in Str1_{DNF}, \exists str2 \in Str2_{DNF}$ such that $\exists\widehat{str1}, \models \widehat{str1} \Rightarrow str2$.
2. The value restriction of $C2$ is not violated by the value restriction of $C1$, i.e. the mapped variables in $Val1$ and $Val2$ can be valued to the same values in such a way that if $v(Val1) = true$ then also $v(Val2) = true$.

Definition 13 (Strong condition implication)**.** There is a strong implication between conditions $C1 = \langle Str1, Val1\rangle$ and $C2 = \langle Str2, Val2\rangle$, iff the following holds:

Point 1. from Definition 12 holds for each $str1 \in Str1_{DNF}$, i.e. $\forall str1 \in Str1_{DNF}$, $\exists str2 \in Str2_{DNF}$ such that $\exists\widehat{str1}, \models \widehat{str1} \Rightarrow str2$.

Point 2. from Definition 12 holds for any valuation of variables in $Val1$ and $Val2$.

Condition $C1$ implicates condition $C2$ if there is a weak or strong implication between them. The fact that $C1$ implicates $C2$ is denoted as $C1 \Rrightarrow C2$.

Definition 14 (Condition restriction aware service composition)**.** Let us denote the set of final services of the composition as $FS = \{S_1, \ldots, S_n\}$. Let $FStr$ be a structural statement created as a conjunction of all structural statements of the final services, i.e. $FStr = Str_1 \wedge \ldots \wedge Str_n$. Analogically, let $FVal$ be a value restriction statement created as a conjunction of the value restriction statements of all final services, i.e. $FVal = Val_1 \wedge \ldots \wedge Val_n$. The resulting condition of the composition is $FC = \langle FStr, FVal \rangle$. Service composition $G = (V, E)$, for a user goal depicted by query $Q = \langle I', O', C' \rangle$, satisfies condition restrictions iff the following holds:

1. For each pair of services $S_{anc} = \langle I_{anc}, O_{anc}, Pre_{anc}, Post_{anc} \rangle$, $S_{succ} = \langle I_{succ}, O_{succ}, Pre_{succ}, Post_{succ} \rangle$ which are chained, it must hold that the postcondition of the ancestor service implicates the precondition of the successor service, i.e. $\forall e = (S_{anc}, S_{succ}) \in E, Post_{anc} \Rrightarrow Pre_{succ}$.
2. FC is implicated by C', i.e. $FC \Rrightarrow C'$.

Based on the set of operators used in the value restriction statements, the process of evaluating the value restrictions is of different complexity. The simplest case is when variables are restricted to constant numeric or string values. Here, we have to check whether the mapped variables from the considered conditions are valued to the same value. More complex evaluation is required if we use inequality operators $\neq$, $>$, $<$, $\geq$, $\leq$, and if we use a variable in the value restriction which is defined based on other variables, e.g. $price = flightprice + hotelprice$.

3.3 Quality of Service

The I/O, pre-/post-conditions of services are used to describe their behavior. They express their functional properties. During Web service composition, these are necessary to consider design a composite service fulfilling the user goal. Beside the functional properties of the composite service, the user may be interested also in some non-functional properties determining the quality of the service (QoS) [59, 58]. The usual QoS attributes of services are response time, throughput, availability, price, and reliability. The values of these are monitored, measured and stand as a subject of a contract between the service provider and consumer. When composing services, the QoS attributes of services are taken into account to design a composite service maximally satisfying also the non-functional requirements. The requirements define hard and soft constraints. Hard constraints define requirements which must be satisfied. Soft constraints define preferences over the QoS attributes.

In the context of QoS-driven composition we usually consider only properties which can be expressed numerically. The aim is to select the composite service having the best overall quality characteristic from all candidates. Depending on the

concrete attribute, the best means the highest, or the lowest value (e.g. for response time, the lower value is better, but considering throughput, the higher value is better). To have a uniform view such that the higher value is better, the attributes for which the lower value is better are multiplied by -1.

Calculation of the overall quality of the composition includes

1. calculation of the particular characteristics for a composition from the QoS values of single services,
2. calculation of the uniform quality representative.

Calculation of the particular characteristic for the composition is based on aggregation functions [59, 58, 3, 34]. These define how we calculate the aggregated quality of a given characteristic regarding the composite service structure.

Calculation of the uniform quality representative is based on a utility function. It calculates one representative value from all the aggregated property values of different unit and range, i.e. we evaluate the multi-dimensional quality with one value. This value is used to rank the compositions. The calculation of the utility function is usually based on *Multiple Attribute Decision Making* method [56]. This method is based on scaling and weighting [59, 3, 48]. Scaling allows a uniform measurement of multi-dimensional attributes, independently on their units and ranges. Weighting allows expressing preferences over different quality attributes.

Definition 15 (Aggregated QoS characteristics)**.** Aggregated QoS characteristics of a composite service $G = (V, E)$, for a given characteristics q, is a number, calculated based on the corresponding aggregation rules for the composition G. It is denoted as $QoS^{\blacklozenge}(G, q)$.

Definition 16 (Uniform QoS characteristics)**.** Uniform QoS characteristics of a composite service $G = (V, E)$, is a number, calculated from the aggregated QoS characteristics $QoS^{\blacklozenge}(G, q)$. It represents a uniform QoS characteristics of the service composition where the higher value is better. It is denoted as $QoS^{\rightarrowtail}(G)$.

Definition 17 (Optimal composite service)**.** A composite service $G = (V, E)$ is optimal, denoted as $QoS^{+}(G)$, if $QoS^{\rightarrowtail}(G)$ is the best considering any other composite service $G' = (V', E')$, i.e. $\nexists G'$ such that $QoS^{\rightarrowtail}(G') > QoS^{\rightarrowtail}(G)$.

4 OVERVIEW OF EXISTING APPROACHES

4.1 QoS Focusing Approaches

In [7, 9, 8] and [23, 25], two approaches which took a part at *Web Services Challenge 2009* are presented. *Web Services Challenge* is a competition aimed at developing software components and/or intelligent agents that have the ability to discover pertinent Web services and also compose them to create higher-level functionality[5].

[5] `http://ws-challenge.georgetown.edu/wsc09/index.html`

In 2009 it focuses on automatic web service composition considering the QoS [29]. Both [7] and [23] proved to be scalable approaches. Even during the hardest data set at the competition, consisting from 15 000 Web services, they were able to find a solution in acceptable time[6] (below 300 milliseconds). Both approaches realize preprocessing during which effective data structures are built. These are used during the user querying phase to quickly compose a desired composition. The approach in [7] benefits from a data structure based on a relational database and parallel process execution. Composition in [23, 25] is effective due to a filtering utilized to reduce the search space. The pruning removes services

1. which have no inputs (thus cannot be executed and used in the composition) and
2. which are not optimal from QoS point of view.

The removal of unusable services is applied also in [7].

Another approach to QoS aware automatic Web service composition dealing with scalability is described in [3]. Unlike [7] and [25], this approach does not deal with design of the structure of the composite service. As an input it already takes an abstract service, i.e. the aim is only to select concrete services for each service class used, which is the best considering the QoS characteristics. The previous approach [25] finds the best plan for each QoS characteristics separately. This approach uses a utility function to express the uniform QoS characteristics. Calculation of the utility function is based on *Simple additive weighting* technique. It involves scaling of the quality attribute values to allow uniform measurement independently on the unit and range of the given attribute. Then, weighting process follows to represent user priorities and preferences. In this context the approach deals with the scalability issues. It is based on a heuristic algorithm decomposing the original optimization problem into sub-problems which can be solved more effectively. The decomposition allows finding the best candidate for each service class separately, i.e. it is not required to check all the possible service combinations. Before this, only two global parameters for each attribute must be calculated. After this, calculations are performed locally for each service class. Based on this, the approach presents good scalability when finding near-to-optimal solution, according to the number of service classes and candidates per class.

The same problem as approach in [3] deals with, is the objective of the work presented in [48]. The aim of the approach is a large-scale QoS aware service composition. The used composition model allows a flexible specification of the QoS constraints based on hierarchies. The authors present a metaheuristics-based optimization approach able to find a near-to-optimal solution, even in large-scale situations. Similarly to [3] and [59], here also the authors use simple additive weighting to find the uniform QoS characteristics. Both approaches presented in [3] and [48]

[6] The time includes a call of the composition system from a client application, composition and result transformation to BPEL format, and realization of a callback from the composition system to submit the result to the client application

show promising results in the performance context. Due to incomparable test sets, it is not possible to properly compare their effectiveness.

Almost all approaches consider two services chainable, if there is a subsumption relation between the output and input parameters. In [34, 35] the authors use a notion of *semantic link* to denote the service chain. They define five semantic matching types:

1. *Exact*, i.e., $\models \mathcal{O} \equiv \mathcal{I}$
2. *PlugIn*, i.e., $\models \mathcal{O} \sqsubseteq \mathcal{I}$
3. *Subsume*, i.e., $\models \mathcal{O} \sqsupseteq \mathcal{I}$
4. *Intersection*, i.e., $\models \neg(\mathcal{O} \sqcap \mathcal{I} \sqsupseteq \bot)$
5. *Disjoint*, i.e., $\models \mathcal{O} \sqcap \mathcal{I} \sqsupseteq \bot$.

The usual subsumption relation explored by other approaches corresponds to the *PlugIn* matching type. Two services can be directly chained if there is an *Exact*, or *PlugIn* matching type between the I/O. Other approaches do not chain services if there is another relation. In [34] the services can also be chained in the case of *Subsume* and *Intersection*. In this case the output does not provide all information required as input. To define the missing information, *Concept abduction* is used [33]. It produces a description of the information which is missing and should be provided. Although the idea of *concept abduction* is nice, it is not clear if it is practical to realize it. The authors do not discuss how the missing information is provided to ensure correct data flow. One option could be their acquisition from the user. During composition, the proposed approach also considers the QoS. It introduces also a function combining the quality of the semantic link and non-functional properties to evaluate the overall quality of the composition. Using a *Hill climbing* based algorithm, they search for a solution satisfying the user goal at certain level, i.e. they do not guarantee the optimal solution.

4.2 Approaches Exploiting Additional Meta-Data

In [49] the authors claim that it is not enough to chain services by considering their I/O only. They state that composite services produced this way cannot guarantee the required functionality. Their solution is based on explicit definition of functional semantics of the services. It is done by description of the Web service functionality with an action-object pair (the action is meant to be performed over the object). The actions are mapped to concepts in a special part of an ontology – *Domain-action*. The objects and I/O of services are mapped to another part called *Domain-data*. By this extension of service descriptions the authors ensure functionality in compliance with user expectations. We believe that all this can be solved by introducing the pre-/post-conditions of services as we see in several approaches. At least the motivating examples in the paper can be easily solved by involving

this approach. The additional information about service in a form of action-object annotations can be transformed to simple pre-/post-conditions consisting from one predicate. Moreover, we believe that the aspect of pre-/post-conditions is more expressive.

In [20] the authors present a composition approach handling so called user constraints. These represent value restriction constraints to input, output, or local service parameters. To describe the constraints a KIF language is used. For binding the constraints to service parameters they have extended OWL-S. Our suggestion is to use the existing construct of OWL-S for this purpose, namely *process:hasPrecondition* and *process:hasEffect*. We believe that expressivity of the user constraints in the approach is the same as in [6]. However, the overall approach has a differing composition problem definition as in [6] and [30]. It seems that the problem neglects the design of the composite service structure, i.e. the set of used services and the control-/data-flow are partially known (similarly as in [3]) and the related issues are not a part of the problem solving. For each service the user may define value restrictions over the service parameters. The approach takes care about satisfying these constraints during Web service execution. Before the execution, it is checked whether the input data hold the constraints to input parameters. Constraints to output parameters are checked after execution. If these do not hold, the user is informed about the failure or a backtrack mechanism is realized. The user may also define multi-service restrictions. Such constraint is checked for first services. After it is executed, the value of the constraint is updated and used for checking the next services. Unlike [6] and [30], this approach only takes care of conditions defined by the user. It does not deal with the pre-/post-conditions of services during chaining.

In [28] a composition tool called OWLS-Xplan is presented. It uses OWL-S Web service descriptions converted to *Planning Domain Description Language* – PDDL 2.1 and an artificial intelligence planner called *Xplan* to generate a composition. The service descriptions include also pre-/post-conditions, which are simple conjunctions of predicates. Xplan extends an action based FastForward-planner (graph based planning) with HTN planning and re-planning component. Xplan consists of several modules for preprocessing and planning. Preprocessing includes creation of the required data structures, generation of the initial connectivity graph, and goal agenda. The planning consists of two interleaving activities: the heuristically relaxed graph-plan generation and enforced hill-climbing search. The heuristics approximates the distance between the current state and the goal state. This is used to guide the forward search process. The algorithm is a forward search with the following steps. First, the distance between the starting state and the goal state is computed. Second, the set of helpful actions is determined. Third, enforced hill-climbing analyzes all reachable states. If a better state is found, it is included into the plan and used for next search. The search terminates if the goal state is reached. If there are several actions with the desired effects, heuristics is used to select the one which will be used. The resulting heuristic goal distance is simply a sum of all actions in the relaxed plan [22].

In [50] a prototype of a semi-automatic composer is presented. It has two basic components: a composer and an inference engine. The inference engine stores information about the available services in a knowledge base. It also provides a way to find matching services. The composer presents a user interface to the inference engine. The inference engine is an OWL reasoner built on Prolog and is used to create entailments in the knowledge base. The composer enables creating the workflow by interacting with the user. In [50] the composition starts by selecting one of the available services. Based on the knowledge about the inputs for this service a query is sent to the knowledge base to get a list of services providing the input data. The user has the possibility to define constraints on the attributes of a service to filter the list of offered services and choose the most appropriate one.

The composer presents options for the composition by allowing choosing from the fitting services. This is based on the information from the service profile. Two types of matches between the I/O are defined: *exact* and *generic* match. The exact match means that the parameters are restricted to the same OWL class. The generic match means that the output type of the found service is a subclass of the input type of the selected service. In the list of matching services, the services with exact match are placed at the top.

The presented prototype is one of the earliest implementations of a Web service composer. Its main disadvantage is the level of automation of the composition. In [51] the authors propose an approach using artificial intelligence planning to automate the Web service construction process. They use a system called SHOP2 [41] based on hierarchical task networks.

SHOP2 automatically composes Web service described in OWL-S. The authors state that OWL-S does not have control constructs to flexibly describe abstract processes. To overcome this problem, they use its extension including a definition of a new process type. We believe that OWL-S provides enough constructors to define the service at abstract level. To compose services SHOP2 uses hierarchical task network planning technique. The planning algorithm starts with one or more tasks. These are decomposed until each of them is decomposed to an operator. The resulting plan is a sequence of operators which can be executed to achieve a used defined goal.

In the approach proposed in [31], the USDL language is used to specify the formal semantics of services [32]. This OWL based language uses WordNet as a common basis for understanding the meaning of services. In [30] the authors extend the notion of composition presented in [31] to handle non-sequential conditional composition. The pre-/post-conditions are expressed as atomic statements combined with conjunction, disjunction, and negation. The approach focuses also to other important features of a composition system. One of them is the ability of *incremental updates*. In dynamic world the Web service set used during the composition may change. Services may be added/removed. The composition system should react to these changes quickly. This usually cannot be achieved if the change requires a complete reconstruction of the internal service repository. It is desired to handle the change by adding new data to the repository affecting only a local part of it.

Another approach to pre-/post-condition aware service composition is described in [6, 9, 8]. It was already introduced as a QoS aware service composition approach. From the expressivity of pre-/post-conditions point of view, this approach is the same as [30]. To make the processing of the pre-/post-conditions effective, during the preprocessing, several characteristics of the conditions are precalculated and encoded. The encodings are then used to quickly evaluate the compatibility between two conditions. This approach showed to be effective enough to deal also with conditions of non-trivial complexity. Similarly to [30], this approach is also able to react to the changes in the service set quickly, by performing only a local change in the internal service registry [9].

4.3 Approaches Considering User Preferences

In [37] the user may define his/her preferences over generic and also domain specific service quality criteria. The preferences are formalized as fuzzy sets. Using fuzzy expressions, the user may also define trade-offs among the criteria. Overall user preference can be seen as a disjunction of conjunctions of preferences over a quality attribute, i.e. a disjunctive normal form over atomic fuzzy expressions. The atomic expression is expressed as a fuzzy set. The user may select one from five predefined sets expressing satisfaction from poor to extreme. The composition algorithm is based on *Depth-first Branch* and *Bound Method* methods. Instead of looking for a best solution (highest satisfaction degree), it finds a good enough solution saving much composition time. The user may define a minimal satisfaction degree which must be fulfilled by the found composition. To reduce useless search, the authors propose a simple uncritical consistency checking algorithm. However, from the experiments realized by authors, it is shown that if looking for a first acceptable solution, when the minimal satisfaction degree is high, the algorithm without consistency checking is much better in most cases.

Similarly to [37], the authors of [1] also deal with fuzzyfication of user constraints over QoS attributes. In this approach the user may define his/her preferences using fuzzy IF-THEN rules. The fuzzy rule expresses which combination of attribute values the user is willing to accept at which satisfaction degree. The user may combine atomic preferences, expressed as fuzzy sets, using fuzzy conjunction, disjunction, and negation. The user should define at most as many rules as many degrees of satisfaction s/he wants to differentiate. Unlike [37], the aim of this approach is to rank the solutions fulfilling the user goal based on the satisfaction degree. For each composition, based on the calculated aggregated quality attributes it calculates the degrees of fulfillment of the rules (i.e. the rule is interpreted and the results are aggregated as usual in IF-THEN rule based fuzzy optimization). At this point, we get a fuzzy representation of the result. After this, deffuzyfication is applied to get a number representing the overall satisfaction degree. This degree determines the position of the particular composition in the ranking. Unlike [37], the authors of [1] do not deal with performance issues of their approach. Hence, the practical applicability due to performance and scalability issues is unknown. From the user perspective,

the approaches offer similar opportunities. However, the approach from [1] performs more complex calculations to determine the satisfaction degree. The approach in [1] does only a part of these. It is questionable whether the more complex approach brings more appropriate results. Our opinion is that from practical usability point of view the simpler approach is sufficient and more applicable.

In [38] the authors present a hierarchical task network based automatic Web service composition approach satisfying user preferences. It does not try to satisfy them necessarily in absolute manner. If this is not possible, it looks for a composition satisfying the user preferences as much as possible. The user may express constraints over the states and the state trajectory corresponding to the plan using a special preference language. It allows to define *basic* and *temporal* preferences. Basic preference is a first order logic formula. Temporal preferences define additional restrictions over basic preferences. Let us denote the basic preference as $\mathcal{BP}$. Using a temporal preference the user may express that:

1. $\mathcal{BP}$ must hold in each state,
2. $\mathcal{BP}$ must hold in some state,
3. $\mathcal{BP}$ must hold at most once,
4. $\mathcal{BP}$ must hold in the final state,
5. $\mathcal{BP}$ must hold after some state, or
6. $\mathcal{BP}$ must hold before some state.

The work mainly describes how these kinds of preferences augment service composition and how are they mapped into a planning language for HTN. They also present a *Best-first* search based planning algorithm which showed to be useful during experiments realized by the authors.

In [26] the authors present a composition framework allowing modeling and scheduling composite Web services under user constraints. In this case, the composition starts with a manual modeling of the abstract workflow using a graphical interface. The abstract workflow includes abstract services depicting template of the service which must be used at this point. After this, the framework finds concrete services to create a composite service fulfilling the user constraints. The user may define value restriction constraints with IF-THEN rules. Moreover, dependencies between services may be defined. First, the dependency may depict that if service S_1 executes, S_2 must execute as well. The second kind of dependency may state that if abstract service AS_1 is done by S_1, the *same/different* service must be used to do the abstract service AS_2. To satisfy the constraints, the approach uses an external tool *Choco*. It is a Java library for constraint satisfaction problems. The experiments realized by authors show good scalability of the approach regarding

1. the number of candidate services,
2. flow complexity – the number of AND, OR, XOR, sequence and iteration blocks, and
3. the number of variables and constraints.

In [36] an approach to personalized Web service composition is presented. It considers hard and also soft constraints during composition. Hard constraints are value restriction statements. Soft constraints express the user preferences. To order the compositions based on user satisfaction, the Pareto dominance principle is applied. Composition $C1$ pareto dominates composition $C2$, iff there is an atomic preference satisfied more by $C1$ than by $C2$ and there is no atomic preference satisfied more by $C2$. A composition is said to be pareto dominant, iff there is no other composition pareto dominating it. The principle is implemented introducing a relaxation degree of the composition. It expresses the level of satisfaction. The proposed algorithm finds the composition with the minimal relaxation degree and thus with the highest satisfaction degree. It is proved in the paper that such a composition is pareto dominant.

5 DISCUSSION

5.1 Achievements

Table 1 summarizes selected characteristics of the presented approaches. We depict whether the approach deals with the design of the control-/data-flow, whether it considers additional information about the Web services beside the I/O, whether it is QoS-driven, and whether it calculates a uniform representative of the QoS attribute values. We also show whether the approach aims to find an optimal solution from the QoS point of view, whether performance evaluation is presented, and whether it deals with the dynamic changes in the service environment.

Approach	Control-/data-flow design	Functional extension	QoS	Uniform QoS	Optimal plan	Performance evaluation
Bartalos et al. [9, 8]	Yes	Yes	Yes	No	Yes	Yes
Huang et al. [23, 25]	Yes	No	Yes	No	Yes	Yes
Alrifai et al. [3]	No	No	Yes	Yes	No	Yes
Rosenberg et al. [48]	No	No	Yes	Yes	No	Yes
Lécué et al. [34]	No	No	Yes	Yes	No	Yes
Shin et al. [49]	Yes	Yes	No	–	–	Yes
Gamha et al. [20]	Yes	No	No	–	–	No
Klusch et al. [27, 28]	Yes	Yes	No	–	–	Yes
Sirin et al. [51]	HTN	Yes	No	–	–	No
Kona et al. [31, 30]	Yes	Yes	No	–	–	Yes
M. Lin et al. [37]	No	No	Yes	Yes	Yes	Yes
Agarwal et al. [1]	No	No	Yes	Yes	Yes	No
N. Lin et al. [38]	HTN	Yes	No	–	–	No
Karakoc et al. [26]	No	Yes	No	–	–	Yes
Li et al. [36]	HTN	Yes	No	–	–	Yes

Table 1. Overview of selected properties of composition approaches

From the semantic compatibility at the I/O level point of view, the approaches evaluate the semantic relation between the concepts associated with I/O. Most of the approaches consider the exact match and the subsumption relation. A few approaches look also for other relations such as plug-in, intersection, and disjoint

relations. The problem is that these are not sufficient to get the desired results. Additional data must be provided to absolutely satisfy the demands. The identification of the missing data is done by concept abduction. Nevertheless, the provision of the data is a complication which must be solved to make the approach applicable.

The semantics of services is crucial to make the automatic composition feasible and to get meaningful results. There is a consensus that the semantics is necessary. However, it has not been decided yet what level of semantics is sufficient. It has already been shown that considering only the semantics of the I/O is not enough. Since the aim of semantics is to capture the behavior of services, more complex information is required. It seems that for most services and a lot of practical composition scenarios, good results can be achieved when also pre-/post-conditions are used to express the functionality of the services. The formalism suitable to describe these conditions is a predicate logic. However, the pre-/post-conditions aware composition achieves much better results, the consideration of the conditions involves more complicated processing. Despite of this, handling the conditions is feasible and required in practical scenarios.

Beyond the functional properties of services, also the non-functional properties are necessary to deal with. The QoS-driven service composition has a lot of attention. In addition to the functional requirements, the user may specify also some constraints and preferences regarding the non-functional properties. From the non-functional properties, usually the approaches consider the technical QoS such as response time, availability, and throughput. The user may define hard and also soft constraints over multiple properties. Moreover, the requirements do not have to be defined with absolute precision. Fuzzy approaches are exploited to allow more vague definition of the user demands. This is very important from the user point of view. To express the requirements in a more vague form is more convenient and feasible. The approaches dealing with satisfaction of the non-functional requirements can be usually generalized to any attributes which can be expressed in a numerical form. They are not delimited to the technical QoS. In total, these approaches are basically used to optimize the composition. In general, multiple compositions satisfy the functional requirements. Consideration of the non-functional properties allows to select the best, or the near-to-optimal solution. The QoS-driven service composition showed to be important and several approaches show that the additional requirements can be handled effectively.

The overall composition process, including the processing of semantics and consideration of the QoS with optimization, involves a lot of computations to do. Several issues require calculations which rise exponentially regarding certain parameters. This makes the process NP-hard. Moreover, as the Web grows, also the number of available services rises. There are several public service repositories consisting of thousands, or even tens of thousands of services and their count is raising. Due to this, several approaches care about the performance and scalability. Currently, the approaches achieve good experimental results even when considering repositories consisting from thousand up to hundred thousand services. This is mainly true for the QoS-driven approaches which do not deal with the pre-/post-conditions. Only

a little attention is devoted to the effectiveness of the pre-/post-condition aware composition.

Since the Web in general and also the service change over time, the Web service composition system must adapt to the evolution of the environment it operates in. Most of the composition approaches are based on different data structures storing information about the service set. The change of these is required as the service environment evolves. Due to this, the data structures should be designed in such a way that they can be easily updated. There are only a few works addressing this problem and no detailed evaluation of the effect of the changes to the overall composition time has been presented.

5.2 Open Research Areas

Web services design. As far as we know, there is no methodology describing how should the Web services be aimed to be automatically composed and designed. It is quite obvious that the design of a Web service significantly affects the ability to use it during the composition. The most important aspects include granularity and the interface of the domain services. Coarse-grained services, providing complex functionality, could potentially not fit all the requirements. Fine-grained services provide higher variability and allow designing a composition fitting the requirements better. On the other side, the composition process is more challenging in this case.

Semantics. The approaches assume in general that all the required semantics is available and correctly defined in a desired form. It is known that to make the idea of the Semantic Web and the Semantic Web services a really working and beneficial implementation, the semantic methods must provide much higher precision in recognition of the semantics made by distinct parties. Creation of the ontologies, semantic annotation, and maintenance of the semantics must be supported by tools and methodologies must be developed to make the whole idea practically manageable. More attention should be devoted to the pre-/post-conditions. It has been shown that they are necessary to precisely define the functionality of the service; however, their description and management is more difficult than considering the I/O only.

Service composition utilization. Only a few works have been devoted to the way how dynamic service composition can be utilized by the user. Most approaches do not deal with this and focus only on the phase when the user goal is already known. Service composition allows different ways how to exploit it. One option is to use the composition system as an integral part of a complex system to realize tasks, which are not suited to be hard-coded. Another option is to develop an application standing as a mediator between a human and the composition system. Its task is to

1. transform the user goal of the human to the representation required by the composition system,

2. invoke the composition system, and
3. present the results to the user.

An idea of provisioning dynamic composition of services is introduced as *Composition as a Service* (CaaS) [47, 16]. Its aim is to reduce the complexity of developing a composite application by providing dynamic composition features. CaaS allows dynamically composing and deploying composite services based on a specification. The CaaS concept is useful in situations when a composite application is being developed and realization of some parts of the application is not clear to its designer. In this case, s/he can specify the requirements and use the CaaS to design the missing part.

Killer application. The research of Web service composition has started a quite long time ago already and numerous problems have been solved already. However, the research was motivated rather by artificial scenarios far-away from the reality. We believe that the current methods are sophisticated enough, so we should try to apply them in scenarios coming from practice. Business process management and SOA are here to help applying information technologies in a changing world, where the business processes evolve in time to adapt to the changing demands. It should be studied how the dynamic Web service composition could be applied in this context. The research attention should try to look for a killer application, showing the real benefits of the dynamic Web service composition.

6 CONCLUSIONS

As it is true and natural for any young branch of research, also the area of Web service composition is currently bringing solutions for more or less isolated problems: I/O and pre-/post-condition semantic compatibility, optimization according to non-functional properties, performance, user context and personalization, and transactional behavior of service compositions. To draw the whole picture and make a practical utilization of service composition a reality, several other problems and their dependencies must be addressed.

We suppose that service composition should be studied also from software engineering point of view. A methodology must be developed describing the process of Web service design, semantic annotation, maintenance, and the utilization of Web service composition. Tools supporting these tasks must be provided. Currently, it is not clear in which way is it beneficial to exploit service composition. We do not know if it can be applied only in closed domains, if we really can effectively use public services or must rely on private ones only. Research attention should be devoted to find a killer application showing the real benefits and limitations of service composition in practice.

It is questionable whether a more visionary exploitation of the service composition is feasible. In this scenario a software agent, acting on behalf of a human,

having knowledge about the human, automatically constructs a composition goal and utilizes service composition to satisfy it. This agent should be a delegate who actively uses the possibilities of service composition to continuously achieve user satisfaction. This agent must maintain a user model of the human and know its actual context.

Acknowledgement

This work was supported by the Scientific Grant Agency of Slovak Republic, grant No. VG1/0508/09 and it is a partial result of the Research and Development Operational Program for the projects Support of Center of Excellence for Smart Technologies, Systems and Services, ITMS 26240120005 and Research and Development Operational Program for the projects Support of Center of Excellence for Smart Technologies, Systems and Services II, ITMS 26240120029, co-funded by ERDF.

REFERENCES

[1] Agarwal, S.—Lamparter, S.: User Preference Based Automated Selection of Web Service Compositions. In ICSOC Workshop on Dynamic Web Processes, 2005, pp. 1–12.

[2] Agarwal, V.—Chafle, G.—Mittal, S.—Srivastava, B.: Understanding Approaches for Web Service Composition and Execution. In COMPUTE '08: Proceedings of the 1st Bangalore Annual Compute Conference, New York, NY, USA, ACM 2008, pp. 1–8.

[3] Alrifai, M.—Risse, T.—Dolog, P.—Nejdl, W.: A Scalable Approach for Qos-Based Web Service Selection. In Service-Oriented Computing 2008 Workshops, Springer-Verlag, 2009,.pp. 190–199.

[4] Baader, F.—Calvanese, D.—McGuinness, D. L.—Nardi, D.—Patel-Schneider, P. F. (Eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA 2003.

[5] Babík, M.—Hluchý, L.: Optimizing Description Logic Reasoning for the Service Matchmaking and Composition. Computing and Informatics, Vol. 27, 2008, No. 4, pp. 681–698.

[6] Bartalos. P.—Bieliková, M.: 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, pp. 414–421.

[7] Bartalos, P.—Bieliková, M.: Semantic Web Service Composition Framework Based on Parallel Processing. In Int. Conf. on E-Commerce Technology, IEEE CS, 2009, pp. 495–498.

[8] Bartalos, P.—Bieliková, M.: QoS Aware Semantic Web Service Composition Approach Considering Pre/Postconditions. In Int. Conf. on Web Services, IEEE CS 2010, pp. 345–352.

[9] Bartalos, P.—Bieliková, M.: Effective QoS Aware Web Service Composition in Dynamic Environment. In: Int. Conf. on Information Systems Development, Springer 2010.

[10] Bartalos, P.: Effective Automatic Dynamic Semantic Web Service Composition. Information Sciences and Technologies Bulletin of the ACM Slovakia, Vol. 3, 2011, No. 1, pp. 61–72.

[11] Bellur, U.—Kulkarni, R.: Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching. IEEE International Conference on Web Services, 2007, pp. 86–93.

[12] Bellur, U.—Vadodaria, H.: On Extending Semantic Matchmaking to Include Preconditions and Effects. In ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services, Washington, DC (USA), 2008, pp. 120–128.

[13] DiBernardo, M.–Pottinger, R.—Wilkinson, M.: Semi-Automatic Web Service Composition for the Life Sciences Using the Biomoby Semantic Web Framework. Journal of Biomedical Informatics, Vol. 41, Issue 5, Semantic Mashup of Biomedical Data, ISSN 1532-0464, 2008, pp. 837–847.

[14] Berners-Lee, T.—Hendler, J.—Lassila, O.: The Semantic Web. Scientific American, Vol. 2001, pp. 34–43.

[15] Bertoli, P.—Kazhamiakin, R.—Paolucci, M.—Pistore, M.—Raik, H.—Wagner, M.: Control Flow Requirements for Automated Service Composition. In Int. Conf. on Web Services 2009, IEEE CS 2009, pp. 17–24.

[16] Blake, M. B.—Tan, W.—Rosenberg, F.: Composition as a Service. IEEE Internet Computing, Vol. 14, 2010, No. 1, pp. 78–82.

[17] Cardoso, J.—Sheth, A. P.: Semantic Web Services, Processes and Applications. Springer 2006.

[18] Dustdar, S.—Papazoglou, M. P.: Services and Service Composition – An Introduction (Services Und Service Komposition – Eine Einführung). IT – Information Technology, Vol. 50, 2008, No. 2, pp. 86–92.

[19] Euzenat, J.—Shvaiko, P.: Ontology Matching. Springer Verlag, Berlin Heidelberg 2007.

[20] Gamha, Y.—Bennacer, N.—Naquet, G. V.—Ayeb, B.—Romdhane, L. B.: 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, IEEE CS 2008, pp. 228–237.

[21] El Haddad, J.—Manouvrier, M.—Rukoz, M.: TQoS: Transactional and QoS-Aware Selection Algorithm for Automatic Web Service Composition. IEEE Transactions on Services Computing 99 (PrePrints), 2010, pp. 73–85.

[22] Hoffmann, J.: A Heuristic for Domain Independent Planning and Its Use in an Enforced Hill-Climbing Algorithm. In ISMIS '00: Proc. of the 12th Int. Symposium on Foundations of Intelligent Systems. Springer-Verlag, London, UK 2000, pp. 216–227.

[23] Huang, Zh.—Jiang, W.—Hu, S.—Liu, Zh.: Effective Pruning Algorithm for QoS-Aware Service Composition. In Int. Conf. on E-Commerce Technology 2009, IEEE CS 2009, pp. 519–522.

[24] Hull, D.—Wolstencroft, K.—Stevens, R.—Goble, C.—Pocock, M.—Li, P.—Oinn, T.: Taverna: A Tool for Building and Running Workflows of Services. Nucleic Acids Research, Vol. 34, Web Server issue, pp. 729–732, 2006.

[25] Jiang, W.—Zhang, C.—Huang, Zh.—Chen, M.—Hu, S.—Liu, Zh.: Qsynth: A Tool for QoS-Aware Automatic Service Composition. In Int. Conf. on Web Services 2010, pp. 42–49. IEEE CS 2010.

[26] Karakoc, E.—Senkul, P.: Composing Semantic Web Services Under Constraints. Expert Syst. Appl., Vol. 36, 2009, No. 8, pp. 11021–11029.

[27] Klusch, M.—Gerber. A.: Evaluation of Service Composition Planning With Owls-Xplan. Web Intelligence and Intelligent Agent Technology – Workshops 2006, pp. 117–120.

[28] Klusch, M.—Gerber, A.—Schmidt. M.: Semantic Web Service Composition Planning with Owls-Xplan. In AAAI Fall Symposium on Semantic Web and Agents, Arlington (VA), USA, AAAI Press 2005, pp. 55–62.

[29] Kona, S.—Bansal, A.—Blake, B.—Bleul, S.—Weise, T.: A Quality of Service-Oriented Web Services Challenge. In Int. Conf. on E-Commerce Technology 2009, IEEE CS 2009, pp. 487–490.

[30] Kona, S.—Bansal, A.—Blake, M. B.—Gupta, G.: Generalized Semantics-Based Service Composition. In ICWS '08: Proc. of the 2008 IEEE Int. Conf. on Web Services. IEEE CS 2008, pp. 219–227.

[31] Kona, S.—Bansal, A.—Gupta, G.: Automatic Composition of Semantic Web Services. In ICWS '07: Proc. of the 2007 IEEE Int. Conf. on Web Services, IEEE CS 2007, pp. 150–158.

[32] Kona, S.—Bansal, A.—Simon, L.—Mallya, A.—Gupta, G.—Hite, T. D.: USDL: A Service-Semantics Description Language for Automatic Service Discovery and Composition. Int. J. Web Service Research, Vol. 6, 2009, No. 1, pp. 20–48.

[33] Lécué, F.—Delteil, A.—Leger, A.: Applying Abduction in Semantic Web Service Composition. In ICWS '07: Proc. of the 2007 IEEE Int. Conf. on Web Services, Los Alamitos, CA (USA), IEEE CS 2007, pp. 94–101.

[34] Lécué, F.—Mehandjiev, N.: Towards Scalability of Quality Driven Semantic Web Service Composition. In Int. Conf. on Web Services 2009, IEEE CS 2009, pp. 469–476.

[35] Lécué, F.—Salibi, S.—Bron, Ph.—Moreau, A.: Semantic and Syntactic Data Flow in Web Service Composition. In ICWS '08: Proceedings of the 2008 IEEE International Conference on Web Services, Washington, DC (USA), IEEE Computer Society.2008, pp. 211–218.

[36] Li, Y.—Huai, J. P.—Sun, H.—Deng, T.—Guo, H.: PASS: An Approach to Personalized Automated Service Composition. IEEE International Conference on Services Computing, Vol. 1, 2008, pp. 283–290.

[37] Lin, M.—Xie, J.—Guo,H.—Wang, H.: Solving QoS-DrivenWeb Service Dynamic Composition As Fuzzy Constraint Satisfaction. In EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, EEE '05, Washington, DC (USA), IEEE Computer Society 2005, pp. 9–14.

[38] Lin, N.—Kuter, U.—Sirin, E.: Web Service Composition With User Preferences. In European Semantic Web Conference 2008, Vol. 5021 of LNCS, Springer, 2008, pp. 629–643.

[39] Martínez Carreras, M.—Gómez Skarmeta, A.: Combining Web 2.0 and Web Services in Collaborative Working Environments. In Computing and Informatics. Vol. 30, 2011, No. 1, pp. 137–164.

[40] Mokhtar, S.—Fournier, D.—Georgantas, N.—Issarny, V.: Context-Aware Service Composition in Pervasive Computing Environments. In: Nicolas Guelfi, Anthony Savidis (Eds.): Rapid Integration of Software Engineering Techniques, Vol.e 3943 of Lecture Notes in Computer Science, Springer Berlin/Heidelberg 2006, pp. 129–144.

[41] Nau, D. S.—Au, T. C.—Ilghami, O.—Kuter, U.—Murdock, W.—Wu, D.—Yaman, F.: Shop2: An HTN Planning System. J. Artif. Intell. Res. (JAIR), Vol. 20, 2003, pp. 379–404.

[42] Kirci Ozorhan, E.—Kuban, E. K.—Cicekli, N. K.: Automated Composition of Web Services With the Abductive Event Calculus. Information Sciences, Vol. 180, 2010, pp. 3589–3613, ISSN 0020-0255.

[43] Papazoglou, M. P.: Web Services and Business Transactions. World Wide Web, Vol. 6, 2003, No. 1, pp. 49–91.

[44] Papazoglou, M. P.—Traverso, P.—Dustdar, S.—Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. Computer, Vol. 40, 2007, pp. 38–45.

[45] Papazoglou, M. P.: Service-Oriented Computing: Concepts, Characteristics and Directions. IEEE Computer Society, Washington, DC (USA) 2003, pp. 3–12.

[46] Peer, J.: Web Service Composition as AI Planning – A Survey. University of St. Gallen 2005.

[47] Rosenberg, F.—Leitner, P.—Michlmayr, A.—Celikovic, P.—Dustdar, S.: Towards Composition as a Service – A Quality of Service Driven Approach. In ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering, IEEE Computer Society, Washington, DC (USA) 2009, pp. 1733–1740.

[48] Rosenberg, F.—Muller, M. B.–Leitner, P.—Michlmayr, A.—Bouguettaya, A.—Dustdar, S.: Metaheuristic Optimization of Large-Scale QoS-Aware Service Compositions. In IEEE International Conference on Services Computing, Los Alamitos, CA (USA), IEEE Computer Society 2010, pp. 97–104.

[49] Shin, D. H.—Lee, K. H.: An Automated Composition of Information Web Services Based on Functional Semantics. In 2007 IEEE Congress on Services, July 2007, pp. 300–307.

[50] Sirin, E.—Hendler, J.—Parsia, B.: Semi-Automatic Composition of Web Services Using Semantic Descriptions. In Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS 2003, 2002.

[51] Sirin, E.—Parsia, B.—Wu, D.—Hendler, J. A.—Nau, D. S.: HTN Planning for Web Service Composition Using Shop2. J. Web Sem., Vol. 1, 2004, No. 4, pp. 377–396.

[52] Studer, R.—Grimm, S.—Abecker, A.: Semantic Web Services: Concepts, Technologies, and Applications. Springer 2007.

[53] Urbieta, A.—Azketa, E.—Gomez, I.—Parra, J.—Arana, N.: Analysis of Effects- and Preconditions-Based Service Representation in Ubiquitous Computing Environments. International Conference on Semantic Computing, Los Alamitos, CA (USA), IEEE Computer Society 2008, pp. 378–385.

[54] Williams, A. B.—Padmanabhan, A.—Blake, M. B.: Experimentation with Local Consensus Ontologies with Implications for Automated Service Composition. IEEE Transactions on Knowledge and Data Engineering, Vol. 17, 2005, pp. 969–981.

[55] Yang, J.—Papazoglou, M. P.: Service Components for Managing the Life-Cycle of Service Compositions. Inf. Syst., Vol. 29, 2004, No. 2, pp. 97–125.

[56] Yoon, P. K.—Hwang, C. L.—Yoon, K.: Multiple Attribute Decision Making: An Introduction (Quantitative Applications in the Social Sciences). Sage Publications Inc., 1995.

[57] Yu, H. Q.—Reiff-Marganiec, S.: A Backwards Composition Context Based Service Selection Approach for Service Composition. In: Int. Conf. on Services Computing 2009, IEEE CS 2009, pp. 419–426.

[58] Zeng, L.—Benatallah, B.—Dumas, M.—Kalagnanam, J.—Sheng, Q. Z.: Quality Driven Web Services Composition. In Int. Conf. on World Wide Web 2003, ACM, 2003, pp. 411–421.

[59] Zeng, L.—Benatallah, B.—Ngu, A. H. H.–Dumas, M.—Kalagnanam, J.—Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Trans. Softw. Eng., Vol. 30, 2004, No. 5, pp. 311–327.

[60] Zhao, H.—Doshi, P.: Towards Automated Restful Web Service Composition. IEEE International Conference on Web Services 2009, pp. 189–196.

Peter Bartalos received his Master degree (with cum laude) in 2007 and his Ph. D. degree in 2011, both from the Slovak University of Technology in Bratislava. Since 2006 he has been working as a researcher at the Institute of Informatics and Software Engineering, Slovak University of Technology in Bratislava. His research covers the Web services area with the focus on dynamic composition of semantic Web services. Since May 2011 he is a postdoctoral research associate at the University of Notre Dame, Indiana, USA. His research at this university is more oriented to Green IT.

Mária Bieliková received her Master degree (with summa cum laude) in 1989 and her Ph. D. degree in 1995, both from the Slovak University of Technology in Bratislava. Since 2005, she has been a Full Professor, presently at the Institute of Informatics and Software Engineering at the Slovak University of Technology. Her research interests are in software web-based information systems, especially personalized context-aware web-based systems including user modelling and social networks.