Model-based development of Web service descriptions enabling a precise matching concept

Jan Hendrik Hausmann, Reiko Heckel, Marc Lohmann
Department of Computer Science
University of Paderborn
Germany
hausmann||reiko||mlohmann@upb.de

ABSTRACT:

Web services are software components that can be discovered and employed at runtime using the Internet. Conflicting requirements towards the nature of these services can be identified. From a business perspective, Web services promise to enable the formation of ad-hoc cooperations on a global scale. From a technical perspective, a high degree of standardization and rigorous specifications are required to enable the automated integration of Web services. A suitable technology for Web services has to mediate these needs for flexibility and stability. To be usable in practice, this technology has to be aligned to standard software engineering practice to allow for a seamless development of Web service enabled components.

In this paper, we introduce a new approach to the description of Web services. It is a visual approach based on the use of software models and graph transformations and allows for the flexible description of innovative services while providing a precise matching concept. A methodology enabling the seamless development of such Web service descriptions in the context of a standard model-based development approach is presented.

KEY WORDS:

Web Services, UML, model-based development, ontology, Design by Contract, matching, graph transformation

INTRODUCTION

The global connection of computers by the Internet has made a huge impact on the life of many people. It is now possible to gather information on almost every conceivable subject by just using the WWW. Additionally, the advances in eCommerce have also made it possible to request services by the same medium, e.g., booking flights, buying books, and making payments. Yet, most of these tasks require (sometimes rather complex) interactions between a computer and a human user. The vision of Web services is to make these (and other kinds of) services available to machines. This means that programs should be able to locate and invoke needed services dynamically at runtime over the Internet. Web services therefore follow the service-oriented architecture (SOA), which defines the roles of provider, requestor, and central discovery services. Providers advertise their offered Web services by publishing descriptions on a discovery service. When clients (requestors) are looking for a specific kind of service, they query this discovery service and receive a list of suitable candidates. After selecting the best of these services, the client is able to use the Web service directly by contacting the provider.

Economically, this scenario integrates B2B and B2C Commerce solutions on a global scale: basic services like authentication, payments, and shipping can be obtained from suitable service providers and be combined into innovative and attractive product offers to customers. One of the central characteristics of electronic commerce is its highly dynamic nature. New companies might spring up overnight (and close down just as fast) and new kinds of services can be created with a very low overhead (as compared to traditional markets). The Web service technology must reflect these dynamics in allowing for the ad-hoc advertisement and location of new providers and new service types. Having a global market comprising continuously evolving offers also requires flexibility in the matching of requests and offers. Flexibility is the ability to match offers to requests in a way that satisfies both parties, but which must not be restricted to a simple identity function (as that would force clients to evolve their requests along with the offers). Providers might describe their services in a specialized way while clients might formulate their requests in a more general fashion to get many competing offers. In any case, the discovery process must be dependable in that it finds all offers that match the request and that all returned results are adequate to the request.

Technologically, the scenario presents many problems typical for distributed component-based computing. The common solution to enable communication between remote components is the definition of standard protocols and vocabularies to which all involved components must adhere. Thus, technologically, it is ideal, if all participants agree on something and keep to that agreement for a long time. This need for stability is in stark contrast with the business requirements.

Another important aspect that needs to be taken into account is the *development* of Web services. On the one hand, it is clear that Web services need to be independent from the implementation technologies used by the requestor and the provider. On the other hand, one should also keep in mind that Web services are not a standalone product but that they require a tight integration with the surrounding software and the technologies used to implement it. In particular, this means that developers (of service requesting as well as service providing software) need to understand about the connections between the Web service parts and the rest of their application. An alignment of Web service technologies to current software development practices is necessary to allow for a seamless and consistent employment of Web service technologies.

Current technologies and research approaches do not address (all of) these criteria (see Section 2 for a discussion). We thus propose an innovative approach to the description of Web services, which is based on models (Section 3). By using graph transformations as the underlying mechanism of our approach, we have a well-known formal basis, which allows for a precise formulation of a matching concept while retaining an intuitive interpretation. In combination with the concept of ontologies, graph transformations provide the required flexibility in describing the semantics of innovative Web services and enable a reliable matching process (as described in Section 4).

The integration with the development of systems on the requestor as well as on the provider side is achieved by employing diagrams of the Unified Modeling Language (UML) (Object Management Group, 2003) as a visualization known to software developers. We furthermore provide a methodology to derive the information required for the semantic description of a Web service that can be applied during a model based development process (Section 5). Using this methodology enables developers to embrace Web service technology while staying in their familiar development framework.

TOWARDS SEMANTIC SERVICE DESCRIPTIONS

The vision of service-oriented architectures is that a program in need of some functionality (which it cannot provide itself) queries a central directory to obtain a list of service descriptions of potential suppliers of this functionality. As an example think of an eLearning system which offers its users the option to order books recommended for a specific course. As bookselling is beyond the scope of the eLearning application, a suitable Web service is to be invoked if users want to order books. An important fact to note at this point is that the client application knows *what kind of service* it needs (i.e., its *semantics*), but not necessarily how the service is actually called by the provider (i.e., its *syntax*).

The notion of service description is central to the whole Web service discovery process. A service description is published by the service provider and forms the base against which the central directory can match requests. Information that is not present in the service description is not available for an automated matching. In this section, we will thus focus on the service description and analyze current standards and proposals.

WEB SERVICE DESCRIPTION STANDARDS

An interface definition is a technical description of the public interfaces of a Web service. It specifies the format of the request and response from the Web service. The Web Service Description Language (WSDL) (Chinnici et al., 2004) proposed by the World Wide Web Consortium (W3C) is an XML (Extensible Markup Language) format for describing interfaces offered by a Web service as a collection of operations. For each operation its input and output parameters are described. The W3C and other related work refers to this kind of service description as the “documentation of the mechanics of the message exchange” (Benatallah et al., 2003; Booth et al., 2004). While these mechanics must be known to enable binding, an automated discovery of services based on WSDL is not possible. WSDL only encodes the syntax of the Web service invocation, it does not yield information on the service’s semantics. Of course, human users might guess which service an operation *orderBook(isbn:String)* provides but such explicit operation names are technically not required.

UDDI (Universal Description, Discovery, and Integration) (UDDI Consortium, 2001), the protocol for publishing service descriptions allows users to annotate their WSDL file with information about the service in the form of explanatory text and keywords. Keywords are one way of supplying semantics but not a reliable one as the results of a query are strongly influenced by guessing the right search terms. The current state of Web service technology is such that a developer solves these semantic problems. When creating a client program, which needs a Web service, the developer queries a UDDI server manually, using either categorization or keyword based search features, and selects a suitable Web service according to the description given in natural language. Information about the Web service and its invocation is coded into the client (or a suitable proxy) and discovery at runtime is (at best) restricted to discovering alternative providers offering the same service.

To overcome the problem of different (synonymous) keywords, the notion of ontology plays an important role. Ontologies “are formal and consensual specifications that provide a shared and common understanding of a domain, an understanding that can be communicated across people and organization systems" (Fensel and Bussler, 2002), i.e., ontologies define a vocabulary that is common to all stakeholders in a domain.

OWL (Web Ontology Language) (Dean et al., 2003) and DAML+OIL (Connolly et al., 2001) are industrial standards to enable the creation of ontologies. OWL-S (Semantic Markup for Web Services) (Martin et al., 2004) is based on OWL and provides an ontology markup language to represent capabilities and properties of Web services. Its goals are to achieve automatic Web service discovery, invocation, composition, and execution monitoring. OWL-S allows describing a Web service based upon three basic types of information: provider information (simple contact information), functional descriptions, and further information for specifying the characteristics of a service (mainly by references to existing taxonomies). The most interesting part is the functional description. It describes the input, output parameters, and additionally pre-conditions (called external conditions that must be true for the execution of a service) and effects (called side effects of the execution of a Web service). The pre-conditions and effects are logic formulas to restrict the input and output parameters. That means, the functional descriptions are on the same level as a WSDL document of a service, it does not yield information on the service's semantics.

PROPOSALS

The problem of describing Web services in a way to enable their truly dynamic discovery at runtime has been the focus of a number of scientific publications. In the approaches of (McIlraith et al., 2001) or (Dogac, Cingil et al. 2002) the semantic of a Web service is described by referencing a term of the ontology, which describes the functionality of a service. The approach of (McIlraith et al., 2001) is based on DAML (DARPA Agent Markup Language), while in (Dogac, Cingil et al. 2002) DAML-S (DAML Services - the predecessor of OWL-S) is exploited and integrated with UDDI registries. Both approaches allow for the definition of special kinds of services in an ontology. The terms in the ontologies in (McIlraith et al., 2001) are very specific, up to containing special product information (e.g., there is a term *buyLufthansaTicket* in an ontology relating to air travel business). While the paper shows how this detailed information may be employed to select suitable Web services according to complex strategies, specifying these detailed semantics in an ontology contradicts the dynamic nature of the eBusiness. Every time a new brand is introduced or an old one disappears (e.g., by renaming) the ontology has to be changed accordingly. The approach of (Dogac, Cingil et al. 2002) introduces abstract service types (e.g., *CarRentalService)* which can be referenced by concrete services. With both approaches the introduction of new business ideas (e.g., a bidding option for cheap tickets or rental contracts) requires an extension of the ontology to reference this new idea with a proper term. Since changing an ontology is a process of (international) agreement, it may take a lot of time. Additionally, applications using the ontology have to be adapted for using the changed ontology. Because of this agreement and adaptation process, ontologies often contain only very common terms to ensure a broad applicability. In (Dumas et al., 2001) e.g., classifications based on type of service (tangible/intangible) or industry branches are proposed. This categorization is clearly not detailed enough to actually distinguish specific services.

Trying to describe a service with a single term also ignores the operational nature of Web services. When executing a Web service one expects certain changes, i.e., the real world is altered in some significant way (e.g., an order is created, a payment made or an appointment is fixed). Web service descriptions should reflect this functional nature by providing a semantic description in the form of pre- and post-conditions (a style of description also known from contract-based programming (Meyer, 1997)). This kind of semantic description can be found in (Paolucci et al., 2002; Sivashanmugam et al., 2003). The pre- and post-conditions are once again expressed in terms of specialized ontologies. While (Paolucci et al., 2002) shows the matching only for single input and output concepts, (Sivashanmugam et al., 2003) combines a number of predefined terms to express the pre- and post-conditions (e.g., *CardCharged-TicketBooked-ReadyforPickup*). By using this style of description, it is possible to distinguish between rather similar services (e.g., booking a ticket, which is sent to the customer vs. booking a ticket, which has to be picked up) without coining special phrases for each in the ontology.

While this combination of terms points in the right direction, we believe that this concept should be taken one step further to building structures over the terms of an ontology. The advantage of this approach is that the ontology itself can contain very common and basic terms of the business (which are rather stable), and services can combine these terms in an individual way to express their specialized semantics. Thus, stability in the ontology and flexibility in describing innovative services is achieved.

MODELING THE SEMANTICS OF WEB SERVICES WITH ONTOLOGIES AND GRAPH TRANSFORMATION RULES

The previous section outlined our basic idea how to mediate the need for flexibility and stability in Web service descriptions. However, many technologies can be employed to actually realize this idea. In this section we will argue that a main criterion in choosing the formalism used to express the semantics of Web services is the understandability and usablity of the formalism for the intended users of the notation (i.e., Software Engineers). From this point of view, we chose the combination of UML and graph transformations to express our ideas.

THE CASE FOR MODELS

Web services are not a stand-alone technology. They rather provide interfaces to either publish or request certain functionalities of a complex software system. The use of Web services is not a primary goal in this system's development (unless its goal is technology demonstration) nor are Web services the main or only implementation technology used. Software engineers are rather faced with the task of integrating the Web service technology with the rest of a system, which is constructed independently of the Web service part.

For a provider this means to decide which of the operations in the application is suitable to be published as a Web service interface. Then one has to translate this interface into the descriptions of WSDL. If the service has a complex structure (i.e., it possibly consists of a sequence of operation calls) it is necessary to encode this in a BPEL (Business Process Execution Language) specification (Andrews et al., 2003). A layer (middleware) that enables the translation of SOAP based communications into method invocations has to be provided and a UDDI server must be used to publish the descriptions. If ontologies are to be used, some dialect of the DAML or OWL languages needs to be employed, too. Not only is this a vast body of different languages and technologies that is being used, but these languages often encode similar information in different ways. Without suitable support for planning, implementing, testing and documenting this part of the system, it is highly doubtful whether the Web service part of a system will be reliable and consistent with the rest of the application (and almost more important: whether it will stay that way during the next changes to the system). From the requestor's view, additional problems arise from the need for an automated comparison of offers and an automated integration of an offered service (i.e., constructing necessary input data and interpreting results).

The usage of models is a significant step toward the solution of these problems. Models provide an abstraction from the detailed problems of the implementation technology and allow the developer to focus on the more abstract tasks. In particular, the diagrams of the industry standard UML (Unified Modeling Language) have become very successful and are now an established part of a system's development. With the advent of the Model Driven Architecture (OMG (Object Management Group), 2001) models become even more crucial to software development, as implementation artifacts can automatically be generated from the models, thus saving time and guaranteeing consistency. Extending the modeling support to the development of Web services would have several advantages:

- Models are independent of the target language(s). It is thus possible to derive the different specifications consistently from one source. For our scenario, this means that the same models, which generate the code for the application (e.g., Java code), could generate the required WSDL documents including semantic descriptions. Consistency between the implementation and the Web service interface would thus be guaranteed.

- Models are visual. Structures (like ontologies) can be understood much more intuitively if they are presented in a visual manner (as compared to pages of XML code). Software Engineers have long since recognized this feature of visual languages and make heavily use of it. In the field of Web services, this idea resulted in UML class diagram representations, which are equivalent to DAML+OIL or OWL specifications (Baclawski et al., 2001).
  It is also possible to present mappings between different representations of the same information visually (Hausmann and Kent, 2003).

- Models are becoming more important and a number of tools for using models in the development of software are available. Basing the integration of Web services on models allows reusing existing information and tools, thus integrating the development of Web services in the development of the surrounding software.

Due to these reasons, we chose a technique that is highly compatible with the modeling of UML.

COMBINING ONTOLOGIES AND GRAPH TRANSFORMATION RULES

As indicated in Sect. 2.2, the main idea of our description of Web services' semantics is to form expressions over the terms provided by an ontology. Different technologies to achieve this goal come to mind (e.g., logical formulae, as used by RDF - Resource Description Framework). While a formal precision is necessary to guarantee a reliable matching of offers and requests, we need to use a formalism, which is in line with the visual representations of ontologies. Graph transformations are such a formalism. They combine formal rigor with a visual appeal and - especially in combination with the notations of the UML - have found a large number of practical applications.

We therefore propose a technique, which is based on ontologies but uses the mechanism of graph transformation rules to allow for the flexible formulation of new services. Before we introduce the technical details let us look at an example. Figure 1 provides a small sample ontology for bookselling. It provides the basic terminology and is depicted as a UML class diagram. A generalization relationship indicates that *CreditCard* and *BankAccount* are specializations of the basic concept *Payment*.

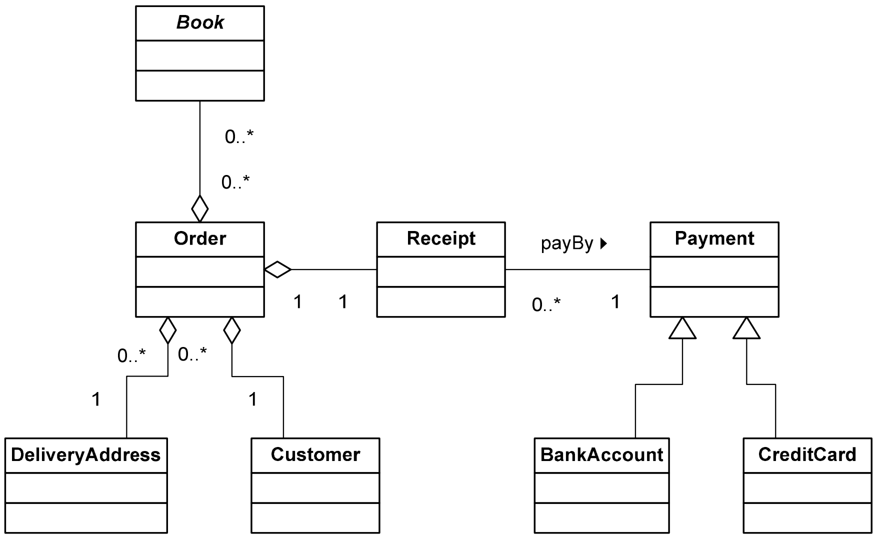

Figure 1. Ontology for bookselling

If a vendor now wants to express that his service is able to handle orders for books which are payable by credit card, he can formulate the rule in Figure 2. The rule expresses that the Web service needs data on a book to order, the personal data of the buyer and his credit card and will effect a new order for the specified book and a receipt (which is paid by credit card) to be created for this customer. Note that the notion of buying a book is not directly encoded in the ontology, but rather stems from the combination of terms chosen in the service's description.

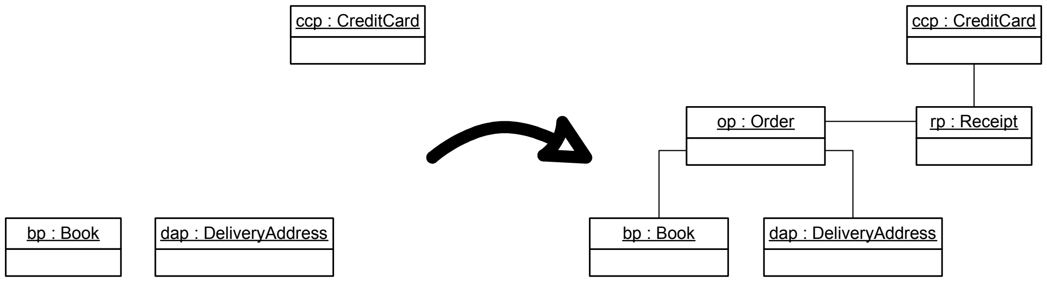

Figure 2. Provider Rule

Using this mechanism, it becomes easy to describe all kinds of different services in this area. One might e.g., formulate descriptions for services, which add items to existing orders or delete items or which offer books without any payment information. As real ontologies are vastly bigger than the small example presented here, many services can be described by constructing this kind of description.

Formally, a graph transformation rule consists of two graphs (left- and right-hand side), which are in our case visualized by using UML object diagrams. Each of the graphs is typed over the ontology. Refer to (Rozenberg, 1997) for the technical details. The basic intuition is that every object, which is only present in the right hand side of the rule, is newly created and every object, which is present only in the left hand side of the rule, is being deleted. Objects which are present on both sides are unaffected by the rule. If only one object of a type exists, it can remain anonymous, if a distinction between different objects of one type is necessary, then they must carry object identifier, separated from their type by a colon. If an even closer resemblance to standard UML concepts is called for, it is also possible to encode graph transformations in UML Collaboration Diagrams (Heckel and Sauer, April 2001).

Graph transformation rules can serve as both: a description of an offered service and as the formulation of a request. From a providers point of view the left hand side of the rule specifies the pre-condition of his service, i.e., the situation that must be present or the information that must be available for the service to perform its task. The right hand side of the rule depicts the post-condition, i.e., it characterizes the situation after the successful execution of the Web service. From a requestors point of view the left hand side of the rule represents the information he is willing to provide to the service and the right hand side of the rule represents the situation he wants to achieve by using the service. Such a rule from a requestor's point of view can be found in Figure 3. This rule expresses that the client is able to provide information on a book, a credit card, and his delivery address and is looking for a provider, which is able to construct an order for him based on these information. This means he is looking for a bookseller and intuitively the provider rule from Figure 2 should be a suitable candidate for this request, because in this rule an order is created. For more examples of this matching concept see (Hausmann et al., 2003). The next section provides in-depth information on the formalization of this "intuitive" matching concept.

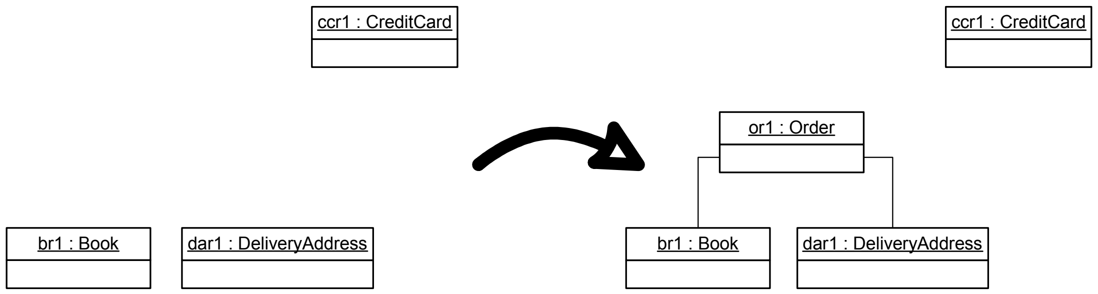

Figure 3. Requestor Rule 1

MATCHING SERVICE DESCRIPTIONS AND REQUIREMENTS

As with any software, the motivation for implementing a Web service is to satisfy some demand, formally expressed in a requirements specification. What is more specific (albeit not new if we consider component-based systems) is that, often, the desired functionality is not entirely implemented by the service alone, but in part provided by other services, not necessarily known at design time. When a service *R* (the requestor) finds and binds dynamically to a service *P* (the provider), *R* has to be sure that *P* provides its functionality in a way that is consistent with the assumptions about the required services made in *R*'s implementation. These assumptions are expressed in the requestor rule *r*. The discovery service compares *r* to the available provider rules *p*, which represents descriptions of actually implemented services. The desired result is the set of those provider descriptions, whose services fulfill the requirements expressed in *r*. In this section, we give a formal explanation of what it means that rule *p* fulfils the requirements of rule *r*, or briefly, that *p* matches *r*.

The assumptions made in *R* about *P* are captured in the requestor rule $r{:}L_r \rightarrow R_r$. Its left-hand side L_r declares the objects and links whose existence *R* guarantees when the service is invoked. For example, in the requestor rule of Figure 3 these are objects *ccr1:CreditCard*, *br1:Book*, *dar1:DeliveryAddress*. Note that object identifier in rules act like logic variable names: if consistently substituted on the left- and right-hand side of a rule, the meaning does not change. The effects that *R* hopes to achieve by means of the invocation of a service are deduced by comparing left- and right-hand side. Objects and links in $L_r \backslash R_r$ are meant to be deleted, objects and links in $R_r \backslash L_r$ are meant to be newly created, while objects and links in the intersection $L_r \cap R_r$ are meant to be preserved, when they are present. In our example rule of Figure 3, no deletion is specified, so the objects on the left are also part of the intersection, but an object *or1:Order* with two links should be created. The provider rule $p{:}L_p \rightarrow R_p$ has a slightly different interpretation. It requires the existence of all objects and links in L_p as a pre-condition and guarantees as effect the creation of $R_p \backslash L_p$, the deletion of $L_p \backslash R_p$, and the preservation of $L_p \cap R_p$. That means for the provider rule of Figure 2, objects *ccp:CreditCard*, *bp: Book*, *dap:DeliveryAddress* are required, it is guaranteed that these are preserved and that, moreover, objects *op:Order*, *rp:Receipt* and four links are created.

Matching provider and requestor now means to compare the guarantees given by either side to the assumptions of the other: the requestor's pre-condition must entail the provider's pre-condition and the provider's effects must entail the requestor's effects. Considering our sample rules, this is easily seen to be the case. Formally, it results in the following definition:

A provider rule $p{:}L_p \rightarrow R_p$ matches a requestor rule $r{:}L_r \rightarrow R_r$ if there exists an arbitrary structure compatible partial one-to-one correspondence $\boldsymbol{h} \subseteq L_p \cup R_p \times L_r \cup R_r$ with the following characteristics:

1. $\forall x \in L_p\ \exists y \in L_r : x\ \boldsymbol{h}\ y$, i.e., everything in L_p must have a correspondence in L_r;

2. $\forall y \in L_r \backslash R_r\ \exists x \in L_p \backslash R_p : x\ \boldsymbol{h}\ y$, i.e., everything in L_r meant to be deleted by r, must correspond via $\boldsymbol{h}$ to an item that is indeed deleted by p;

3. $\forall y \in L_r \cap R_r\ \exists x \in L_p \cup R_p : x\ \boldsymbol{h}\ y \Rightarrow x \in L_p \cap R_p$, i.e., everything in L_p corresponding via $\boldsymbol{h}$ to an item meant to be preserved by r, must indeed be preserved by p;

4. $\forall y \in R_r \backslash L_r\ \exists x \in R_p \backslash L_p : x\ \boldsymbol{h}\ y$, i.e., everything in R_r meant to be created by r, must correspond via $\boldsymbol{h}$ to an item that is indeed created by p;

The structure compatible partial one-to-one correspondence $\boldsymbol{h}$ is a binary renaming relation over the sets $L_p \cup R_p$ and $L_r \cup R_r$. We write $x\ \boldsymbol{h}\ y$ if $x \in L_p \cup R_p$ is h-related to $y \in L_r \cup R_r$. Structure compatibility means that related objects and links must be of compatible type, and related links must have related source and target objects. Partiality means that not all objects and links of the sets $L_p \cup R_p$ and $L_r \cup R_r$ have to occur in the relation, that means $\boldsymbol{h}$ is neither total nor surjective. By one-to-one we mean that every object or link can occur at most once in the relation, i.e., $\boldsymbol{h}$ is functional. Renaming is necessary because, as in our example, items occurring in provider and requestor rules may have different object identifier that are, however, not of interest as long as the structure is the same.

Let us apply this definition to the rule in Figure 2 for p and Figure 3 for r. In order to satisfy 1, the correspondence has to include the pairs $\{(ccp, ccr1), (bp, br1), (dap, dar1)\} \subseteq \boldsymbol{h}_1$, whose corresponding elements are of the same type. Item 2 requires an inclusion of the deleted elements of r into those of p, which is trivial because these sets are empty for both rules. For the same reason, intersections $L_r \cap R_r = L_r$ and $L_p \cap R_p = L_p$ are, in fact, isomorphic via $\boldsymbol{h}_1$. Finally, 4 requires that all elements created in r (i.e., the object *or1:Order* with its two (anonymous) links) are in correspondence with elements created in p. Thus, the relation $\boldsymbol{h}_1$ must be extended to include the pair (*op*, *or1*) as well as the links from *or1* and their corresponding ones from *op*.

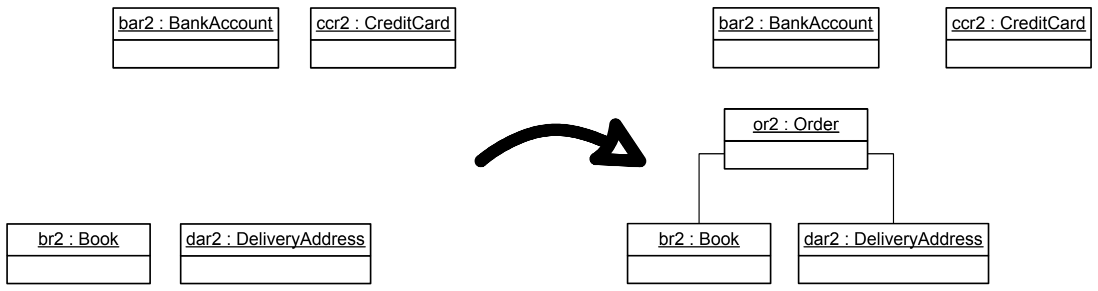

Figure 4. Requestor Rule 2

The situation is a little more subtle if we consider the requestor rule in Figure 4. The correspondence $\boldsymbol{h}_2$ = *{(ccp, ccr2), (bp, br2), (dap, dar2)}* satisfies requirement 1. Notice that the new object *btr2:BankAccount* in L_r is not part of the correspondence, because this data is not assumed by the provider. However, it is included in the requestor rule to make it as flexible as possible, thus returning more matches. The sets of deleted and created elements are similar to the first rule, but as the elements required to be preserved by the requestor, we have *btr2*, *ccr2*, *br2*, *dar2* in the intersection $L_r \cap R_r$. The set of elements of p corresponding to these via $\boldsymbol{h}_2$ is *ccp*, *bp*, *dap*, and these are indeed preserved, as required by item 3 above. This explains the difference with items 2 and 4, where elements of the respective kind in *r* must be in correspondence with elements of the same kind in *p*, while in 3 elements that are in correspondence must be of the same kind. Thus, objects that are in $L_r \cap R_r$ (required to be preserved by the requestor) are optional in the sense that they need not be present in the provider, but if they are, then they must not be deleted.

A prototypical implementation of our approach is available (see also (Hausmann et al., 2003; Hausmann et al., 2004)). We use DAML+OIL as semantic Web language for representing ontologies and graph transformation rules. An implementation of our matching approach uses the RDQL (A Query Language for RDF) (Seaborne, 2004) implementation of the semantic Web tool Jena from HP. RDQL is a query language for specifying graph patterns that are matched against a graph to yield a set of matches, which allows computing subgraph relations. As visual editor for graph transformation rules we use the tool AGG (The Attributed Graph Grammar System) (Taentzer, 2004).

EMBEDDING IN DEVELOPMENT PROCESS

In the last sections, we have focused on the descriptions of Web services' semantics and the matching of offers and requests based on these descriptions. While our choice of formalisms already took the creation perspective for these descriptions into account (by aligning the descriptions to current Software Engineering notations) we will focus upon this creation in this section. The whole matching technology (in fact the whole Web service technology) becomes meaningless, if they are not aligned to, or, better yet, integrated into the creation of the actual application software. Without such an integration Web service interfaces will soon suffer from inconsistencies with the system they are supposed to represent.

In this section, we describe a methodology that enables a developer to systematically create semantic service descriptions and request in the context of a standard model-based development approach.

For the description of this methodology, we have to shift the focus from the models describing the semantics of a Web service description or request to the models describing the implementation of a service provider or requester. The whole scenario (as illustrated by Figure 5) now contains three different vocabularies. On the one hand, a common ontology for bookselling allows for a relation of the semantic Web service descriptions and requests (e.g., both service description and requester have to use the identical term *Customer*) according to our approach for the specification of Web service semantics as described in Section 3 and 4. On the other hand, the implementation of both service requester and provider are oriented along their domains and technical requirements. They need not to have anything in common with the ontology. For example, an eLearning application (requester in Figure 5) will probably call its users *Learner* and only when they order a book for a course, they play the role of *Customer* as coined by the bookselling ontology.

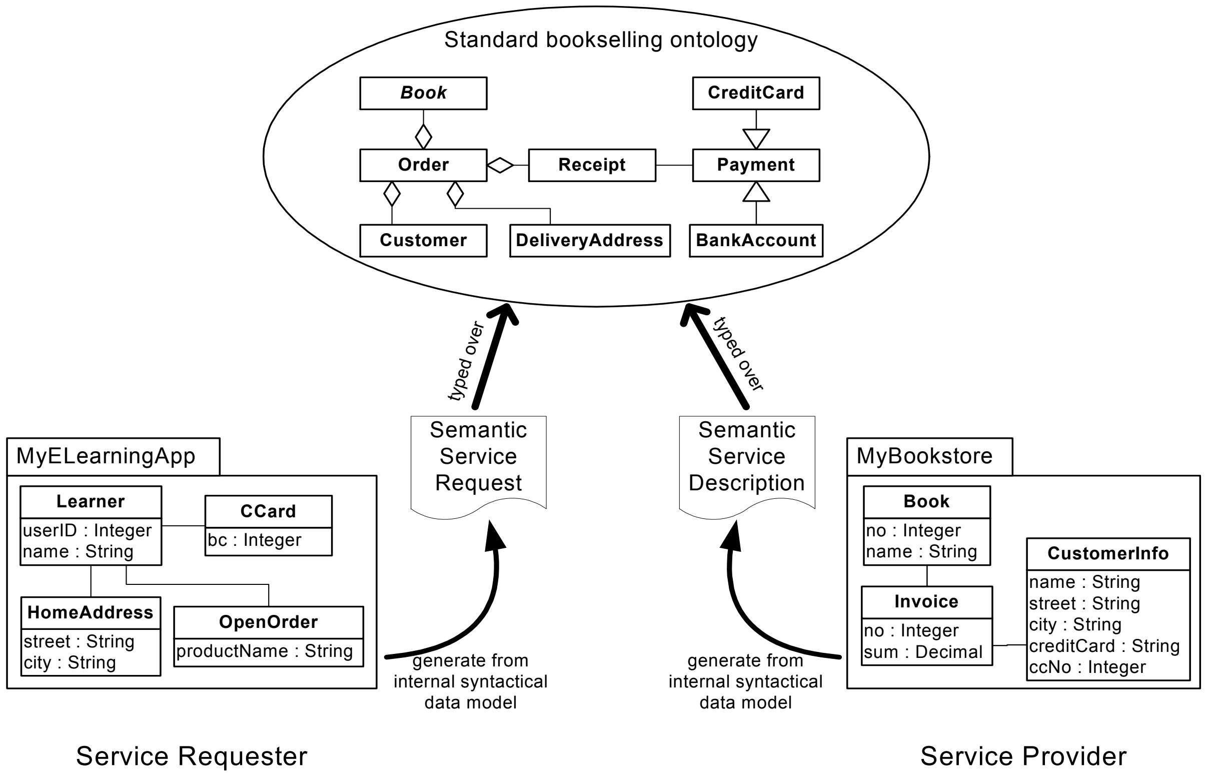

Figure 5. Scenario for Web service interoperation highlighting the different vocabularies involved

A Software Engineer either implementing a potential client or a potential provider is now faced with two tasks: First, he has to recognize the relations between the terms of the internal vocabulary of his application and the terms in the ontology. Second, he has to apply this general understanding on the mapping of terms to the actual application at hand to create the necessary semantic descriptions or requests according to the ontology. While we cannot fully automate the first step (since it once more relies on the human interpretation and understanding of terms and their implications), we can offer to support the second step by code-generation mechanisms. Unless this second step is automated, each evolution of the underlying application will potentially invalidate the published Web service interface, thus making the Web service unreliable.

The assumptions underlying our methodology are presented in Section 5.1. In Section 5.2 we describe how to provide support for the definition of the relation between an internal vocabulary and an ontology. Finally, Section 5.3 shows the gain in terms of automation from the formulation of these relations. Figure 6 gives an overview of this process. To avoid redundant formulations, we concentrate on the part of the service provider only.

ASSUMPTIONS

If a Web service is being developed according to standard model-based development processes (Jacobson et al., 1999), class diagrams are used to describe the static structure of its implementation. Such a class diagram can be regarded as an internal vocabulary of the service provider. Classes are defined in terms of their name, attributes, and methods. Associations, aggregation, and inheritance may define the static relationships.

Further, we assume that the design by contract (Meyer, 1997) principle is used extensively during the development. Design by contract is well known from component-based development approaches to allow for a reliable development. A core concept of design by contract is the formulation of pre- and post-conditions for each operation. While these conditions are usually expressed in logic formulae, one can also use graph transformation rules to visualize them. The result of this visualization is a notation very much like the graph transformation rules employed in Section 3 and 4 for the semantic Web service descriptions. They are however typed over the implementation class diagram of the component and not over any ontology. We assume that at least the methods of the classes implementing the interface of a Web service are detailed by such graph transformation rules.

RELATING INTERNAL AND STANDARDIZED VOCABULARIES

If a developer wants to create a semantic description of a Web service, he has to relate his internal vocabulary to a standardized vocabulary (an ontology). This relation cannot be established automatically because it involves human knowledge on the terms and the concepts they represent. However, we can offer automated support in easing the task (assuming that we have already determined the scope of our Web service interface): While the full internal vocabulary can be rather vast, only a small part is truly significant for the Web Service interface. We call this part the *interface information model.*

The interface information model is a projection from the internal vocabulary, which contains all information types (classes) that are essential to specify the behavior of the Web service. Included are all classes of the interface operations' signatures and additionally all classes that are part of the transformation rules describing the behavior of the interface operations. This construction ensures that the rules describing our interface's operations are typed over the interface information model. Classes that never appear in the interface information model can safely be considered as being purely internal and these need not have any correspondence to terms defined in a common ontology.

Based upon this interface information model, a developer can now proceed to map the relevant terms of his internal vocabulary to the terms of a standardized ontology. Usually, the interface information model will be more technical oriented while the ontology will be a more business-oriented model.

Note that this mapping cannot be restricted to 1:1 class mappings. In practice, it will rather entail splitting classes (i.e., one concept from the internal vocabulary is mapped to several distinct concepts in the ontology), merging classes (several distinct concepts in the internal vocabulary have to be combined into one concept of the ontology) and even the mapping of attributes to classes. For this purpose, we can use an extension of the UML for expressing mappings between these two class diagrams, which is inspired by mathematical relations (Hausmann and Kent, 2003). Using this model-based approach for the definition of the mappings has the advantage that the mapping information can be stored in a uniform way together with the models of the component implementation and the Web service interface. Furthermore, we can still adhere to the requirement of keeping the abstract specifications visual where possible.

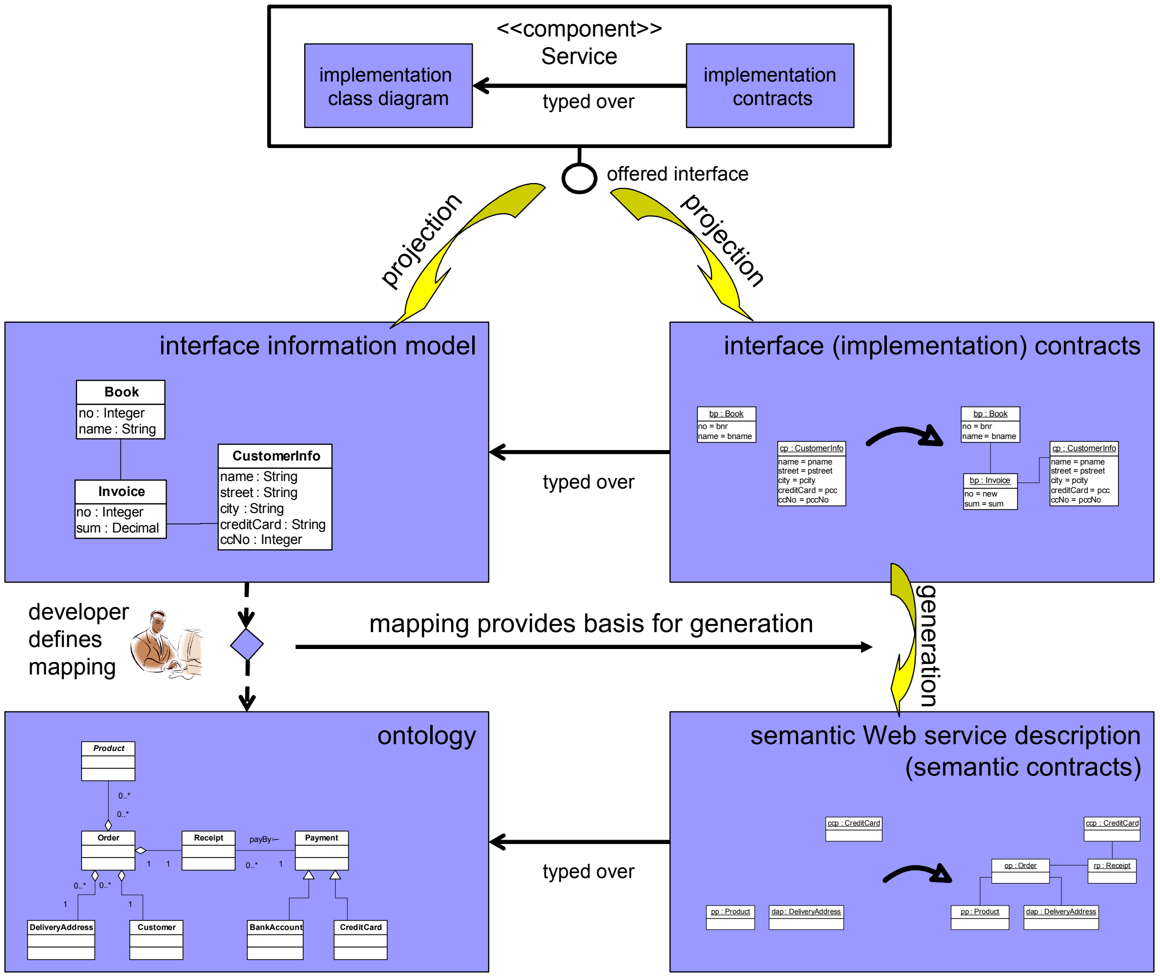

Figure 6. Creating semantic service description from object oriented models

In the example presented in Figure 5 and Figure 6 a developer has to map the internal concept *CustomerInfo* to the concepts of *Customer* and *DeliveryAddress* as defined by the ontology. The details of the mapping (not shown here) will reveal how the different attributes of *CustomerInfo* contribute toward the concepts of the ontology.

CREATING SEMANTIC CONTRACTS FROM IMPLEMENTATION LEVEL CONTRACTS

This generation is the main step in our methodology. If we know which static concepts match, we can use schema evolution techniques (as developed for object-oriented databases (Banerjee et al., 1987; Kolmschlag and Engels, 1998)) to transform the implementation level design by contract rules for the operations into semantic descriptions (contracts that are typed over the ontology) for the offered Web services. Within the boundaries of well-defined invariants, schema evolutions allow to transform the instances of one class diagram (in our case the implementation level model) into instances of another class diagram (in our case the ontology). To overcome the boundaries currently posed by the (sometimes rather restrictive) invariants, further research is being conducted. As an interim solution, we have to rely on additional user input to decide some not automatically decidable properties in the generation process.

The benefit of this generation approach is that every time the implementation is changed in a way as to affect the contracts specified for operations that are part of a Web service interface, a re-generation can be triggered and the semantic Web service descriptions instantly reflect this change. For the developer the burden of specifying the mapping between the internal structure and the ontology is reduced to a minimum as this has to be provided only once and can be incrementally adapted in case of changes. Even in complex situations where an operation is published in different domains (i.e., has service descriptions according to different ontologies), this approach allows for permanently consistent specifications.

SUMMARY AND FUTURE WORK

In this paper, we have provided a high-level approach to the specification of Web service semantics, which combines formal rigor with an intuitive visual representation. To allow for a flexible, yet reliable matching, we use ontologies to formulate a common set of concepts and express services by combing these concepts by the mechanism of graph transformation rules. One characteristic of our matching approach is that only effects, which are observable in terms of structural changes (i.e., deleted or created objects or links), can be expressed. While an extension towards attributed graph transformations (Ehrig et al., 2004) or the inclusion of logical formulae would add expressiveness to the notation, they would also hinder the efficient matching of request and offers (up to making the matching impossible in the completely unrestricted case). The limited expressiveness is also not a problem as we consider that the registry matching is not the last step in the service discovery process; it only yields *potential* service providers. Additional steps to select the most relevant service among these results have to be taken by the client. Since a rather small number of candidates have to be checked in this selection process, finer grained techniques can be used for this subsequent matching.

While this paper has provided the basic concepts and formalization of our approach towards automated service discovery, further work includes improvements to the usability of the notation and the extension towards further steps in the Web service discovery process. Concerning the usability, we would like to integrate advanced concepts from graph transformation rules like alternative elements, optional elements, set-valued elements, and negative application conditions. These features either extend the expressiveness or reduce the amount of rules to be created and transmitted for a service. All of these features have an impact on the matching process however and need to be carefully implemented to allow for an efficient treatment during the matching process. Based upon the concepts given in this paper, one can also start reasoning about further parts of the Web service integration, like choosing from a set of potential services, combining different services, and binding a concrete service automatically. Thus, the technology presented here does not form the answer to all open questions concerning Web services but provides a precise answer to one central problem and enables a more concrete discussion on the remaining issues.

REFERENCES

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., & Weerawarana, S. (2003). Business Process Execution Language for Web Services version 1.1.

Baclawski, K., Kokar, M.K., Kogut, P.A., Hart, L., Smith, J., Holmes III, W.S., Letkowski, J., & Aronson, M.L. (2001). Extending UML to Support Ontology Engineering for the Semantic Web. In Gogolla, M., & Kobryn, C. (Ed.), UML 2001 - The Unified Modeling Language - Modeling Languages, Concepts, and Tools - 4th International Conference. Springer, Lecture Notes in Computer Science, Vol. 2185, 342-360.

Banerjee, J., Kim, W., Kim, H.-J., & Korth, H.F. (1987). Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In Dayal, U. (Ed.), Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data. ACM Press, 311-322.

Benatallah, B., Hacid, M.-S., Rey, C., & Toumani, F. (2003). Semantic Reasoning for Web Services Discovery. In WWW 2003 Workshop on E-Services and the Semantic Web (ESSW' 03).

Booth, D., Haas, H., McCabe, F., Newcomer, E., Michael, C., Ferris, C., & Orchard D. (2004). Web Services Architecture - W3C Working Group Note 11 February 2004.

Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S.A-, Narayanan, S., Paolucci, M., Parsia, B., Payne, T.R., Sirin, E., Srinivasan, N., & Sycara, K. (2004). OWL-S: Semantic Markup for Web Services - W3C Member Submission 22 November 2004.

Chinnici, R., Gudgin, M., Moreau, J.-J., Schlimmer, J., & Weerawarana, S. (2004). Web Services Description Language (WSDL) Version 2.0 part 1: Core language. W3C Working Draft.

Connolly, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., & Stein, L.A. (2001). DAML+OIL (March 2001) Reference Description - W3C Note.

Dean, M., Schreiber, G., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., & Stein, L.A. (2003). OWL Web Ontology Language Reference - W3C Working Draft.

Dogac, A., Cingil, I., Laleci, G., & Kabak, Y. (2002). Improving the Functionality of UDDI Registries through Web Service Semantics. In Buchmann, A.P., Casati, F., Fiege, L., Hsu, M.-C., & Shan, M.-C. (Ed.), TES '02: Proceedings of the Third International Workshop on Technologies for E-Services. Springer, Lecture Notes in Computer Science, Vol. 2444, 9-18.

Dumas, M., O'Sullivan, J., Heravizadeh, M., Edmond, D., and ter Hofstede, A. (2001). Towards a Semantic Framework for Service Sescription. In Meersman, R., Aberer, K., & Dillon, T.S. (Ed.), Semantic Issues in E-Commerce Systems, IFIP TC2/WG2.6 9th Working Conference on Database Semantics. Kluwer, 277-291.

Ehrig, H., Prange, U., & Taentzer, G. (2004). Fundamental Theory for Typed Attributed Graph Transformation. In Ehrig, H., Engels, G., Parisi-Presicce, F., & Rozenberg, G (Ed.)., Proceedings of 2nd International Conference on Graph Transformation (ICGT'04). Springer, Lecture Notes in Computer Science, Vol. 3256, 277-291.

Fensel D., & Bussler C. (2002). The Web Service Modeling Framework WSMF. Technical Report. Vrije Universiteit Amsterdam.

Hausmann, J.H., Heckel, R., & Lohmann, M. (2003). Towards Automatic Selection of Web Services using Graph Transformation Rules. In Tolksdorf R., & Eckstein R. (Ed.), Berliner XML Tage. XML-Clearinghouse, 277-291.

Hausmann, J.H., Heckel, R., & Lohmann, M. (2004). Model-based Discovery of Web Services. In Proceedings of the IEEE International Conference on Web Services (ICWS'04). IEEE Computer Society, 324-331.

Hausmann, J.H., & Kent S. (2003). Visualizing Model Mappings in UML. In Proceedings of the 2003 ACM symposium on Software Visualization. ACM Press, 169-178.

Heckel, R., & Sauer, S. (2001). Strengthening UML Collaboration Diagrams by State Transformations. In Hussmann, H. (Ed.), Proceedings of the 4th International Conference on the Fundamental Approaches to Software Engineering (FASE 2001). Springer, Lecture Notes in Computer Science, Vol. 2029, 109–123.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). The Unified Software Development Process. Addison-Wesley Professional.

Kolmschlag, S., & Engels, G. (1998). Unterstützung der Flexibilität eines Electronic Commerce Systems durch Evolutionstechniken. In Conrad, S., & Hasselbring, W. (Ed.), Workshop "Integration heterogener Softwaresysteme (IHS'98)" im Rahmen der GI-Jahrestagung Informatik '98. Magdeburg, 13-24.

McIlraith, S.A., Son, T.C., & Zeng, H. (2001). Semantic Web Services. IEEE Intelligent Systems (Special Issue on the Semantic Web), 16(2), 46 – 53.

Meyer, B. (1997). Object-Oriented Software Construction (2nd ed.). Sams

OMG (Object Management Group) (2001). Model Driven Architecture (MDA).OMG Web Site: http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

OMG (Object Management Group) (2003). UML 2.0 Superstructure Specification.

Paolucci, M., Kawmura, T., Payne, T.R., & Sycara, K. (2002). Semantic Matching of Web Services Capabilities. In Horrocks, I., & Hendler, J.A. (Ed.), Proceedings of the First International Semantic Web Conference on the Semantic Web. Springer, Lecture Notes in Computer Science, Vol. 2342, 333-347.

Rozenberg, G. (1997). Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific Publishing.

Seaborne, A. (2004). RDQL - A Query Language for RDF - W3C Member Submission 9 January 2004.

Sivashanmugam, K., Verma, K., Sheth, A., & Miller, J. (2003). Adding Semantics to Web Services Standards. In Zhang, L.-J. (Ed.), Proceedings of the International Conference on Web Services (ICWS '03). CSREA Press, 395-401.

Taentzer, G. (2004). AGG: A Graph Transformation Environment for Modeling and Validation of Software. In Pfaltz, J.L., Nagl, M., & Böhlen, B. (Ed.), Applications of Graph Transformations with Industrial Relevance: Second International Workshop. Springer, Lecture Notes in Computer Science, Vol. 3062, 446-453.

UDDI Consortium (2001) UDDI Technical White Paper. UDDI Consortium Web Site: http://www.uddi.org/whitepapers.html.

ABOUT THE AUTHORS

Jan Hendrik Hausmann graduated in Computer Science in 2001 at the University of Paderborn. He holds a position as research and teaching associate at the University of Paderborn in the group of Prof. Dr. Gregor Engels. His research focus lies on Visual Modeling Languages (UML in particular), their definition, and their application. On this topic he has published multiple works on international conferences and journals. He is the principal author of the Dynamic Meta Modeling approach to the definition if UML's dynamic semantics and a fellow of the Segravis RTN.

Dr. Reiko Heckel graduated in Computer Science at the Technical Universities of Dresden and Berlin. After 3 years as a research assistant in Berlin and a scholarship of seven months in the group of Ugo Montanari at the University of Pisa, in 1998 he received his doctoral degree from the TU Berlin. Holding a position as assistant professor in the group of Prof. Dr. Gregor Engels at the University of Paderborn from 1999 to 2004, he is now Reader at the University of Leicester.

Marc Lohmann graduated in Computer Science in 2001 at the University of Paderborn. He holds a position as research and teaching associate at the University of Paderborn in the work group database and information systems of Prof. Dr. Gregor Engels. His research focuses are processes and methods for the development of interactive Web applications and Web services with visual modeling languages (UML in particular). On this topic he has published multiple works on international conferences and he arranged different industrial seminars about Web based standards.