Towards A Flexible Mediation Framework for Dynamic Service Invocations

Philipp Leitner, Florian Rosenberg, Anton Michlmayr, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8/184-1
A-1040, Vienna, Austria
lastname@infosys.tuwien.ac.at

Abstract

One of the main benefits of service-based systems is the loose coupling of components, which allows for flexibility in the selection of internal and external business partners. However, currently this flexibility is severely limited by the fact that components have to provide not only the same functionality, but do so via virtually the same interface. Invocation-level mediation may be used to overcome this issue – using mediation interface differences can be resolved transparently at runtime. In this paper we present the general concepts of invocation-level mediation, and show how these ideas are integrated in our dynamic service invocation framework DAIOS. *To demonstrate the flexibility of our mediation framework we have implemented two fundamentally different mediation strategies, one based on structural similarity and one based on semantically annotated WSDL (SAWSDL). We evaluate the runtime performance of our mediation strategies, compare them with unmediated invocations and relate the overhead introduced by invocation mediation to the flexibility gained by this approach.*

1. Introduction

Systems based on the Service-Oriented Architecture (SOA) [15] decouple clients from the service providers they are using by utilizing standardized protocols and languages (HTTP, SOAP [22], WSDL [23]) and a registry such as UDDI [20] as service broker. In theory, this loose coupling allows service clients to roam freely between internal and external business partners, and always select the partner that is most appropriate at any given time. However, in practice this flexibility in the selection of partners is currently severely limited by the problem that clients rely on specific service interfaces for their invocation – therefore, services need to adhere to identical WSDL contracts in order to actually be interchangeable at runtime. The assumption of interface compatibility is of course not realistic if services are provided by different departments or companies.

Currently, most work in the area focusses on providing an additional infrastructure to resolve these compatibility issues: ESBs [17] provide an additional bus that decouples clients and services, and integration adapters or mediators [1, 18] are used as intermediary to resolve the inherent problems of invocation heterogeneity. The approach that we present within this paper follows a different idea: we use a pure client-side approach to mediation, i.e., we enable the clients themselves to adapt their invocation to specific target services. Specific mediation behavior is introduced in the clients using mediation adapters, which can either be general (e.g., a SAWSDL-based [24] semantic mediator) or tailored towards specific domains or scenarios. This lightweight approach removes the need for an explicit mediation middleware, and resembles the traditional idea of SOA (where clients and services interact directly) more closely.

The contributions of this paper are fourfold: firstly, we summarize the general concepts of dynamic invocation mediation; secondly, we present how the existing DAIOS Web service invocation framework has been extended to include a dynamic mediator interface, thirdly, we explain the implementation of a set of initial mediators that exemplify the capabilities of this interface, and finally we present the results of an initial evaluation of the DAIOS mediation interface.

The rest of this paper is structured as follows: Section 2 clarifies the need for invocation mediation based on a illustrative example, Section 3 elaborates on some related work in the field, Section 4 explains the general concepts of dynamic invocation mediation, Section 5 details the DAIOS mediation interface and the mediators that we have implemented using this interface, and Section 6 shows the results of a preliminary evaluation of these adapters. Finally, Section 7 concludes the paper with some final remarks, and an outlook on future work.

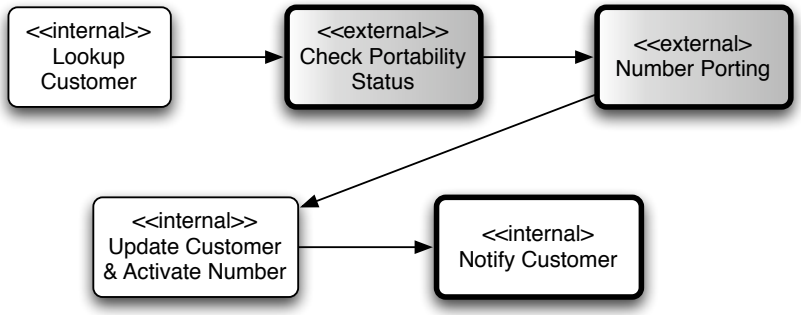

Figure 1. Number Portability Process

2. Motivating Example

To illustrate the need for runtime mediation and dynamic service invocation we now present a simple motivating example. Consider the problem of building a composite service for *cell phone number portability*. Number porting is a service pushed by the European Union that allows clients to take their mobile telephone number with them if they change their mobile provider. The number porting related business process of a provider may look as sketched in Figure 1 (simplified for clarity).

The process itself runs internally within the telecommunication provider where the customer recently signed a contract (which is a precondition of this process). After signing the contract, the new provider has to port the number from the previous telecommunications company. The process starts by looking up the customer using the internal `Lookup Customer` service. After finding the customer, the process has to send a message to the customer's former provider to check the portability status `Check Portability Status`). If, for some reason, the porting is not possible, the process is terminated and rescheduled to be executed at a later point (this is not shown in Figure 1 for reasons of brevity). After the portability check, a number porting request is sent to the old provider to release the number and transfer it (`Number Porting`). After that, the account of customer is updated with the ported number, and the account is activated (`Update Customer & Activate Number`). Finally, the customer is notified (via SMS, Mail, etc) that the porting is finished (`Notify Customer`).

In this workflow, only the activities `Lookup Customer` and `Update Customer & Activate Number` are provided by internal services, which can be assumed to have stable and relatively fixed interfaces. The activities `Check Portability Status` and `Number Porting` have to be carried out by external services provided by the respective previous telecommunications provider. Lastly, `Notify Customer` is an internal activity, which may be provided by a variety of services made avilable by different internal departments, depending on how the client should be notified.

This scenario illustrates how essential dynamic adaption to different service providers is – in some cases the services to invoke differ between instances of the same business process; in other cases the ability to dynamically exchange service providers simply adds value to the process by increasing overall flexibility. Of course it would be possible in this scenario to use BPEL [7] `Switch` and `Assign` statements to select the appropriate partner service and explicitly reformat the invocation input and output data according to the respective target service, but this approach unnecessarily complicates the business process by shifting what is essentially an implementation issue (selecting the right service provider in a given process instance) up into the business process. Additionally, this approach would only scale to a small and well-known number of alternate service providers – if the number of potential alternatives is very large, or if the alternatives change frequently this workaround quickly becomes unfeasible. Even worse, if the service to invoke (e.g., the service that implements the activity `Notify Customer`) has to be looked up dynamically in a service registry such as UDDI this approach fails at any rate.

3. Related Work

There's a lot of related work: (1) ESBs, (2) WSMX mediator [18], (3) requester-based mediation [11], (4) ad hoc invocation of sem. WS [6], (5) adapting service requests [3], adapters for WS integration [1], mediation-enabled Web service usage [4], requester-based mediation [11] and probably a 100 other papers one could cite. Have to distill that to get the most important ones.

This is a dummy text. This is a dummy text. This is a dummy text. This is a dummy text. This is a dummy text.

This is a dummy text. This is a dummy text.

4. Interface-Level Invocation Mediation

As we have explained earlier, most current work focusses on providing an additional infrastructure to resolve service incompatibilities. Our invocation mediation architecture, on the other hand, does not rely on any additional infrastructure; using our approach the client itself can adapt its invocation to a specific target service interface. This is achieved by extending clients with a mediation interface, which can be used to plug in a number of either general-purpose or domain-specific invocation mediators.

Generally, mediation can happen on two different levels: invocation-level mediation defines the mapping of messages (single invocations) between services, while protocol-level mediation considers resolving incompatibilities in the business protocol (invocation ordering) of services. Similar distinctions have previously been identified by different researchers (among others [1,4,11,18]). Per definition, protocol-level mediation is only important for stateful services, since stateless services do not rely on a specific ordering of invocations. Given that SOA traditionally focusses on stateless services we do not cover protocol-level mediation in this publication, and leave this topic for future work. However, others have already laid some ground work in this area [5,21].

4.1. General Concepts

Before going on to explain our mediation architecture, we need to define a number of general concepts that we are going to use in the forthcoming sections. Where applicable, we will use already existing, well-known terms (e.g., from the Semantic Web Services community [13]) instead of inventing new ones.

First of all, we have to distinguish between two different *formats*, *high-level (domain) concepts* and *proprietary (low-level) formats*. High-level concepts represent things and ideas that exist in the real world, i.e., which are independent from a concrete service or implementation. High-level concepts may (but do not necessarily need to) be concepts in a Semantic Web ontology [12,16]. Domain concepts are what domain experts talk about. Proprietary formats, on the other hand, are concrete implementations of high-level concepts. They are optimized towards concrete implementation goals, and are specific for single services. In general, proprietary formats motivate dynamic mediation – in the end, invocation-level mediation is the process of mediating between different low-level formats that implement the same domain concepts. Mediation between services is only senseful, if the services implement the same concepts, even though they are probably using different low-level formats to represent them.

High-level concepts are mutually comparable: for every pair of concepts it is possible to define whether they are *equal*, *compatible* or *different*. If a concept is compatible to another concept, it may be used whenever the other concept is expected. Note that this compatibility is non-symmetric – if concept A is compatible to concept B, B will usually not be compatible to A (otherwise, A and B would be equal). In some contexts it makes sense to define the degree of similarity between two concepts. In this case, the similarity of two equal concepts is 0, of two entirely different concepts the distance is defined to be ∞. For compatible concepts the similarity may be in the interval $[1;\infty[$, with higher values meaning "less similar".

The general operation of invocation-level mediation is the *transformation* of one format into another. We can distinguish three different types of transformation: (1) transforming high-level concepts into a low-level format is called *lowering* [8]; (2) the inverse operation, transforming propri-

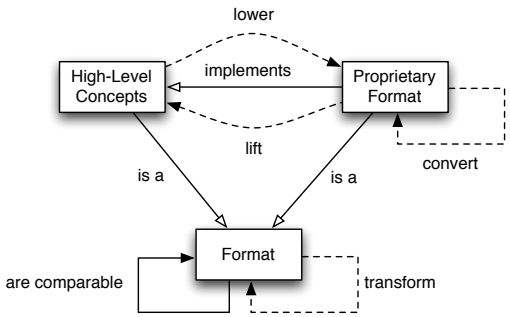

Figure 2. General mediation concepts

etary formats into domain concepts is called *lifting*; and (3) we refer to the transformation of one proprietary format into another as *conversion*.

These general concepts and their relationships are summarized in Figure 2.

4.2. Mediation Scenarios

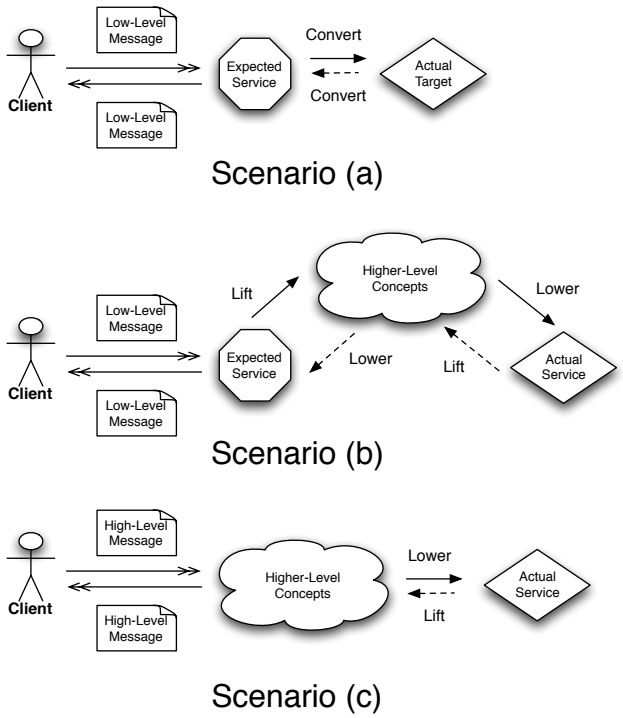

Figure 3. Mediation scenarios

From a high-level perspective we can distinguish three scenarios for invocation-level mediation (Figure 3). In scenario (a), a client's implementation expects a concrete service interface (`Service A`), but is actually invoking a different service (`Service B`). The invocation is mediated by converting the low-level format provided by the client directly into the format expected by the actual target service. Scenario (b) is similar, but in this scenario mediation is a two-step procedure. Firstly, the client invocation is lifted into more general domain concepts. Afterwards, this general representation is lowered into the proprietary format expected by `Service B`. The response of `Service B` is processed analogously (lifting to a high-level representation, lowering to the proprietary format used by `Service A`). Finally, in scenario (c) the client does not provide his input in a proprietary format, but already in the conceptual high-level representation. Obviously, this scenario is a special case of the ones explained earlier – in this case the processing is simpler since no lifting of the input and no lowering of the response is necessary.

These three scenarios are similar from an implementation point of view, but are conceptually different. The first two scenarios are typical for legacy clients, or clients that invoke a specific well-known service instance "most of the time", but still need to invoke other services with different contracts betimes. Speaking in terms of the example from Section 2, one can image that a client for the activity `Notify Customer` was implemented against an e-mailing service (since this is the usual way of notifying customers), but still needs to use a short message service (and probably a number of other customer interaction services) from time to time. The third scenario is characteristic for clients that have already been built with dynamic binding and runtime service selection in mind. In our example we can assume that clients for the activities `Check Portability Status` and `Number Porting` are implemented in such a way, since there is no "default" service that has to be used more often than others – in this cases, the service to use is entirely dependent on the concrete process instance. An invocation-level mediation architecture needs to address both of these scenarios adequately. In the first scenario, no explicit high-level conceptual representation is used. This eases the general mediation model, but leads to the well-known data integration problem of having to implement the conversion logics for $n(n-1)$ different conversions into the client-side mediator. Therefore, scenario (a) is only applicable if the number of possible service alternatives is small and well-known, while the scenarios (b) and (c) also scale to a higher number of alternate services.

5. Mediation Adapters

Following we will detail how client-side mediation as introduced in Section 4 has been implemented within the DAIOS [10] project. The general idea of DAIOS is to decouple clients from the services they are invoking by abstract-

ing from service implementation issues such as encoding styles, operations or endpoints. Therefore, clients only need to know the address of the WSDL interface describing the target service, and the input message that should be passed to it; all other details of the target service implementation are handled transparently. In DAIOS, data flowing into and out of services are represented by specific data structures (`DaiosMessges`). These messages are on a higher level of abstraction than e.g., SOAP messages, and can be represented as an unordered labelled tree structure. An example `DaiosMessage` representing a customer is shown in Figure 4. DAIOS implements mapping rules between `DaiosMessages` and WSDL / SOAP (more information on this mapping can be found in [10]).

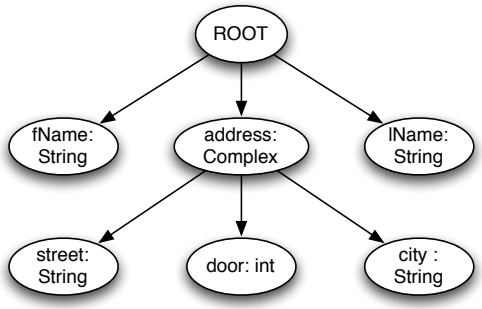

Figure 4. Example Daios Message

5.1. DAIOS Invocation Mediation

Even though DAIOS decouples clients from the service providers they are using by leveraging the concept of dynamic invocation, clients still need to know the exact structure of the message that the service expects. This data coupling is problematic – services from different providers will usually not rely on the same data model, even if the functionality that they implement is equivalent or similar. Therefore, we have extended DAIOS to include an interface that can be used to hook a *Chain of Mediators* into the client. The chain of mediators implements a stepwise transformation from the original input (which may be in the proprietary format of a different service, or directly representing high-level concepts) into the proprietary format expected by the target service. Input usually enters the chain encoded as `DaiosMessage` (since this is, what service clients deal with), and the output of the chain is SOAP (since this is what Web services expect. Therefore, the mediator chain should at some point contain the "default" mediator, which implements the mapping of the DAIOS-internal message format to SOAP (as indicated above).

In Figure 6, DAIOS' overall mediation architecture, and how it leverages the standard SOA model of service providers, consumers and registry. In the figure, we exemplify the mediation model based on the activity `Notify Customer` from the process in Figure **??**. Additionally, we assume that the client has been developed according to the mediation scenario (b) from Section 4. (1) A number of different messaging services are published in the service registry. The messaging services all have a similar domain purpose (sending messages to customers), but they are provided by providers and are accessible using a different interface. Note that we do not assume a public service registry [19] in this paper; instead, we assume a company-private service registry containing only well-known services provided in-house or by well-known partners, since such registries are more common in today's real-world service-based systems. (2) When the activity `Notify Customer` in the process has to be carried out, the client looks up a messaging service in the registry (according to the preferences of the customer) and constructs a DAIOS frontend to the service (for more details refer to [10]). (3) Finally, the client constructs a message in the proprietary format of one of the possible alternatives (the SMS service in the example) and commences the invocation. The message is now passed through the mediation chain of this client, and will be lifted to a common domain representation using well-defined transformation rules, and again lowered into the format expected by the actual target service. As a last mediation step, the "default" mediator serializes the message to SOAP. This SOAP message is then passed to the Web service stack included in DAIOS, and sent to the target service. If a return message is received as a response to the invocation, it travels through the mediator chain in the opposite direction, and passed back to the client in the proprietary format of the SMS service.

So far, we have only discussed the larger framework that enables mediation, and not the implementation of the mediators themselves. Concrete DAIOS mediators have to implement the Java interface sketched in Listing 1.

```
1  public interface IMediator {
2
3    public int similarity
4    (
5      IGrounding input, IWSDLType type
6    ) throws UnsupportedGroundingException;
7
8    public IGrounding transformToService
9    (
10     IGrounding input, IWSDLType target
11   )  throws UnsupportedGroundingException;
12
13   public IGrounding transformFromService
14   (
15     IGrounding input, IWSDLType target
16   )  throws UnsupportedGroundingException;
17
18 }
```

Listing 1. DAIOS Mediation Interface

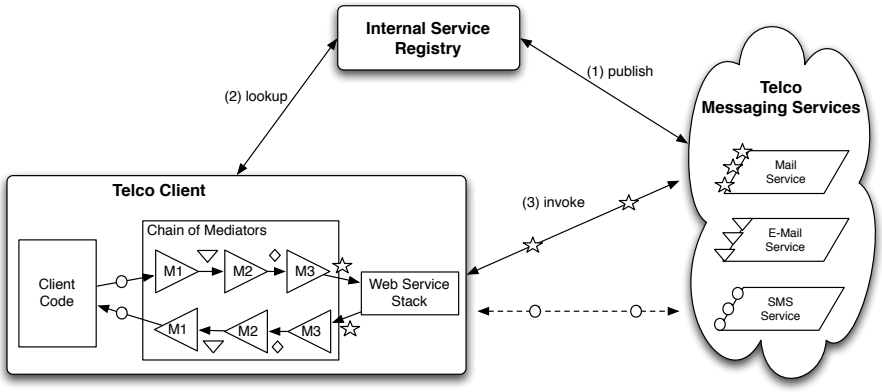

Figure 5. Client-Side Mediation Architecture

In the listing, the methods `transformToService` and `transformFromService` are used to implement the actual transformation from one representation (`IGrounding`) into another one. Typically, this transformation methods expect a specific input representation and emit messages in a specific output representation (e.g., the `transformToService` operation from the SAWSDL-based mediator described below expects input in form of high-level domain objects and lowers it into the proprietary format expected by the target service). If an unexpected representation is passed to the mediator, it should by convention throw an UnsupportedGroundingException. In this case, the mediator may have been inserted into the mediator chain in the wrong position. The second parameter of the operation, the target WSDL type, is used to steer the transformation (i.e., the target WSDL type represents the format that message should have after the transformation). Often, this target type will contain additional metadata necessary for transformation (e.g., in the case of the SAWSDL-based mediator this parameter contains the SAWSDL lifting and lowering mappings). A third public operation, `similarity`, can be implemented by mediators to numerically describe the amount of transformation necessary in order to mediate between a given input and a service interface (e.g., the `similarity` operation of the structural mediator explained below delivers the tree edit distance [2] as similarity metric). Regarding Figure 2 described earlier, the two transformation methods implement the various possible different transformations (lift, lower, convert), while the `similarity` method represents the "are comparable" association in the figure. Note that every mediator is free to implement any of the three mediation scenarios from Section 4.

Ultimately, it is the decision of the client how he wants to construct the mediation chain for every given invocation (i.e., which mediators should be used, and in which order). To do that, the client can choose from a set of general-purpose mediators. Additionally, domain-specific mediators can be implemented. Using such special-purpose mediators existing domain or service mapping knowledge can be re-used, e.g., existing mediation infrastructure can be integrated into the DAIOS model easily. In the following section we will detail the implementation of two very different general-purpose mediators, which come shipped with the DAIOS code. We believe that these mediators demonstrate the flexibility of our client-side mediation approach.

5.2. Structural Mediation

Sketch the implementation of the structural mediator. Explain what the advantages and disadvantages of this mediator are. Specifically, explain that the transformation of DaiosMessages results in a tree edit distance problem, which is NP-hard.

5.3. Semantic Mediation

Sketch the implementation of the SAWSDL mediator. Explain what the advantages and disadvantages of this mediator are. Specifically discuss how we integrate ontologies and stuff, how lifting and lowering happens, etc ...

6. Evaluation

For the evaluation we should at least do performance comparison between unmediated invocations and mediated ones (the two different versions), and show that we (hopefully) don't have a big performance penalty because of mediation. Obviously the performance penalty will be dependent on the degree of mediation necessary, therefore evaluate at least three different scenarios. We should do a good evaluation here

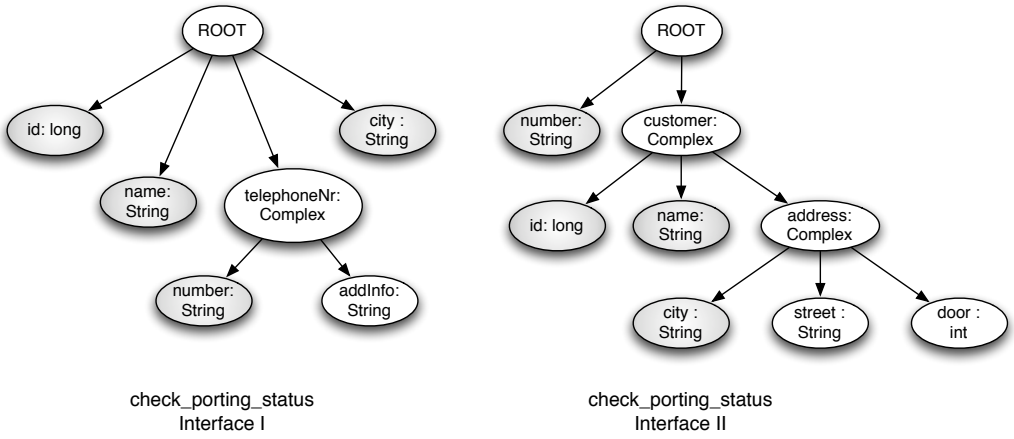

Figure 6. Structurally Different Service Interfaces

7. Conclusion

Currently, dynamic selection of services in SOA-based systems is severely limited by incompatibilities in the interface of these services. Enterprise integration solutions such as ESBs or mediation middleware can be used to resolve these problems, but these solutions add additional layers and complexity to the service-based systems built. In this paper we have presented a mediation architecture that enables the clients themselves to adapt to varying service interfaces. We have explained the general concepts of interface-level mediation, and how based on these concepts a mediation interface has been implemented in our existing DAIOS project. The implementation of two conceptionally different mediators has been used as a showcase of the flexibility of our approach. The current working Java prototype of DAIOS is available as open source software at Google Code[1]. This prototype includes the mediation interface described in this paper.

As part of our future work, we plan to extend the work presented in this paper in various directions: firstly, we plan to extend our approach to interface mediation also to mediation on protocol level; secondly, we are aiming at aligning DAIOS more closely with our SOA runtime environment VRESCO [9, 14], in order to provide an end-to-end SOA environment to clients.

8. Acknowledgements

The research leading to these results has received funding from the European Communitys Seventh Framework Programme [FP7/2007-2013] under grant agreement 215483 (S-Cube).

[1] http://code.google.com/p/daios/

References

[1] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani. Developing adapters for web services integration. In *CAiSE*, Lecture Notes in Computer Science, 2005.

[2] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[3] L. Cavallaro and E. D. Nitto. An approach to adapt service requests to actual service interfaces. In *SEAMS'08: Proceedings of the International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2008.

[4] E. Cimpian, A. Mocan, and M. Stollberg. Mediation enabled semantic web services usage. In *ASWC*, volume 4185, pages 459–473, 2006.

[5] M. Dumas, M. Spork, and K. Wang. Adapt or perish: Algebra and visual notation for service interface adaptation. In *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 65–80, 2006.

[6] A. Eberhart. Ad-hoc invocation of semantic web services. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, 2004.

[7] O. for the Advancement of Structured Information Standards (OASIS). Web Services Business Process Execution Language Version 2.0. `http://www.oasis-open.org/committees/download.php/18714/wsbpel-specification-draft-May17.htm`, 2007. Visited: 2008-04-28.

[8] J. Kopecky, D. Roman, M. Moran, and D. Fensel. Semantic web services grounding. In *AICT-ICIW '06: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services*, page 127, Washington, DC, USA, 2006. IEEE Computer Society.

[9] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. End-to-End Versioning Support for Web Services. In *Proceedings of the International Conference on Services Computing (SCC 2008)*. IEEE Computer Society, July 2008.

[10] P. Leitner, F. Rosenberg, and S. Dustdar. Daios – Efficient Dynamic Web Service Invocation. Technical Report TUV-1841-2007-01, Vienna University of Technology, 2007.

[11] B. Lin, N. Gu, and Q. Li. A requester-based mediation framework for dynamic invocation of web services. In *SCC '06: Proceedings of the IEEE International Conference on Services Computing*, 2006.

[12] A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16(2):72–79, 2001.

[13] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2), 2001.

[14] A. Michlmayr, F. Rosenberg, C. Platzer, and S. Dustar. Towards Recovering the Broken SOA Triangle - A Software Engineering Perspective. In *Proceedings of the 2nd International Workshop on Service Oriented Software Engineering (IW-SOSE'07)*, 2007.

[15] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 11, 2007.

[16] J. R. G. Pulido, M. A. G. Ruiz, R. Herrera, E. Cabello, S. Legrand, and D. Elliman. Ontology languages for the semantic web: A never completely updated review. *Knowledge-Based Systems*, 19(7):489–497, 2006.

[17] M.-T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen. The enterprise service bus: making service-oriented architecture real. *IBM Systems Journal*, 44(4), 2005.

[18] M. Stollberg, E. Cimpian, A. Mocan, and D. Fensel. A semantic web mediation architecture. In *CSWWS*, volume 2 of *Semantic Web And Beyond Computing for Human Experience*, 2006.

[19] A. Tsalgatidou and T. Pilioura. An overview of standards and related technology in web services. *Distrib. Parallel Databases*, 12(2-3):135–162, 2002.

[20] UDDI.org. UDDI Technical White Paper. `http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf`, 2000. Visited: 2007-07-31.

[21] S. K. Williams, S. A. Battle, and J. E. Cuadrado. Protocol mediation for adaptation in semantic web services. In *ESWC*, pages 635–649, 2006.

[22] World Wide Web Consortium (W3C). SOAP Version 1.2 Part0: Primer. `http://www.w3.org/TR/soap12-part0/`, 2003.

[23] World Wide Web Consortium (W3C). Web Services Description Language (WSDL) Version 2.0 Part 0: Primer - W3C Candidate Recommendation 27 March 2006. `http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327/`, 2006.

[24] World Wide Web Consortium (W3C). *Semantic Annotations for WSDL and XML Schema*, 2007. `http://www.w3.org/TR/sawsdl/` (Last accessed: Aptril 15, 2008).