Model-driven Adapter Development for Web Services Interactions

Hamid Reza Motahari Nezhad[1,2]

[1] The School of Computer Science and Engineering,
The University Of New South Wales
Sydney, NSW 2052, Australia
hamidm@cse.unsw.edu.au
[2] National ICT of Australia (NICTA)
Sydney, NSW 1430, Australia

1 Introduction

Application integration has been one of the main drivers in the software market during the late nineties and into the new millennium. The typical approach to integration and process automation is based on the use of adapters and message brokers [1, 13, 5]. Adapters wrap the various applications (which are in general heterogeneous, e.g., have different interfaces, speak different protocols, and support different data formats) so that they can appear homogeneous and therefore integrated more easily.

Web services have emerged as a solution to (or at least as a simplification of) the integration problem [1]. The main benefit they bring is that of standardization, in terms of data format (XML), interface definition language (WSDL), transport mechanism (SOAP) and many other interoperability aspects. Standardization reduces heterogeneity and makes it therefore easier to develop business logic that integrates different (Web service-based) applications. While standardization makes interoperability easier, it does not remove the need for adapters. In fact, although the lower levels of the interaction stacks are standardized, different Web services may still support different interfaces and protocols. For example, although two driving direction services may support XML and use SOAP as transport mechanism, they may still provide operations that have different names, different parameters, and different business protocols (that is, different constraints on the order in which they should be invoked for the clients to achieve a certain functionality).

The need for adapters in Web services comes from two sources: one is the heterogeneity at the higher levels of the interoperability stack (e.g., at business protocol level), and the other is the high number and diversity of clients, each of which can support different interfaces and protocols, thereby generating the need for providing multiple faces to the same service.

In this paper, we elaborate on the problem of developing adapters for Web service interactions. In the following, first, we characterize the problem of adaptation by identifying and classifying different kinds of adaptation needs in Web services (Section 2). Then, we briefly explain our preliminary contribution on adapter development for Web services (Section 3.1). Section 3.2 highlights the open problems, plans and research directions. Finally, Section 4 discusses related work.

2 Characterizing the Problem of Adapter Development for Web services

Interoperability among Web services, just like interoperability in any distributed system, requires that services use the same (or compatible) protocols, data formats, and semantics. Broadly speaking, we identify the following integration layers:

- **Messaging**. Any interaction requires information to be transferred among two or more parties. Hence, services should support messaging protocols interoperability, regardless of the syntax and semantics of the information. In Web services, the most common protocol at this layer is SOAP.
- **Basic Coordination**. The coordination layer is concerned with requirements and properties related to a set of message exchanges among two or more partners. For example, two or more services may need to coordinate to provide atomicity based on 2-Phase commit [1]. WS-Transaction is an example of specification at this level.
- **Business-level interfaces and protocols**. The above layers are concerned with the transfer of messages among services. To interact, services must also have compatible:
  - Interfaces (i.e., the set of operations supported by services). WSDL is the common standard for expressing interfaces.
  - Business protocols (i.e., the supported message exchange sequences by the service). These can be expressed for example using BPEL abstract processes, WSCI, or other protocol languages (see, e.g., [4, 1]).
- **Policies and non-functional aspects**. The definition of a service may include policies (e.g., privacy policies) and other non-functional aspects (e.g, QoS descriptions) that are useful for clients to understand if they can/want interact with the service.

Standardization efforts in Web services reduces the heterogeneities between service specifications, and thereby the number of required adapters. Nonetheless, standardization does not address all interoperability problems and does not remove the need for adaptation. One typical interoperability issue arises by standard evolution, so that services, which are using different versions of the same standard may not be able to interoperate. For instance, SOAP 1.2 allows the use of HTTP GET method, while SOAP 1.1 supports only HTTP POST method. This change is in the syntax of the specification, but it affects the interoperation of interacting services. Ryan et. al. [2] study this problem and employs a translator to translate the message from one format to another. The interoperation problem at the higher integration layers, e.g., business-level interfaces and protocols is more discernible. What standardization provides at these layers is a set of languages to define service specifications and hence leaves these specifications open to syntactical, structural and semantics heterogeneities, even though they are described using the same languages. Since interoperation at the other layers has been the subject of other research (e.g., [2]) or standardization, we focus on interoperability issues and adapter development at the business-level interfaces and protocols layer.

We classify the need for adaptation in Web services in two basic categories: adaptation *for compatibility* and *for replaceability*. The first category refers to wrapping a Web service so that it can interact with another service. It is needed when two services are functionality-wise compatible, but incompatible in interface or protocol specifications. The second category refers to modifying a Web service so that it becomes compliant with (i.e., can be used to replace) another service (see [4] for detailed discussion on compatibility and replaceability).

3 Model-driven Adapter development for Web services

The intended benefit of model-driven adapter development is to help programmers to develop adapters through a methodology and semi-automated code generation, starting from the protocol definitions. In the following, we first explain our preliminary contribution in this direction and then outline the open problems and future work.

3.1 Capturing Mismatches between Protocols as Design Patterns

Essentially, an adapter is a service which sits in between incompatible services and compensate for their differences [13]. This requires performing activities such as receiving and storing messages, transforming message data, and invoking service operations. These tasks can be modeled by process-centric service composition languages such as BPEL. We have analyzed interfaces and protocols of Web services to identify the most typical differences and for these we have captured the corresponding mismatches in form of mismatch patterns, which are design patterns [3]. Each mismatch pattern includes an adapter template, which is represented as a BPEL process skeleton to handle the mismatch, as well as a sample usage. The template can be used both as guideline for adapter developers and as input to a tool that automatically generates the adapter code.

We distinguish between two types of mismatches: *interface-level* and *protocol-level*. Mismatches at the interface-level characterize heterogeneities related to operation definition in WSDL interfaces. Examples at this level include operation signature mismatch and parameter constraint mismatch. The former is concerned with identifying differences and providing mappings between the signatures of operations (e.g., operation names, name and type of in/out messages and and their parameters), and the latter deals with potential mismatches between the constraints on parameters in the input/outputs messages (e.g., accepted value ranges). Protocol level mismatches characterize heterogeneities related to message choreography and temporal/transaction properties. Examples include differences on the order in which each protocol accepts a certain message, there is an extra (missing) messages in one protocol, or the case where the information required in one message in one protocol is captured in several messages in the other and so there is a need for a merge/split action.

Given the mismatch patterns, an analyst can identify the differences between business protocols by instantiating the mismatch patterns as many times as required. Currently, we are developing mechanisms to compose adapter templates. We are also implementing all research on adapter development using Java/J2EE as plug-ins in the Eclipse platform. These include a mismatch pattern editor and an adapter code generator, which generates BPEL code.

3.2 Towards an Extensible Framework for Model-driven Adapter Development

We believe the effective use and widespread adoption of service technologies and standards requires high-level frameworks and methodologies for supporting automated development and interoperability. The requirement of such a framework are as the followings:

- **(Semi-)Automatic identification of mismatches**. Our research so far does not address the problem of (semi-) automatically identifying differences between two business protocols. This problem is extremely challenging as any solution to it must take into account advances in multiple research disciplines including software component matching and adaptation [14, 13], protocol algebra for Web services [4] and schema matching [12] and semantic Web services [15–17]. This problem could be studied at two levels:
  - **Interface level**. The goal is to (i) (semi-)automatically identify mismatches between WSDL interfaces of services in terms of identified mismatch patterns, (ii) generate relevant mappings between operations and messages in the interfaces to be the input for the adapter templates in the mismatch pattern. It is a very challenging problem, and a variation of it has been the subject of other research [6, 9, 7], although they do not consider service interfaces with the constraint of protocol models.
  - **Protocol level**. The goal is to be able to semi-automatically discover mismatches between two protocols. We use an extended state machines formalism to specify protocols [4]. Protocols also can be represented using citeabstract process notion of process-centric languages such as BPEL. The research at this level is mostly related to adapter development for protocols in software components [13] and model-driven service development [10]. Our preliminary investigation into semi-automatic identification of mismatch shows that the problem of finding mismatches at the protocol level would be considerably simplified by identifying the mappings at the interface level. So, an approach to tackle this problem is to first elaborate on the problem of generating mappings at the interface level and then extend it to the protocol level.

– **A methodology to guide semi-automated adapter development**. There is a need for an extensible model-driven framework and a methodology to generate adapter code from given protocol models. The framework should be extensible to allow for interpolating of variant methods for interface and protocol mismatch identification, when they become available.

4 The State of the Art in Adapter Development

There has been substantial efforts and progress in in the area of Web services most of which has been focused on service description models and languages, standards, and on automated service discovery and composition [1]. Recently, researchers have considered the problem of similarity and compatibility at different levels of abstractions of a service specification (e.g., [4, 8, 6, 9, 7, 11, 16]).

In [6] and [9], techniques for assessing the similarity of WSDL interfaces of services are proposed for service discovery but these do not consider the problem of identifying interface mappings. In [7], a framework for handling differences among service interfaces is proposed, however, the assumption that Web service interfaces are derived from a common base limits its application to real world scenarios. This problem of identifying interface mapping is akin to that of identifying correspondence and mappings in schema matching [12] with the difference that in schemas there is no notion of messages and also the operation abstraction to relate them.

In terms of protocols specification and analysis, existing approaches provide models (e.g., based on pi-calculus or state machines) and mechanisms to compare specifications (e.g., protocols compatibility and replaceability checking) [4, 8]. Considering from the software engineering perspective, in [14], the authors focus on analyzing differences related to data types and to pre- and post-conditions in component interfaces. In [5] CORBA IDL is extended based on pi-calculus to incorporate protocol definition that is then used in checking for components compatibility and replaceability. In [13] the focus is on developing adapters for software components that have compatible functionality but incompatible interfaces and protocols. However, this approach assumes that interface mappings are provided. These efforts provide mechanisms that can be leveraged for Web service protocols adaptation, but are not sufficient. In fact, service protocols require richer description models than component interfaces and protocols. This is because clients and services are typically developed by separate teams, possibly even by different companies, and service descriptions are all that client developers have to understand to know how the service behaves.

Another related area is the work on semantic Web services [15–17]. The idea here is to empower the description of Web services with semantic information using ontologies. Two main approaches in this area are: describing services using ontology languages such as OWL-S[3] [15], and annotating service descriptions with semantics (e.g., like in WSDL-S[4])[16]. The main idea of the latter approach

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

[4] http://lsdis.cs.uga.edu/library/download/WSDL-S-V1.html

is to use the same set of WS-* family of standards as opposed to proposing new standards for Web service descriptions in OWL-S, and relax the choice of ontologies to some other alternatives in addition to OWL. The use of semantics information in form of shared ontologies diminishes the semantic heterogeneities of service interfaces and protocols, thus simplifies service adaptation [16, 17]. However, the issues related to defining reference ontologies to be shared, ontology compatibility checking, merging and customization has affected the wide acceptance of these approaches.

In a nutshell, in this work we aim at transforming the development of adapters for web services from a tedious and error prone work to a process that is model-driven and, to a large extent, automated. To the best of our knowledge, there is no existing work that considers this problem for Web services.

References

1. Alonso, G., Casati, F. , Kuno, H., Machiraju, V.: Web Services: Concepts, Architectures, and Applications. Springer, (2004)
2. Ryan N. D., Wolf, A. L.: Using Event-Based Translation to Support Dynamic Protocol Evolution. ICSE'04. Edinburgh, Scotland, United Kingdom (2004)
3. Benatallah, B., Casati, F., Grigori, D., M. Nezhad, H.R., Toumani, F.: Developing Adapters for Web services Integration. CAiSE, Porto, Portugal (2005)
4. Benatallah, B., Casati, F., Toumani, F.: Representing, Analysing and Managing Web Service Protocols. Data and Knowledge Engineering. Elsevier Science (2005)
5. Canal, C., Fuentes, L., Pimentel, E., Troya, J., Vallecillo, A: Adding Roles to CORBA Objects. IEEE TSE, 29(3), (2003)
6. Dong, X., Halevy, A. Y., Madhavan, J., Nemes, E., Zhang, J: Similarity Search for Web Services. VLDB Conference. Toronto, Canada, (2004)
7. Ponnekanti, S. R., Fox, A.: Interoperability among Independently Evolving Web Services. Middleware. Toronto, Canada (2004)
8. Bordeaux et al: When are two Web Services Compatible? VLDB TES, (2004)
9. Wang, Y., Stroulia, E.: Flexible Interface Matching for Web-Service Discovery, WISE, Roma, Italy, December (2003)
10. Baina, K., Benatallah B., Casati, F., and Toumani, F.: Model-Driven Web Service Development. CAiSE, Riga, Latvia, June (2004)
11. Wombacher, A., Mahleko, B., Fankhauser, P., Neuhold, E.: Matchmaking for Business Processes based on Choreographies. EEE. Taipei, Taiwan (2004)
12. Rahm, E., Bernstein, Ph. A.: A Survey of Approaches to Automatic Schema Matching. VLDB J. 10(4), (2001) 334-350.
13. Yellin, D. M., Strom, R. E.: Protocol Specification and Component Adaptors. ACM TOPLAS, 19(2), (1997)
14. Zaremski, A. M., Wing, J. M.: Specification Matching of Software Components. ACM TOSEM, 6 (4) , October (1997)
15. Martin, D., et. al.: Bringing Semantics to Web Services: The OWL-S Approach. SWSWPC, San Diego, USA (2004)
16. Patil, A.A., Oundhakar, S.A., Sheth A.P., Verma, K.: Meteor-s Web Service Annotation Framework. WWW '04, New York, USA (2004)
17. Williams, S.K.; Battle, S.A.; Cuadrado, J.E.: Protocol Mediation for Adaptation in Semantic Web Services. Hewlett-Packard Technical Report (HPL-78). May (2005)