

# ServiceMosaic Project: Modeling, Analysis and Management of Web Services Interactions

Boualem Benatallah<sup>1</sup>, Hamid Reza Motahari-Nezhad<sup>1,2</sup>

<sup>1</sup>University of New South Wales (UNSW)  
NSW 2052, Australia

<sup>2</sup>National ACT of Australia (NICTA)  
NSW 1430, Australia

{Boualem,hamidm}@cse.unsw.edu.au

## Abstract

This paper provides an overview of ServiceMosaic, which is a platform for model-driven analysis and management of service interactions. In particular, in this paper, we focus on business protocols modelling and analysis by providing operators for compatibility and replaceability checking of business protocols, and model-driven adapter development for business protocols.

**Keywords:** Web Services, Business Protocols, Compatibility, Replaceability, Adaptation.

## 1 Introduction

Application integration has been one of the main drivers of software market into the new millennium. The main benefits that Web services bring to integration are (i) support for loosely coupled and decentralized interactions and (ii) standardization at different levels such as message format (XML), interface definition language (WSDL), and transport mechanism (SOAP, typically over HTTP) (Papazoglou and Georgakopoulos 2003, Alonso et. Al 2004). Standardization helps reduce the costs of application integration, which are to a large extent due to the fact that different interacting entities have different interfaces, speak different communication protocols, and support different data formats and interaction models.

Although Web services provide abstractions to simplify the integration at lower levels of the interaction stacks (e.g., data syntax and communication protocols), where many of the issues have already been identified or even solved, they have not (yet) contributed to simplify integration at higher abstraction levels (e.g., data/message types and business-level interaction protocols). In this paper, focus on integration issues at the business protocols level. Generally stated, a business protocol specifies the ordering constraints on the message exchanges, which are allowed by the service in the interactions with other services. We developed a model-driven framework, called *ServiceMosaic*, for modeling,

analysis and management of Web service interactions. We provide an overview of this work. We focus on business protocols and refer to other papers for detailed descriptions of the different concepts and techniques developed in this project.

## 2 Analysis and Management of Web Service Protocols

In (Benatallah, Casati, and Toumani, 2005), we have developed a protocol algebra and protocol management operators that are targeted at three main types of analysis, which we believe to be essential to Web service analysis and management: *Compatibility* (i.e., assessing if two services can interoperate correctly), *replaceability* (i.e., verifying whether two different protocols can support the same set of conversations), and *consistency* (i.e., verifying whether the implementation of a service can support the declared protocol definition).

### 2.1 Modelling Business Protocols

Several languages exist for describing Web service protocols (e.g., WS-BPEL, WSCI, WS-CDL) (Papazoglou and Georgakopoulos 2003, Alonso et. al 2004). These languages are concerned more with implementation aspects than specification of protocol properties. They are not suitable for automating activities such as protocol compatibility and compliance analysis. Our framework features a simple, high level but expressive enough model to represent features and abstractions that are useful and needed in practice (Benatallah, Casati, and Toumani, 2004). This model builds upon the traditional state-machine formalism to represent message choreography constraints and extends it to cater for relevant protocol abstractions such as temporal constraints or the implications and the effects of service invocations from requester perspective.

We model a service business protocol (protocol for short) as a non-deterministic finite state machine, where the states represent the different phases that a service may go through during its interaction with a requestor. Transitions are triggered by messages sent by the requestor to the provider or vice versa (hence, transitions are labeled with either input or output messages). A message corresponds to the invocation of a service operation or to its reply. Each service may be simultaneously involved in several message exchanges (conversations) with different clients, and therefore can be characterized by multiple concurrent instantiations of

the protocol state machine. The purpose of the protocol is essentially to specify the set of conversations that are supported by the service.

The proposed model caters also for the modeling of transactional implications of operation invocations (Benatallah, Casati, and Toumani, 2004). We distinguish the following types of transitions of a protocol state machine. *Effect-less transitions* have no effect from the client's perspective. *Compensatable transitions* denote transitions whose effect can be cancelled. *Definite transitions* denote transitions whose transactional effects are permanent. *Resource-locking transitions* lock certain resources for the requester for a period of time. Temporal properties can also be specified by tagging transitions with a time interval, with the meaning that the transition is fired as the interval expires, hence leading the state machine to a new state from which previously invoked operations are not enabled any more (Benatallah, Casati, Ponge, and Toumani, 2005).

## 2.2 Compatibility Analysis

Compatibility analysis is concerned with verifying whether two services can interoperate. It is necessary for static and dynamic binding, and it also helps in evolution, since it helps verifying that a modified client can still interact as desired with a certain service. More precisely, we identified in two classes of compatibility:

- *Partial compatibility*: A protocol  $P_x$  is partially compatible with another protocol  $P_y$  if there are some executions of  $P_x$  that can interoperate with  $P_y$ , i.e., if there is at least one possible conversation that can take place among a service supporting  $P_x$  and a service supporting  $P_y$ .
- *Full compatibility*: a protocol  $P_x$  is fully compatible with another protocol  $P_y$  if all the executions of  $P_x$  can interoperate with  $P_y$ , i.e., any conversation that can be generated by  $P_x$  is understood by  $P_y$ .

These notions of compatibility are very useful in the context of Web services. For example, it does not make sense to have interactions with services for which there is no (partial or total) compatibility, as no meaningful conversation can be carried on. Furthermore, if there is only partial compatibility, the developer and the Web service middleware need to be aware of this, as the service will not be able to exploit its full capabilities when interacting with the partially compatible ones. When compatibility is partial, developers will likely want to know which conversations are allowed and which are not. We have developed an operator for compatibility checking that takes two protocols as input and returns the conversations that can take place between two services supporting these protocols (Benatallah, Casati, and Toumani, 2005).

## 2.3 Replaceability Analysis

Replaceability analysis helps identify if two protocols are equivalent in terms of conversations that they can support, both in general and when interacting with a

certain client. This can be for example used to determine whether a new version of a service (protocol) can support the same conversations as the previous one, or whether a newly defined service can support the conversations mandated by a given standard specification. We identified several replaceability classes, which provide basic building blocks for analyzing the commonalities and differences between service protocols:

**Protocol equivalence.** Two business protocols  $P_x$  and  $P_y$  are equivalent if they support the same set of conversations. Any conversation that is legal (i.e., does not result in errors) according to  $P_x$  will also be legal according to  $P_y$ , and vice versa. This means that the two protocols can be interchangeably used in any context and the change is transparent to clients.

**Protocol subsumption.** A protocol  $P_y$  is subsumed by another protocol  $P_x$  if  $P_x$  supports at least all the conversations that  $P_y$  supports. Hence, protocol  $P_x$  can be transparently used instead of  $P_y$  but the opposite is not necessarily true.

**Protocol equivalence and subsumption with respect to a client protocol.** It may be important to understand if a service can be used to replace another one when interacting with a certain client. This leads to a weaker definition of replaceability: a protocol  $P_x$  can replace another protocol  $P_y$  with respect to a client protocol  $P_c$  if every legal conversation between  $P_y$  and  $P_c$  is also a legal conversation between  $P_x$  and  $P_c$ . In this case,  $P_x$  can replace  $P_y$  to interact with  $P_c$ .

As for compatibility, the above discussion emphasizes the need for operators to analyze equivalence, subsumption, and different notions of replaceability. There is also the need for understanding, when two protocols are not equivalent, which conversations can be handled by both and which cannot. This leads to providing operators to determine *intersection* and *difference* among protocols, among others, to identify which conversations can and cannot be supported when a service is used in place of another (see (Benatallah, Casati, and Toumani, 2005) for details).

## 3 Model-driven Adapter Development for Web Service Protocols

When two services are not fully compatible or equivalent, we may consider adapting them. 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 with incompatible interfaces 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.

We take the view that although concrete adapter specifications are application-specific, in many cases it is possible to capture in a generic way the type of differences among protocol and the way to resolve them

into what we call *mismatch patterns*. Indeed, we have analyzed interfaces and protocols to identify the most typical differences and for these we have specified the corresponding mismatch patterns (Benatallah, Casati, Grigori, Motahari-Nezhad, and Toumani, 2005). This is akin to detecting structural and semantic differences in data mappings (Rahm and Bernstein, 2001). Each mismatch pattern includes an *adapter template* to tackle the mismatch, as well as a *sample usage*. The template can be used both as guideline for developers and as input to a tool that automatically generates the adapter code.

We distinguish between two types of mismatches: operation-level mismatches and protocol-level mismatches. Operation level mismatches characterize heterogeneities related to operation definitions. Such differences that occur when two services S and SR have operations with the same functionality but differ in operation name, number, order or type of input/output parameters. Protocol level mismatches characterize heterogeneities related to message choreography and temporal/transaction properties. Examples include differences that occur when two services expect a message in different order, when one service sends messages that the other does not accept, when one service requires a single message to achieve certain functionality while the other requires several, and so on. A comprehensive discussion is provided in (Benatallah, Casati, Grigori, Motahari-Nezhad, and Toumani, 2005).

#### 4 Discussion

There has been substantial efforts and progress 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 (Alonso, et al. 2004). Recently, authors have published papers that discuss similarity and compatibility at different levels of abstractions of service specifications (Bordeaux et al. 2004, Wombacher, et al 2004, Dong, et al. 2004). The proposed approaches do not provide fine-grained analysis operators. In addition, they do not consider message and protocol heterogeneities. In terms of protocols specification and analysis, existing approaches provide models (e.g., based on pi-calculus or state machines) and mechanisms (e.g., protocols compatibility and replaceability checking) to compare specifications for software components (Yellin, Strom, 1997, Canal, et al. 2003). These efforts can be leveraged for Web service protocols, but are not sufficient. In fact, service protocols require richer description models than component interfaces, as clients and services are typically autonomous and therefore service descriptions are all that client developers have to understand to know how the service behaves. Our framework builds upon existing protocol models and techniques to provide high level abstractions and operators for service protocols analysis and management. In addition, a component of the tool enables automated code generation (BPEL skeletons) from protocol models, therefore provides support for model-driven and automated Web service development (Baina, Benatallah, Casati, Toumani, 2004). This framework has been implemented in a prototype platform, called

ServiceMosaic, as a CASE toolset for modelling, analysing, and managing service models including business protocols, orchestrations, and adapters. The ServiceMosaic platform is developed using Java and J2EE technologies in the Eclipse platform ([www.eclipse.org](http://www.eclipse.org)).

Our current work focus is on extending analysis and management techniques for timed protocols (Benatallah, Casati, Ponge, and Toumani, 2005), and we will next concentrate on transactional aspects. We are also investigating business protocol discovery techniques to bring the benefits of protocols based interactions to services that do not explicitly model business protocols, or to interactions that involve groups of services. Finally, we plan to explore techniques for cataloging and analyzing previous adapters to improve the process of developing new adapters.

**Acknowledgment.** The work presented here, which is a part of ServiceMosaic project, is performed in collaboration of Fabio Casati, Farouk Toumani, and Julien Ponge.

#### 5 References

- Papazoglou, M.P. and Georgakopoulos, D. (2003): Special Issue on Service-Oriented Computing, *Communications of ACM*, (46)10, ACM Press.
- Alonso, G., et al. (2004): *Web Services: Concepts, Architectures, and Applications*. Springer Verlag.
- Benatallah, B., Casati, F., Grigori, D., Motahari-Nezhad, H. R., and Toumani, F. (2005): Developing Adapters for Web Services Integration, *Proc. of CAiSE'05*, Springer.
- Benatallah, B., Casati, F., Ponge, J. and Toumani, F. (2005): On Temporal Abstractions of Web Services Protocols, *Proc. of CAiSE Forum*. Springer.
- Benatallah, B., Casati, F., and Toumani, F. (2005): Representing, Analysing and Managing Web Service Protocols, *Data and Knowledge Engineering Journal*.
- Benatallah, B., Casati, F., and Toumani, F. (2004): Web Service Conversation Modeling: A Cornerstone for e-Business Automation, *IEEE Internet Computing*, (8)1. IEEE Press.
- Rahm, E., and Bernstein, P. A. (2001): On Matching Schemas Automatically, *VLDB Journal*, 10 (4).
- Yellin, D.M., and Strom, R.E. (1997): Protocol Specifications and Component Adaptors, *ACM TOPLAS*, 19(2).
- Baina, K., Benatallah, B., Casati, F., and Toumani, F. (2004): Model-Driven Web Service Development, *Proc. of CAiSE'04 Conference*, Springer.
- Canal, L., et al. (2003): Adding Roles to CORBA Objects, *IEEE Transaction on Software Engineering*, 29(3).
- Bordeaux et al (2004): When are two Web Services Compatible? *VLDB TES Workshop Proceedings*.
- Wombacher, A., et al (2004): Matchmaking for Business Processes based on Choreographies. *In proc. of EEE*. Taipei, Taiwan.
- Dong, X., et al. (2004): Similarity Search for Web Services. *VLDB Conference*. Toronto, Canada.