The Web Service Modeling Framework WSMF

D. Fensel[1] and C. Bussler[2]

[1] **Vrije Universiteit Amsterdam (VU)**
Faculty of Sciences, Division of Mathematics and Computer Science
De Boelelaan 1081a, 1081 HV Amsterdam, the Netherlands
Fax: +31-(0)84-872 27 22, phone: +31-(0)6-51850619
E-mail: dieter@cs.vu.nl

[2] **Oracle Corporation**
500 Oracle Parkway, Redwood Shores, 94065, CA, U. S. A.
Phone: +1-650-607-5684,
E-mail: chris.bussler@oracle.com

31. January 2002

Abstract. Web Services will transform the web from a collection of information into a distributed device of computation. In order to employ their full potential, appropriate description means for web services need to be developed. For this purpose we define a full-fledged *Web Service Modeling Framework (WSMF)* that provides the appropriate conceptual model for developing and describing web services and their composition (complex web services). Spoken in a nutshell its philosophy is based on the following principle: *maximal de-coupling* complemented by *scalable mediation service.*

1 Introduction

> *"Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes. ... Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service."*
> *IBM web service tutorial*

The current web is mainly a collection of information but does not yet provide support in processing this information, i.e., in using the computer as a computational device. Recent efforts around UDDI[1], WSDL[2], and SOAP[3] try to lift the web to a new level of service. Software programs can be accessed and executed via the web based on the idea of **web services**. A service can provide information, e.g. a weather forecast service, or it may have an effect in the real world, e.g. an online flight booking service. Web services can significantly increase the Web architecture's potential, by providing a way of automated program communication, discovery of services, etc. Therefore, the are in the centre of interests of various software developing companies.[4]

The ultimate vision is that a program tasked to achieve a result can use web services as support for its computation or processing. The program can discover web services and invoke them fully automated. Hence, it becomes a service requester. If the web services has a cost attached, the program knows when to search for a cheaper service and knows all possible payment methods. Furthermore, the program might be able to mediate any differences between its specific needs and a web service that almost fits.

In a business environment this translates into the automatic cooperation between enterprises. An enterprise requiring a business interaction with another enterprise can automatically discover and select the appropriate optimal web services relying on selection policies. They can be invoked automatically and payment processes can be initiated. Any necessary mediation is applied based on data and process ontologies and the automatic translation of their concepts into each other. An example are supply chain relationships where a manufacturing enterprise of short-lived goods has to frequently seek suppliers as well as buyers dynamically. Instead of constantly searching for suppliers and buyers by employees the web service infrastructure does it automatically within the defined constraints.

Still, there need to be done more work before the web service infrastructure can make this vision true. Current technology around UDDI, WSDL, and SOAP provide limited support in mechanizing service recognition, service configuration and combination (i.e.,

---

1. **UDDI** provides a mechanism for clients to find web services. Using a UDDI interface, businesses can dynamically lookup as well as discover services provided by external business partners. http://www.uddi.org/
2. **WSDL** defines services as collections of network endpoints or ports. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. http://www.wsdl.org/
3. **SOAP** is a message layout specification that defines a uniform way of passing XML-encoded data. http://www.soap.org/
4. http://www.w3.org/2001/01/WSWS/

realizing complex workflows and business logics with web services), service comparison and automated negotiation. Therefore, there are proposals such as **WSFL** [Leymann, 2001] that develops a language for describing complex web services or **DAML-S** [Ankolenkar et al., 2001] that employees *semantic web* technology ([Berneers-Lee et al., 2001], [Fensel & Musen, 2001], and [Fensel et al., 2002]) for service description. The **Web Service Modeling Framework (WSMF)** follows this line of research. It is a full-fledged modeling framework for describing the various aspects related to web services. Fully enabled E-commerce based on workable web services requires a modeling framework that is centered around two complementary principles:

- Strong *de-coupling* of the various components that realize an E-commerce application.
- Strong *mediation* service enabling anybody to speak with everybody in a scalable manner.

These principles are rolled out in a number of specification elements and an architecture describing their relationships.

The contents of the paper is organized as follows. In Section 2, we provide a motivation for web services, an analysis of the state of the art of this technology, we identify eight layers as being necessary to achieve automatic web service discovery, selection, mediation and composition into complex services, and introduces the main principles of WSMF. Section 3, provides the architecture and the main modeling primitives of WSMF. Section 4 discusses related work and Section 5 put web services in the context of the semantic web. Conclusions are provided in Section 6.

2 Web Services

This section briefly recalls the vision of web services. Then we analyze the current infrastructure web services are realized with and indicate future steps to go to make the vision workable.

2.1 The Vision

Web Services connect computers and devices with each other using the Internet to exchange data and combine data in new ways. Web Services can be defined as software objects that can be assembled over the Internet using standard protocols to perform functions or execute business processes. The key to Web Services is on-the-fly software creation through the use of loosely coupled, reusable software components. This has fundamental implications in both technical and business terms.[5] Software can be delivered and paid for as fluid streams of services as opposed to packaged products. It is possible to achieve automatic, ad hoc interoperability between systems to accomplish business tasks. Business services can be completely decentralized and distributed over the Internet and accessed by a wide variety of communications devices. Businesses can be released from the burden of complex, slow and expensive software integration and focus instead on the value of their offerings and mission critical tasks. Then the internet will become a global common platform where organizations and individuals communicate among each other to carry out various commercial activities and to provide value-added services. The barriers for providing new offerings and entering new markets will be lowered to enable access for small and medium sized enterprises. The dynamic enterprise and dynamic value chains become achievable and may be even mandatory for competitive advantage.

2.2 State of the Art

The web is organized around URIs, HTML, and HTTP. URIs provide defined ids to refer to elements on the web, HTML provides a standardized way to describe document structures (allowing browsers to render information for the human reader), and HTTP defines a protocol to retrieve information from the web. Not surprisingly, web services require a similar infrastructure around UDDI, WSDL, and SOAP.

UDDI provides a mechanism for clients to find web services. Using a UDDI interface, businesses can dynamically lookup as well as discover services provided by external business partners. A UDDI registry is similar to a CORBA trader, or it can be thought of as a DNS service for business applications. A UDDI registry has two kinds of clients: businesses that want to publish a service description (and its usage interfaces), and clients who want to obtain services descriptions of a certain kind and bind programmatically to them (using SOAP). UDDI itself is layered over SOAP and assumes that requests and responses are UDDI objects sent around as SOAP messages. The UDDI information contains four levels: the top level element is the Business entity, which provides general data about a company such as address, a short description, contact information and other general identifiers. This information can be seen as the *white pages* of UDDI. Associated with each business entity is a list of Business services.

5. See http://www.diffuse.org/WebServices.html.

These contain a description of the service and a list of categories that describe the service, e.g. purchasing, shipping etc. This can be considered as the *yellow pages* of UDDI. Within a business service, one or more Binding templates define the *green pages*: they provide the more technical information about a web service (cf. [Lemahieu, 2001]).

WSDL defines services as collections of network endpoints or *ports*. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions of messages, which are abstract descriptions of the data being exchanged, and port types, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a binding. A port is defined by associating a network address with a binding; a collection of ports define a service.

SOAP is a message layout specification that defines a uniform way of passing XML-encoded data. In also defines a way to bind to HTTP as the underlying communication protocol for passing SOAP messages between two endpoints. Instead of being document-based, automated B2B interaction requires integration of processes. However, although techniques such as DCOM, RMI and CORBA are successful on the local network, they largely fail when transposed to a web environment. They are rather unwieldy, entail too tight a coupling between components and above all conflict with existing firewall technology. Replacing this by a simple, lightweight RPC-like mechanism is the aim of SOAP. SOAP uses XML messaging over plain HTTP, thus avoiding firewall problems (asynchronous communication can also be accomplished via SMTP). Hence SOAP is basically a technology to allow for "RPC *over the web*" providing a very simple one-way as well as request/reply mechanism.

2.3 Eight layers of functionality required for successful web services

UDDI, WSDL, and SOAP are important steps into the direction of a web populated by services. However, they only address part of the overall stack that needs to be available in order to achieve the above vision eventually. The following layers are identified as being necessary to achieve automatic web service discovery, selection, mediation and composition into complex services, see also [Bussler, 2001c].

1) **Document types**. Document types describe the content of business documents like purchase orders or invoices. The content is defined in terms of elements like an order number or a line item price. Document types are instantiated with actual business data when a service requester and a service provider exchange data. The payload of the messages sent back and forth are structured according to the document types defined.

2) **Semantics**. The elements of document types must be populated with correct values so that they are semantically correct and are interpreted correctly by the service requesters and providers. This requires that vocabulary is defined that enumerates or describes valid element values. For example, a list of product names or products that can be ordered from a manufacturer. Further examples are unit of measures as well as country codes. Ontologies provide a means for defining the concepts of the data exchanged. If ontologies are available document

types refer to the ontology concepts. This ensures consistency of the textual representation of the concepts exchanged and allows the same interpretation of the concepts by all trading partners involved. Finally, the intent of an exchanged document must be defined. For example, if a purchase order is sent, it is not clear if this means that a purchase order needs to be created, deleted or updated. The intent needs to make semantically clear how to interpret the sent document.

3) **Transport binding**. Several transport mechanisms are available like HTTP/S, S/MIME, FTP or EDIINT. A service requester as well as service provider have to agree on the transport to be used when service requests are executed. For each available transport the layout of the message must be agreed upon and how the document sent is represented in the message sent. SOAP for example defines the message layout and the position within the message layout where the document is to be found. In addition, header data are defined required for SOAP message processing.

4) **Exchange sequence definition**. Communication over networks are currently inherently unreliable. It is therefore required that service requester and service provider make sure themselves through protocols that messages are transmitted exactly once. The exchange sequence definition achieves this by defining a sequence of acknowledgment messages in addition to time-outs, retry logic and upper retry limits.

5) **Process definition**. Based on the assumption that messages can be exchanged exactly once between service requester and service provider the business logic has to be defined in terms of the business message exchange sequence. For example, a purchase order might have to be confirmed with a purchase order acknowledgment. Or, a request for quotation can be responded to by one or more quotes. These processes define the required business message logic in order to derive to a consistent business state. For example, when good are ordered by a purchase order and confirmed by a purchase order acknowledgment they have to be shipped and paid, too.

6) **Security**. Fundamentally, each message exchange should be private and unmodified between the service requester and service provider as well as non-reputable. Encryption as well as signing ensure the unmodified privacy whereby non-repudiation services ensure that none of either service requester or service provider can claim not to have sent a message or a different one.

7) **Syntax**. Document can be represented in different syntaxes available. XML is a popular syntax, although non-XML syntax is used, too (e.g. EDI[6]).

8) **Trading partner specific configuration**. Service requesters or service providers implement their business logic differently from each other. The reason is that they establish their business logic before any cooperation takes place. This might require adjustments once trading partners are found and the interaction should be

6. http://www.x12.org/

formalized using web services. In case modifications are necessary trading partner specific changes have to be represented.

Many organizations had the insight that message definition and exchange are not sufficient to build an expressive web services infrastructure. In addition to UDDI, WSDL and SOAP standards for process definitions as well as exchange sequence definitions are proposed such as WSFL [Leymann, 2001], XLANG [Thatte, 2001], ebXML BPSS [Waldt & Drummond], BPML [Arkin, 2001] and WSCL [Christensen et al., 2001]. Still, there are important features missing in all of the mentioned frameworks. Very important is to reflect the *loose coupling* and *scalable mediation* of web services in an appropriate modeling framework. This requires mediators that map between different document structures and different business logics as well as the ability to express the difference between public visible workflows (public processes) and internal business logics of a complex web service (private processes). Therefore, we developed a full-fledged **Web Service Modeling Framework (WSMF)**. It provides a rich conceptual model for the development and the description of web services bringing this technology to its full potential. In section 4, we provide a detailed comparison of WSMF with the other proposals.

2.4 The main principles of WSMF

There are important steps to take to bring web services and fully enabled E-commerce to reality. Bringing E-commerce to its full potential requires a Peer-to-Peer (P2P) approach. Anybody must be able to trade and negotiate with everybody else. However, such an open and flexible E-commerce has to deal with many obstacles before it becomes reality:

- Mechanized support is needed in finding and comparing vendors and their offers. Currently, nearly all of this work is done manually which seriously hampers the scalability of electronic commerce. Semantic Web Technology can make it a different story: Machine processable semantics of information allows to mechanize these tasks. We will further discuss this aspect in Section 5.
- Mechanized support is needed in dealing with numerous and heterogeneous data formats. Various “standards” exists on how to describe products and services, product catalogues, and business documents. Ontology technology (cf. [Fensel, 2001]) is required to define such standards better and to map between them. Efficient bridges between different terminologies are essential for openness and scalability.
- Mechanized support is needed in dealing with numerous and heterogeneous business logics. Again, various “standards” exist that define the business logic of a trading partner. Mediation is needed to compensate these differences, allowing partners to cooperate properly (cf. [Bussler, 2001a]).

The later two requirements are explicit rationales underlying the development of WSMF. Fully enabled E-commerce based on workable web services requires a modeling framework that is centered around two complementary principles:

- Strong *de-coupling* of the various components that realize an E-commerce application. This de-coupling includes information hiding based on the difference

of internal business intelligence and public message exchange protocol interface descriptions (see [Bussler, 2001a]). Coupling of processes can only be achieved via interfaces to keep the amount of interactions scalable.

- Strong *mediation* service enabling anybody to speak with everybody in a scalable manner. This mediation service includes the mediation of different terminologies [Fensel, 2001] as well as the mediation of different interaction styles [Bussler, 2001a].

The design of the WSMF is organized around these two principles. We de-couple various aspects of web service enabled E-commerce to the maximum and provide scalable interaction on the basis of public interfaces and mediation service. None of the other approaches we found in the literature take our point to such an extend. We will find many (intended) similarities to existing approaches (see Section 4), however, none of them would really provide scalable interoperability.

3 The Web Service Modeling Framework WSMF

The WSMF consists of four main different elements: *ontologies* that provide the terminology used by other elements, *goal repositories* that define the problems that should be solved by web services; *web services* descriptions that define various aspects of a web service; and *mediators* which bypass interoperability problems.

3.1 Ontologies

Ontologies (cf. [Fensel, 2001]) are key enabling technology for the semantic web. They interweave human understanding of symbols with their machine processability. Ontologies were developed in Artificial Intelligence to facilitate knowledge sharing and re-use. Since the early nineties, Ontologies have become a popular research topic. They have been studied by several Artificial Intelligence research communities, including Knowledge Engineering, natural-language processing and knowledge representation. More recently, the concept of Ontology is also becoming widespread in fields, such as intelligent information integration, cooperative information systems, information retrieval, electronic commerce, and knowledge management. The reason ontologies are becoming so popular is largely due to what they promise: a shared and common understanding of a domain that can be communicated between people and application systems. In a nutshell, Ontologies are formal and consensual specifications of conceptualizations that provide a shared and common understanding of a domain, an understanding that can be communicated across people and application systems. Thus, Ontologies glue together two essential aspects that help to bring the web to its full potential:

- Ontologies define formal semantics for information, consequently allowing information processing by a computer.
- Ontologies define real-world semantics, which makes it possible to link machine processable content with meaning for humans based on consensual terminologies.

In our framework ontologies are used to define the terminology that is used by other elements of WSMF specifications. Therefore, they enable reuse of terminology as well as interoperability between components referring to the same or linked terminology.

3.2 Goal Repositories

The description of a *goal*[7] specifies objectives that a client may have in case he consults a web service. A goal specification consists of two elements:

- **Pre-conditions** describe what an web service expect for enabling it to provide its service. Asking for the *nearest* Cuban restaurant only makes sense if I can tell the web service where I am in a way it understands. Asking for the *best* hotel in a city requires that I have explained him what I understand with “best”.
- **Post-conditions** describe what a web service returns in response to its input. In return to a geographical information it may return the closest Cuban restaurant or it may return the best hotel according to the preferences of a client.

Goal specifications should be kept separate from actual web service description because

7. [Fensel et al., to appear]

there is an *n*2*m* mapping between them, i.e., the same web service can serve different goals and obviously different (competing) web services can serve the same goal. For example, Amazon could be used to buy a book, however, in the same way it can be used as an information broker on bibliographic information about books. Vice versa, different book stores may subscribe to the same goal.

Always, at least one ontology is imported by a goal specification to further define the terms that are used to describe it. For example, the relationship *nearest* should fulfill certain properties:

- *nearest*(*x*) **implies** there exists no *y* and *y nearer* than *x*, i.e., *nearer*(*y*,*x*) is false; *x* is an *optima*.
- **not** *nearer* (*x*,*x*), i.e., *nearer* is *irreflexive.*
- *nearer*(*x*,*y*) **and** *nearer*(*y*,*z*) **implies** *nearer*(*x*,*z*), i.e., *nearer* is *transitive.*
- *nearer*(*x*,*y*) implies **not** *nearer*(*y*,*x*), i.e., *nearer* is asymmetric.
- *nearer*(*x*,*y*) **or** *nearer*(*y*,*x*), i.e., *nearer* is linear.

And there are even more properties than could or could not enforced for such a relationship. Such properties determines heavily whether there is one, many, or no solution for a goal. Ontologies provide reusable vocabulary with precisely defined properties. Software that handle one, many, or potential no solution can be securely used by referring to an ontology with precisely defined properties of its concepts and relationships. Without such precision, a web service that assumes transitivity of a nearer relationship may fall in endless sleep[8] if the actual relationship does not provide this property.

3.3 Web Service

> *The complexity is in the eye of the observer …*
> *as much as the object allows.*

Many web service description languages distinguish between elementary and complex web services. Elementary web services are simple input/output boxes, whereas complex web services decompose the overall process into subtasks that may call other web services. Strictly spoken such a distinction is wrongly and may lead to miss-conceptualizations in a web service modeling framework. It is not the complexity of the web service that makes an important distinction. It is rather the complexity of its *description* or its interface (in terms of static and dynamics) that makes a difference. A complex web service such as a logical inference engine with a web interface can be *described* rather elementary. It receives some input formulas and derives--after some while--a set of conclusions. A much more simpler software product such as a simple travelling information system may be decomposed into several web services around hotel informations, flight informations, and general information about a certain location. Therefore, it is not the inherent complexity of a web service, it is the complexity of its external visible description that makes the relevant difference in our context. This insight may look rather trivial, however, has some important consequences:

---

8. That is, it stays in an endless loop.

1) Many web service description approaches such as WSFL or DAML-S do not make an explicit distinction between an internal description of a web service and its external visible description. The provide description means such as data flow diagrams and control flow descriptions without making clear whether they should be understood as interface descriptions for accessing a web service, or whether they should be understood as internal description of the realization of a web service. In our framework we strictly limit ourselves on describing the external aspects of a web service, i.e., its interface that can be accessed via a network. Therefore, a complex web service can have a very simple description and vice versa. In a nutshell, we do not describe a web service but its interface accessible via a network. Often, the internal complexity of a web service reflects the business intelligence of a web service provider. Therefore, it is essential for him not to make it public accessible. This is the major conceptual distinction between an internal description of the workflow of a web service and its interface description. *"The clear separation between private and public processes is key to provide the necessary isolation and abstraction between enterprise internal processes and processes across enterprises." [Bussler, 2001a]*

2) The dichotomy of elementary and complex web services is too simplistic. As we talk about the complexity of the *description* of a web service we naturally provide a scale of complexity. That is, we start with some description elements and gradually upscale the complexity of available description elements by adding additional means to describe various aspects of a web service.

In the following we will elaborate on the various elements a web service may use to have a description with a certain level of complexity. We start with the question: what is a web service? It is a tool that helps to achieve a certain goal and it is accessible via the web. This already provides some of the elements required to describe a web services. In any case we describe them as black boxes, i.e., we do not want to model internal aspects on how such a service is achieved.

First, a web service has a **name**, i.e., a unique identifier to refer to him. This requirement is elementary.

Second, a web service fulfills a certain purpose, i.e., it should have a **goal reference**. This is the goal a service can achieve.

Third, like goals, web service descriptions contain **pre conditions** and **post conditions** as introduced for goal descriptions. The pre condition is the condition that has to be true for the input in order for the web service being able to successfully execute. The post condition is the condition that holds once the complex service has been executed successfully, i.e., it defines constraints on its output. These conditions of a web service can be linked either directly or indirectly with a mediator to goal conditions. This flexibility of the WSMF architecture has two main advantages. First, a goal and a web service can use different terminology, i.e., a certain goal provides a certain terminological view on input and output of a web service. Second, a web service can *strengthen a pre condition* or *weaken a post condition* of a goal.[9] Our goal may be to

9. Cf. [Fensel & Benjamins, 1998].

find the nearest restaurant to our current location. A restricted service to achieve this goal may asked for our location according to a certain standard for representing geographical information and may only provide the restaurants that are registered at it or that pay marketing fees to it. In these cases, the service is not perfect according to an abstract goal description but may still fulfill our current needs or may just be the best we can find (or want to pay for) at the moment. The opposite cases where a web service weakens a pre condition or strengthen a post condition does not require explicit mediation because in these cases, each result of the web service fulfills the goal descriptions.

Forth, a web service description describes the structure of its **input data** and **output data**. The element through which data are passed to complex services are *input ports*. Input ports behave like formal parameters in that data values are passed to them as actual parameters at runtime. However, in contrast to formal parameters, data can be passed to input ports from the requester concurrently to the complex service execution. This allows the passing of data whenever the complex service needs them (if at all). Analogous to input data passed through input ports, output data can be returned by a web service through *output ports*. This allows a service to return data concurrently to the complex service execution. This allows, for example, to make results available to the requester as soon as possible without waiting for the complex service to be finished first.

Fifth, **error data** can be returned from the complex service through error ports at any time to indicate problems or error states. Several error ports can be defined in order to return error data at different points in the execution (compare [Florescu & Grünhagen, to appear]).

These description elements still tried a web service as a black box. In many cases, we may want to have a more complex description of the interface of a web service.

- **Failure**. If a failure occurs within one of the invoked elements of a services, then the service is in a failure state. If the service cannot recover from the failure itself, it has to make the failure state known to the service requester for it to try to deal with the failure. It is very important for the requester to know exactly which of the invoked elements of a services succeeded and which failed in order to be able to deal with the failure state. If the decomposition would not be known, the requester would have no way finding out. For example, if a service “book_trip” books a hotel, a car and a return flight then all the three bookings need to succeed for the “book_trip” service to be successful. If it fails it is important to know which bookings took place and which not in order for the service requester to cancel the successfully booked parts.
- **Concurrent execution**. Since a web service may call several other web services it might take some time until it finishes. A requester might choose to execute some local logic concurrently to the service to optimize resources or overall execution time. A synchronous invocation model does therefore not work in this case. For example, if “book_trip” uses an auction to get the best flights available it might take 24 hours to complete. It is important to know that this web service takes that long.

- **Concurrent data input and output**. A web service may invoke other services over time. Each time a service is invoked, input data have to be given to it and output data have to be taken from it. It might be that input data for a service are not known at the time the web service is invoked. This means that it must be possible to give input data to a web service while it is already executing for the web service to pass it on to the services it has to invoke. For example, if the hotel requested in "book_trip" is not available the "book_trip" service might want to return a list of alternative hotels to the requester for the requester to make another choice.
- **Dynamic service binding.** A web service calls other services. This means that when the web service is implemented a choice has to be made which service to call. The requester of this service executing the service has no possibility to change the services called and to replace them with the requester's preferences. For example, "book_trip" might try to book trips directly from the airlines instead of a travel service that is independent. The service requester might want to change this and use a last-minute flight selling service instead.

In consequence, we provide additional description elements to characterize a web service in terms what it is expecting and what it is providing in return.

Sixth, a web service in turn may invoke other web services to provide its service. For each invoked web service a proxy called **invoked web service proxy** has to be declared. A proxy may either consists simple of a goal definition or of a name, pre conditions, post conditions, input ports, output ports as well as error ports and a binding to determine a concrete web service at runtime. The proxy is used later on by the complex sequencing rules defining which service is invoked at which point in the overall complex service. The proxy allows to refer to a service without knowing at define time which concrete service is bound. For example, in the "book_trip" complex service there is a proxy called "book_flight" with an input port for origin and destination and an output port that returns the flight itinerary. The "book_trip" web service books the flights first, then the hotel and then the car. "book_flight" is therefore executed first. However, at definition time it is not decided yet, which concrete service is used. For example, it is not defined if airlines are contacted directly or a travel service or a last-minute flight selling service. This binding happens at runtime based on binding rules defined in the proxy. The binding can be fixed to precisely one service, it can be defined as a lookup in e.g. UDDI or it can be dependent on input data coming from an input port. The last case means that the requester has full control over which service to select at runtime because it can specify the binding criteria through input ports. The only condition is that the data required to execute the complex service can be provided by the service bound at runtime.

Seventh, a web service exposes input ports and output ports. Each invoked web service proxy also exposes input ports and output ports. For each input port of a web service a decision has to be made to which input ports of the invoked web service proxies the data values have to be forwarded. This might be to none, one or several input ports of invoked web service proxies. Each connection between a complex service's input port and a invoked web service proxy's input port is a **data flow**. Data flow have to be defined also for invoked web service proxies' output ports and the output ports of the complex web service. Furthermore, output ports of invoked web service proxies might

be connected to input ports of other invoked web service proxies. This is the way to have results of one invoked web service be input of one or several subsequent invoked web services.

- Data flow can be conditional from one input port of the web service to two or more different input ports of invoked web services proxies. The condition contains a Boolean expression that binds to the data values in the input port. Depending on the result of the expression, the subsequent data flow is executed at runtime. This allows to have data flow conditionally to different invoked services. For example, if a first class flight ticket is requested then this might require a travel web service that deals with first class flight tickets.
- Another construct is a “split” where e. g. from on web service input port the values are forwarded to several subsequent input ports at the same time. In this case several invoked web services require the same input data. The split can be “by value” and “by reference”. “by value” means that the data in the subsequent input ports are copies of each other. “by reference” means that the data in the input ports are replica of each other (i. e. a single image). The latter one is often implemented by several references pointing to the same data values.
- Input data on either end of a data flow might not match. In this case a type cast is necessary to make sure that the data passed between ports match according to their type. A data flow can contain a type cast that is used at runtime to make sure that “flown” data are transformed en-route. Through this mechanism binding of different data types is possible.
- While a “split” is easy to achieve, a “join” is more difficult since several data values have to be merged into one. For this case the notion of a “step” is introduced. A step is defined like a invoked web service proxy. However, its implementation does not invoke a web service at runtime. Instead, execution logic is executed that is implemented e. g. with Java. This logic can take the input data of the step input ports and return data to the step’s output ports. A step allows to implement a join function that takes several data values and joins them correctly. Through such a step joining data flows can be achieved. Of course, if a web service for joining data exists, an invoked web service proxy can be used for joining data values, a step is not necessary in this case.

Eighth, data flow implicitly determines the execution sequence of invoked web service proxies as well as the steps within one complex service implementation. However, no all necessary execution sequences can be handled by data flow. For example, if data flow allows two invoked web services to be executed in parallel and they should be executed in a sequence, data flow is not capable of expressing this directly. The only way would be to have an artificial data flow between the two invoked web services. While this is possible, it is not good modeling practice at all. Instead, a **control flow** sequence should be introduced between the two invoked web services that defines the correct execution sequence. In this case no artificial data has to flow at all. Other control flow constructs in addition to sequence are conditional branching, for-loops, while-loops as well as parallel execution. All these constructs are available to define the appropriate execution sequence of invoked web services and steps.

Control flow and data flow both implement the external accessible par of the business logic of a web services. With both set of modeling constructs it can be made sure that the web service execution achieves the desired outcome.

Ninth, web services may require **exception handling**. Invoked web services can fail and return an error or exception code. In this case, depending on the error code, exception handling must take place in order to deal with the error situation. The web service execution is in error state since an invoked web service was not able to execute correctly as planned by the complex service. The invoked web service proxy contains an exception handling section that detects at runtime if an invoked service returns an error code. If this is the case, it is possible to specify a retry. If a retry is specified then the failed invoked web service is invoked again with the expectation that it works in the case of a retry. If it returns successfully, the execution of the web service continues. Otherwise the error code is returned in an error port of the invoked web service. From there it can be picked up by compensation logic (discussed next).

Tenth, after an invoked web service finally failed the invoking web service is in error state. One possibility is that it notifies the service requester about the error state by returning the error code to the requester through an error port. In this case the requester has to deal with the error state and react to it. For example, if a hotel booking failed, the requester had to find out how to find an alternative hotel. If this does not succeed, the requester had to cancel the flights that are already booked. Alternatively, the complex service itself can implement a strategy of **compensation** for a failed invoked web service. Upon failure of an invoked service the service can define a compensation strategy for the failure. A compensation construct allows to define what happens after an invoked web service finally failed. Fundamentally, the compensation construct represents another web service, the compensation. In the compensation invoked web service proxies can be used to invoke web services as well as steps. For example, if the hotel booking failed, the corresponding compensation can search for alternative hotels within a radius of 5 miles of the original hotel. In order to do the search, a hotel search service is invoked. Once hotels are found, attempts are made to reserve a room at each of them until one attempt succeeds or all attempts fail. If one attempt succeeds then the complex service continues as planned with booking a car. However, if all attempts fail, the flight reservation is canceled. After the cancellation succeeds the web service returns to the requester with an error code indicating that no hotel could be found. All side effects (like in this case the flights) are cleaned up. Of course, compensation can be very sophisticated. For example, if no attempt succeeds in booking an alternative hotel, the return to the requester could be a question if the flight should be kept or canceled.

Eleventh, web services need description related to the **message exchange protocol**. Messages from a web service requester to a web service provider and vice versa are sent over networks like the Internet. Networks can be reliable as well as unreliable. Reliable networks guarantee that a sent message will be delivered. Unreliable networks don’t guarantee this. Web services assume that the transmission of messages is exactly once. In the case of reliable networks an exactly once message transmission can be achieved through duplicate detection. Since the network guarantees that messages are delivered, only duplicates have to be detected and removed in order to achieve exactly once semantics. In order to achieve exactly once semantics over an unreliable network more

work has to be done. First, guaranteed delivery has to be achieved. This is done by message retries and time-outs in conjunction with retry upper limits. If a message is sent and no acknowledgment is received in a given time-out period, the message is assumed to be lost. A resent of the message happens and the time-out starts again. In case the acknowledgment is again not received in the given time-out a resent of the message happens. In order to not get into an endless loop, an upper limit restricts the number of resents. If the upper limit is reached, the message transmission failed. If duplicate detection is implemented on top of this, exactly once message delivery is guaranteed. The implementation of exactly once message delivery is called the message exchange protocol. This protocol is a separate layer by itself and provides an exactly once semantics to web services. It deals with the different type of networks and their properties and provides a nice abstraction.

Twelfth, there are important **non functional properties** that characterizes a web service. Examples are the geographical reach of a service (e.g., a web-based flower shop), the price related to using a service, or the average/maximal time it may take it to produce its output.

3.4 Mediator

Adapters are of general importance for component-based software development. [Gamma et al., 1995] introduces an adapter pattern in his textbook on design patterns for object-oriented system development. Such adapters enable reusable descriptions of objects and make it possible to combine objects that differ in their syntactical input and output descriptions. [Fensel & Groenboom, 1997] and [Fensel & Groenboom, 1999] introduced the concept of an *adapter* in architectural descriptions of knowledge-based systems to (1) *decouple* different element of this model, to (2) *encapsulate* these different elements and to (3) explicitly model their *interactions*. Work on software architectures describes systems in terms of components and *connectors* that establish the proper relationships between the former (cf. [Garlan & Perry, 1995], [Shaw & Garlan, 1996]). Here, the focus is to mediate between different interaction styles of components (which we call the business logic of a web service).[10] Finally, work on heterogeneous and distributed information systems developed the concepts of wrappers and a mediator. Instead of assuming a global data schema, heterogeneous information and knowledge systems have a *mediator* [Wiederhold, 1992]that translates user queries into sub-queries on the different information sources and integrates the sub-answers.

For an open and flexible environment such as web-based computing, adapters are an essential means to cope with the inherit heterogeneity. This heterogeneity can wear many cloths:

- Mediation of **data structures**. A web service may provide an input for a second one, however, not in the format it is expecting.
- Mediation of **business logics**. Two web services provide complementary functionality and could be linked together in principle (one is a shopping agent and one is a provider of the searched goods), however, their interaction patterns do not fit.

10. (cf. [Yellin & Strom, 1997]).

- Mediation of **message exchange protocols**. SOAP over http is unreliable requiring trading partners have to implement transport level acknowledgments as well as time-out, retry, upper resent limits as well as duplicate detection in order to guarantee exactly once semantics. Web services may differ in the way, they achieve such a reliability layer.
- Mediation of dynamic **service invocation**. A web service may invoke other web services to provide its functionality. This can be done in a hard-wired manner, however, it can also be done more flexible by just referring to certain (sub-)goals. During execution other services can be invoked dynamically.

Mediation of these different aspects can be done with different underlying process models. We will start with discussing these different process models and will then look at each aspect in more detail.

3.4.1 Process models for mediation

Let's assume a service provider *P* that provides service ws_1 and a requester *R* that requests ws_1. In this case there are two approaches.

Client/server approach. In this case *P* provides service ws_1. Whoever wants to request ws_1 has to deal with the data structures (message format), business logic, and message exchange protocol defined by *P*. In this case the mediation takes place within the control of *R* and only by *R*. In this case *P* executes its public process the way it is defined by *P* and R has to "deal" with it, i.e. has to mediate in such a way that its expected behavior is achieved after mediation.

> **Example**. A huge corporation *R* that requests from its suppliers that they comply if they want to do business. In this case the huge corporation is unwilling to change.
>
> **Example**. *P* sends a purchase order in several messages, each line item in a separate message. However, *R* expects a complete purchase order. In this case *R* has to build some mediation that collects all individual line items for that one purchase order and once all line items are received (interesting problem in itself) it can put them into one purchase order. Then *R*'s original expectations are fulfilled and the mediation was successful.

Peer-to-peer approach[11]**.** *P* and *R* agree on two matching public processes. In this case both, *R* and *P* have to mediate to their original public process from the newly agreed one. In this case requester *R* has a choice of approach. If *P* provides data structures etc. that *R* "likes", *R* just takes the data structures and does the mediation. If *R* does not like the data structures, *R* and *P* can agree on a format that is different from what *P* already provides. In this case *P* has to do some of the mediation from its original data structures

11. Our definition of P2P is like this: if a sender wants to send data to a recipient the sender opens a direct connection to the recipient. With direct connection we mean not necessarily on the transport level (i.e. hops are ok), but no third party in between gets to see the data (either as forwarder or as blackboard, etc.). Napster in that sense is not P2P since there is a shared place where the peers go to. For example, in the EDI world, an enterprise could send a purchase order to another enterprise without any interpreting third party in between. The VAN they used was a store and forward device, i.e. not interpreting. Similar with RosettaNet. Both parties open a direct http connection and send each other the RosettaNet messages. No third party in between.

into the newly agreed one. *R* also has to mediate from the newly agreed one to the one *R* really needs. Of course, if the newly agreed upon data structure is exactly what *R* needs, only *P* has to do mediation.

> **Example.** Two trading partner may agree on RosettaNet as their exchange data structure. In this case both trading partners have to transform RosettaNet to whatever internal data structures they have.
>
> **Example**. *P* and R agree on using PIP 3A4 of RosettaNet. 3A4 is the PO - POA exchange. *R* has to internally mediate to its "half" of 3A4 and *P* has to mediate to its half. In the best case there is no real mediation. This is when *P* and/or *R* have their private process match exactly the public process.

The first case requires only one transformation per message exchanged. This transformation has to take place at *R* or an intermediary between *P* and *R*. The latter case might require up to two transformations, one at *P* and one at *R*. However, it might require none if both, *R* and *P* have RosettaNet as their internal data structure too (too good to be true). If there is an intermediary in the latter case it could handle the situation with one transformation. But that would violate the P2P approach.

Mediation enabled Peer-to-Peer approach. Such a P2P scenario as described above may not scale because it would require a trading partner to implement hundreds of formats to be fully P2P enabled. Therefore, this scenario may only work with intermediate mediation service that makes differences in data formats transparent to the agents. Then they can fully enjoy P2P EC. Therefore, such a mediation does not violate but rather enable P2P. This is like the fact that both need an internet provider to be only and can start their P2P relation. Only the service we are taking about is at a higher conceptual level.

Real P2P communication implies that any transformation is done at either end, the sender and/or the receiver. This requires the sender/receiver to maintain 100s of transformations. It is further true that this only scales if the transformations are maintained. Otherwise the parties could not talk. Solving this problem by mediated P2P communication also has its own problems:

- The third party needs to maintain all the required transformations.
- All trading partners of a trading partner need to be connected to the same third party the trading partner choose. If there is one trading partner that chose another third party, there is a connectivity problem (like in the case of VANs): the third parties have to communicate between each other in order to connect the two trading partners.
- In order to do the transformations the third party needs to get clear text access to the messages. This requires an elaborate security schema. Traditional signing and encryption does not work for the end-to-end case any more. Both trading partners have to trust the third party.
- Trading partner specific transformation data like domain value maps and cross-reference tables have to be maintained by the third party. A trading partner must be able to modify those at any time.
- No dynamic transformations requiring access to other data sources of a trading

partner can be present. Otherwise the third party needs to get access to the trading partners data.

- Any trading partner specific modifications of documents cause specific transformations at the third party.

In summary, conceptually the third party approach scales until we get into trading partner specific modifications of documents as well as transformations as well as several connected third parties. In real life this means that if Sun and Cisco want to communicate, they have to go through a third party that is an independent organization (company). That company must have a solid long-term business model since without it communication would break down. In order for Sun or Cisco to trust the third party, it needs to show not only long-term viability, but also trustworthiness as well as responsiveness to companies special requirements.

We left out the possibility that *R* is the "stronger" trading partner able to force *P* to mediate. However, this is a real possibility and we need to provide for that.

3.4.2 Mediation of data structures

Support for integrating various XML-based standards for E-commerce integration is adopted by several B2B integration frameworks, for example, Microsoft® BizTalk™ Server[12]. However, as shown in [Omelayenko & Fensel, 2001(a)] such a flat and direct approach does not scale because of its flatness and directness.

[Omelayenko & Fensel, 2001(a)] illustrates that even simple integration tasks such as the mapping of an address description in XML standards such as cXML and xCBL leads to complex XSL-T rules. Such rules are difficult to program, their reuse is limited, and maintenance effort is high. In consequence, such an integration approach is costly and does not scale. The difficulties of this direct mapping approach is caused by the fact that it interweaves conceptually different aspects of the alignment process.

- **Extracting information from a certain syntax.** The source XML dialect defines a certain syntactical representation of an (address) information. Therefore, an XSL-T rule is concerned with extracting the actual information from a certain syntax.
- **Mapping different conceptual representation of an information.** An XML standard may use different concepts, different structuring of concepts, different value types, and different natural language descriptions to model a certain piece of information. An XSL-T rule may be used to merge different structures at a conceptual level.[13]
- **Representing information in a certain syntax.** The target XML dialect defines a certain syntactical representation of an (address) information. Therefore, an XSL-T rule is concerned with exporting the actual information in a certain syntax.

Therefore, we describe in [Omelayenko & Fensel, 2001(b)] a layered integration architecture that uses an intermediate data model to represent the actual information and three substeps in actually aliening information from different standards. BAsed on this

12. http://www.microsoft.com/biztalk/default.asp
13. Giving a simple example: One standard may take a street address as street name and house number, whereas another standard may distinguish both.

approach we are able to brake up complex XSL-T rules in a concatenation of three simple and reusable rules that can be gained by simply instantiating given rule patterns. This is an essential first step into a scalable integration framework for heterogeneous data formats.

A second step is required to deal with the number of mappings. The number of different XML "standards" is large and still growing. Providing directly mappings for n standards require n^2 mappings. This does not scale when *n* increases. The only viable approach to deal with this problem is to define an intermediate conceptualization from which is mapped in and out (cf. [Omelayenko & Fensel, 2001(b)]). Then *n* mappings are enough to mediate *n* standards. This intermediate conceptualization (i.e., an Ontology) cannot be just a simple document collection like defined by many XML documents. IT requires a well-defined and principled organization of the domain of concern to keep this conceptualization maintainable and easy to align with external document formats.

In general, many concepts and techniques developed in *Intelligent Information Integration*[14] (*III*, cf. [Wache & Fensel, 2000], [Fensel et al., 2001a]) and related areas can significantly help to overcome these problems. However, most of them require adaptation to the specific needs of web service integration.

3.4.3 Mediation of business logics

On the business logic level messages with business content are processed like purchase orders and invoices. It might be that two trading partners have implemented their version of a public process, each for its role (like buyer and seller). However, the two public processes might mismatch.

For example, a buyer decided to send all line items of a purchase order (PO) individually. In addition, the buyer expects a separate purchase order agreement (POA) for each line item. The reason for a buyer to define its public buying process like this might have been that it is simpler for its internal processing to have every line item separately processed and acknowledged.

Independently, a seller might have implemented its public process in such a way that for each purchase order coming in a purchase order acknowledgment is sent back. No individual line items are accepted, only complete purchase orders. The motivation behind this approach might have been that the seller does not want to coordinate different but related message in its environment.

Business logic mediation has to compensate for the mismatches in public processes. Two types of mismatches can be found: *data mismatches* and *process sequencing mismatches*. The different cases for data mismatches are as follows.

- *Data-complete match*. In this case the messages sent by the requester and expected by the provider match each other precisely. The only task the mediation would have to do (if at all) is to transform them into each other. This is the trivial case where the mediator has almost no work to do.
- *Data-complete mismatch*. Data complete mismatch means that mismatches can be resolved through re-factoring the messages. All data are available in the given

14. http://www.tzi.de/grp/i3/, http://www.aifb.uni-karlsruhe.de/WBS/dfe/iii99.html

messages, they just need to be restructured and re-formatted.
The above given example is of this nature. In this example it is possible for the mediation to collect all line items from the buyer and compose a complete purchase order of it. Once the complete purchase order is available, it is given to the seller. Analogous, once the seller publishes the purchase order acknowledgment, it is split up in to individual POAs for the buyer. In this case the messages could be restructured into each other so that both, seller and buyer send and receive the messages as originally defined in their public processes.

- *Data-over-complete mismatch.* A data-over-complete mismatch occurs when a requester sends more data than the provider requires. However, the superfluous data cannot be dropped by the mediation since return messages to the requester might require precisely these data. In this case the mediation must store the superfluous data in order to have it available for the return messages to be completed.
- *Data-incomplete mismatch*. This case happens when a requester sends not enough data to the provider. This means that messages to the provider would miss data and will cause a failure in the provider. The mediation has to get back to the requester to request more data. This is possible if the requester has another public process available for this purpose. Alternatively, the requester can change the public process in order to provide more data. Once the missing data is available the mediator can complete the message and send it to the provider. Conversely, the provider could return either too much or not enough data to the requester. In the former case the mediator has to decide if the superfluous data can be dropped or if it has to be kept around. In the latter case the mediator has to ask the provider for additional (the missing) data so that the requester's need can be satisfied.

Secondly, the business logic mediation has to solve *process sequence mismatches*. Fundamentally, each public process defines in which order messages are send by it and message are required by it. The following cases can be distinguished.

- *Precise match*. Each of the public processes involved sends the messages in exactly the order the other public process requests it. In this case the mediator does not have to compensate for sequence mismatches at all since no mismatch exists.
- *Unresolvable message mismatch.* In case of the unresolvable mismatch a provider expects a message that is not sent by a requester. This means that a necessary message is missing. If the mediator cannot provide any mediation logic generating a meaningful a message an unresolvable mismatch exists.
- *Resolvable message mismatch*. In the above example, the buyer sends all line items individually. After all have been collected, they can be forwarded by the mediation to the provider as one purchase order. This means that the requester sent all the data required by the provider. In this example the match worked out fine.

Another example of a resolvable message mismatch is the following. A requester sends a purchase order and expects back a purchase order acknowledgment. A provider, however, receives a purchase order, but does not send back an explicit purchase order acknowledgment. A mediator can in this case generate a "dummy" purchase order acknowledgment for the requester once the provider accepted the purchase order. The

mediator compensates by generating a message.

- *Unresolvable sequence mismatch*. An example that cannot be mediated is when a requester requests a flight, a hotel and a car (in this order) whereby a provider expects to book a car first, then a hotel and finally the flight. These two sequences are mismatched and the mediation cannot mediate between these two public processes since it cannot change the order of how the requester or the provider expect the messages.
- *Resolvable sequence mismatch*. A resolvable mismatch exists when the provider can take flight, hotel and car booking requests in any order. In this case the mediator forwards the message in the order in which they are sent by the requester.

The mediator has to be able to mediate data mismatches and process sequence mismatches concurrently. Both have to be successful mediated for the overall mediation to succeed.

The location of the mediation can be in three places. Either at one of the trading partners or at an intermediary (see Section 3.4.1).

3.4.4 Mediation of exchange protocols

It is important for us to distinguish the mediation message exchange protocols and the business logic. Web services are based on SOAP over http. There might be encryption on the channel (http) and/or encryption of the SOAP message itself. Signing might happen, too. Since http is synchronous, both, receiver and provider have to agree on how to encrypt and how to sign. However, SOAP over http is unreliable. This means that trading partners have to implement transport level acknowledgments as well as time-out, retry, upper resent limits as well as duplicate detection in order to guarantee exactly once semantics. This are the exchange protocols. We need to mediate the exchange protocol if a receiver expects specific behavior a provider does not provide. For example, a provider *P* might send an explicit failure notification message that indicates that *P* thinks the exchange failed. However, a receiver *R* might not have encountered such a behavior so far and now needs to mediate. That is, *R* needs to deal with the explicit failure notification message. This mediation is different from the earlier discussed level that deals with the “real” business messages like purchase order and invoices. The processing of those is the business logics. The exchange logics is the low level means to enable the business logic.

We already mentioned earlier that the different business data structures (purchase order, purchase order acknowledgement) have to be mediated. This is interwoven with the business logic. However, on the message exchange protocol level mediation of its specific data structures again might be necessary, too (like acknowledgments).

Finally, even though we have a layering of exchange protocol (with its specific messages like acknowledgments and failure notifications) and business logic (with its specific message like purchase orders and purchase order acknowledgements as business data), on the wire all message are alike, i.e. the wire itself is not layered.

3.4.5 Scenarios for Service Composition

Let's assume the following:

1) There is a service provider *P* and a service requester *R*.

2) *P* provides web services ws_1, ws_2 and ws_3.

3) R wants to have a combined logic of ws_1, ws_2 and ws_3 (we use “combined logic” to not use “complex web service” at this point).

There are the following possibilities to define a complex web services (and they are not exclusive at all).

The simple case: Provider-based composition

P itself combines ws_1, ws_2 and ws_3 into a new web service ws_4. ws_4 is therefore a complex service (since composed of other services). However, *R* does not necessarily know this since for *R* ws_4 is available like a simple web service. The composition is invisible to *R*.

There is a problem in case *R* wants to/needs to know the status of ws_4 during execution. E.g. ws_4 breaks. The question is, which of the composed services has been executed. Also, ws_4 requires that all input data are available when ws_4 is started.[15] For those cases where *R* does not need to know the execution status and can supply all input data, this approach is fine.

The intermediate case: Client-based composition

P does not combine ws_1, ws_2 and ws_3 at all. Instead *R* combines those by virtue of calling them in a specific order. In this case *R* builds a new web service ws_4 out of ws_1, ws_2 and ws_3. This means that *P* does not know at all that *R* does that. For *P* it looks like that it's provided services are called independently. *R* has to “magically” find out if there are specific constraints invoking ws_1, ws_2 and ws_3. For example, it might be that ws_1 has to be called first, before ws_2 and ws_3 are called.

A special case is that *R* does not even define ws_4 at all, it just calls ws_1, ws_2 and ws_3 in a specific order.

The advanced case: Declarative specification of web service composition

P does not combine the services ws_1, ws_2 and ws_3. However, it defines the possible invocation sequences of the three services through constraints. Thus, the set of constraints could be viewed as a composed web service ws_4. This means that *R* gets told what valid invocation sequences are. This distinguishes this case from the intermediate case. *R* has to make sure that it complies with the constraints when invoking the services.

Summary

From our viewpoint, all three scenarios are valid possibilities. The most advanced proposal is the last one since this means that *P* gives information about how to use the services and *R* can make sure that the constraints are followed without being forced to

---

15. However if the book is not available but there is a secondhand copy, it asks me if this is alright. Thus the breakpoint is the question. If ws_4 is composed out of $ws_{1\text{-}3}$ it does not need all the data when it is started, only at the time when it is needed. (thus the provider does the same as the client before).

adopt a specific implementation of the invocation sequence (like a ws_4). *R* can model the composition in many ways as long as the constraints are fulfilled. In the following we will sketch all three scenarios which we call simple, intermediate, and advanced complex web services. Blowing up these scenarios defines a natural link to work that is done in the *multi-agent system* area (cf. [Giampapa et al., 2001]).

4 Related Work

There exists many related work, mainly performed in three areas: the area of web services and B2B standards, the workflow area, and the area of software architectures.

4.1 Web Services and B2B Standards

Web services and E-commerce are rapidly growing areas. A detailed comparison with all related initiatives is far beyond our scope. After enumerating the most relevant initiatives we will explain why there is still a need for WSMF. Discussions of some of the approaches can be found in [Dogac & Cingil, 1993], who compare the following frameworks for B2B E-commerce: the eCo framework, RosettaNet, BizTalk, cXML, and MESCHAIN, and [Weikum, 2001] collect a number of papers on infrastructure for advanced E-services. The latter states quite clearly that in addition to extremely high scalability, responsiveness, and availability of the data management engine, e-service platforms need to address interoperability, customizability, messaging, process management, and Web application programming and management issues.

[WSAP, 2000] proposes the creation of a **Web Services Activity** of the W3C whose goal is to ensure development of a Web services architecture based on XML, fitting into the Web architecture. This is still an ongoing initiative closely related to WSDL. As mentioned earlier, **WSDL** ([Christensen et al., 2001]) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information, however, it lacks many important features to make it a full-fledged framework for effective B2B integration.

The Web Services Conversation Language (**WSCL**) provides a way to model the public processes of a service, thus enabling network services to participate in rich interactions ([Banerji et al., 2001], [Beringer et al., 2001]). Together, UDDI, WSDL, and WSCL enable developers to implement web services capable of spontaneously engaging in dynamic and complex inter-enterprise interactions. WSCL has been developed as a complement to WSDL. Whereas the latter specifies how to send messages to a service, it does not state the order in which such messages are allowed to be sent. This issue is addressed in WSCL, which defines legal sequences of document exchange between web services. Similar, the **WS-Inspection** specification provides an XML format for assisting in the inspection of a site for available services and a set of rules for how inspection related information should be made available for consumption (cf. [Ballinger et al., 2001]).

Automation of business processes based on web services requires a notation for the specification of message exchange behavior among participating web services. **XLANG** [Thatte, 2001] is proposed to serve as the basis for automated protocol engines that can track the state of process instances and help enforce protocol correctness in message flows.

The Web Services Flow Language (**WSFL**) [Leymann, 2001] is an XML language for the description of Web Services compositions WSFL considers two types of Web Services compositions: The first type specifies the appropriate usage pattern of a collection of Web Services, in such a way that the resulting composition describes how to achieve a particular business goal, typically, the result is a description of a business

process. The second type specifies the interaction pattern of a collection of Web Services; in this case, the result is a description of the overall partner interactions. WSFL is very close in spirit to WSMF, however lack some of the important modeling features of the latter. Examples are the difference between private and publically visible business logic as well as the use of ontologies to keep descriptions reusable. Still a WSMF conform language could be defined as an extension of WSFL.s

The Business Process Modeling Language (**BPML**) [Arkin, 2001] is a meta-language for the modeling of business processes. BPML provides an abstracted execution model for collaborative and transactional business processes based on the concept of a transactional finite-state machine.

ebXML, sponsored by UN/CEFACT and OASIS, is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet (cf. [Waldt & Drummond]). Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes.

The Process Specification Language (**PSL**) is an interchange format designed to help exchange process information automatically among a wide variety of manufacturing applications such as process modeling, process planning, scheduling, simulation, workflow, project management, and business process re-engineering tools ([Schlenoff et al., 2000]). Tools can interoperate by translating between their native format and PSL. Then, any system is able to automatically exchange process information with any other system via PSL. PSL can be used to define formal semantics for process specification in WSMF.

[Bussler, 2001c] provides an excellent survey on many of the B2B protocol standards and develops a conceptual framework for comparing the various kinds of services they provide and they fail to provide. It also illustrates the need for the WSMF we propose. Notice, that none of the mentioned approaches and the approaches discussed in [Bussler, 2001c] provide de-coupling of business logics, message exchange protocols, and document structures (accompanied by adequate information hiding) and complemented by scalable collaboration as the WFML does.

4.2 Workflow Approaches to E-commerce

Workflow is the computerized automatization of a business process. A business process is a set of linked activities, which collectively realize a business objective. One of the myths of inter-enterprise process execution is that workflow management systems (WFMs) deployed in enterprises can achieve the collaboration between enterprises across networks. The reality is that important modeling primitives are missing in WFMs required to achieve inter-enterprise process collaboration (see [Bussler, 2001a]). Like [Bussler, 2001b] illustrates, naive workflow alignment would lead to an unmanageable number of workflow alternatives.[16] Conventional approaches lack three very important features: flexible de-coupling mechanisms of different work flows; scalable mechanism to interweave different work flows; and the distinction between business logic and

16. Very similar to an naive approach to document alignment without proper abstraction steps and a lacking ontology as described in [Omelayenko & Fensel, 2001(a)].

message exchange protocol. In this Section, we will discuss some proposals to overcome this shortcomings and show how the relate with our proposed solution in the WSMF.

Workflow Management Systems are often used in context in B2B integration as a base technology to implement B2B integration processes. [Bussler, 2001b] defines the notion of *distributed inter-organizational workflows*, private and public processes. Based on this definition, the appropriate use of WFMSs is shown in context of an overall B2B integration solution. The need for workflow class inheritance concepts becomes clear once a large set of similar workflow classes have to be managed across a large company. To foster the development of workflow class inheritance [Bussler, 1999] discusses a set of examples that show the benefit of workflow class inheritance. Dynamic sub workflow binding is discussed to avoid the combinatorial explosion of the number of workflow definitions in specific situations. Important modeling primitives are missing in WFMSs requires to achieve inter-enterprise process collaboration. WFMSs were not designed to deal with executing message protocols across networks. In contrast, B2B protocols address all the required functionality to exchange messages reliably between enterprises across networks and are not concerned about enterprise internal processes. Workflow technology does not have the concepts of dynamic binding of public and private processes. [Bussler, 2001a] introduces an approach to bind public and private processes implemented as B2B protocols and workflow types as well as shows an approach of inter-enterprise collaboration management. All these concerns were applied in the WSMF to provide a proper abstraction mechanism for linking together heterogeneous workflows in a scalable manner. Similar proposals have been made by [van der Aalst & Weske, 2001], who describe the Public-To-Private (P2P) approach to inter-organizational workflows, which is based on a notion of inheritance, and [Chiu et al., 2000] and [Kafeza et al., 2001] introduce a novel concept of workflow *views* as a fundamental support for E-service workflow interoperability and for controlled visibility by external parties.

5 Semantic Web enabled Web Services

The easy information access based on the success of the web has made it increasingly difficult to find, present, and maintain the information required by a wide variety of users. In response to this problem, many new research initiatives and commercial enterprises have been set up to enrich available information with machine-understandable semantics. This **semantic web** ([Berneers-Lee et al., 2001], [Fensel & Musen, 2001], and [Fensel et al., 2002]) will provide intelligent access to heterogeneous, distributed information, enabling software products to mediate between user needs and the information sources available. **Web Services** tackle with an orthogonal limitation of the current web. Currently, the web is mainly a collection of information but does not yet provide support in processing this information, i.e., in using the computer as a computational device. Web services can be accessed and executed via the web. However, all these service descriptions are based on semi-formal natural language descriptions. Therefore, the human programmer need be kept in the loop and scalability as well as economy of web services are limited. Bringing them to their full potential requires their combination with semantic web technology. It will provide mechanization in service identification, configuration, comparison, and combination**. Semantic Web enabled Web Services** have the potential to change our life in a much higher degree as the current web already did (see Figure 1) [Bussler, 2001a] identifies the following elements necessary to enable efficient inter-enterprise execution: Public process description and advertisement; discovery of services; selection of services; composition of services; and delivery, monitoring and contract negotiation.

Without mechanization of these processes, internet-based E-commerce will not be able to provide its full potential in economic extensions of trading relationships. There exists already initial attempts to apply semantic web technology to web services.[17] [Trastour et al., 2001] examine the problem of matchmaking, highlighting the features that a

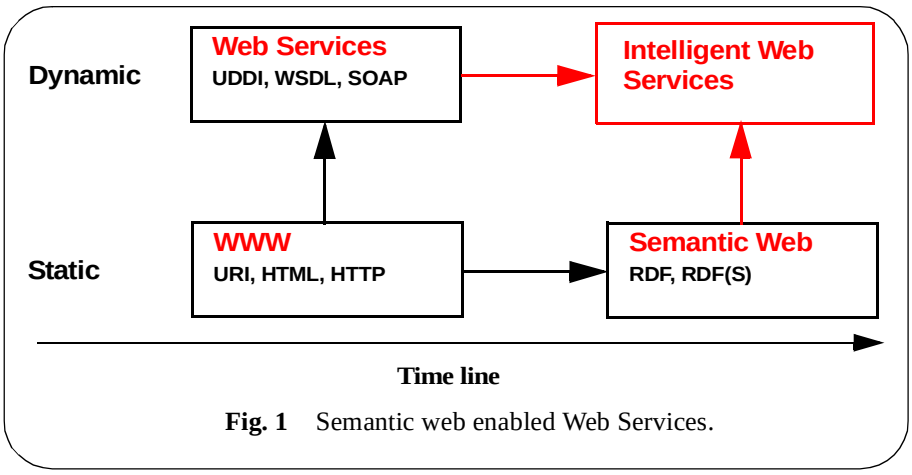

Fig. 1 Semantic web enabled Web Services.

17. [Lemahieu, 2001] provides an excellent introduction in these issues.

matchmaking service should exhibit and deriving requirements on metadata for description of services from a matchmaking point of view. [Hendler, 2001] provides a look at some potential applications of the web semantics and consider some challenges the a research community should be attacking. In particular, he take a look at how information agents and ontologies can together provide breakthrough technologies for web applications. As part of the DARPA Agent Markup Language program, an *ontology* of services has been developed, called **DAML-S** [Ankolenkar et al., 2001], that should make it possible to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties. [Ankolenkar et al., 2001] describe the overall structure of the ontology, the service profile for advertising services, and the process model for the detailed description of the operation of services. The ontology developed in DAML-S is very interesting when trying to define a mark-up language based on WSMF. However, because DAML-S lacks important modeling primitives of WSMF (e.g., it is not aware of the difference between private and public processes, between business logic and message exchange protocols, nor has it any notion of mediation service) this can only be achieved by significantly extending and reorganizing DAML-S.

6 Conclusions

In this paper, we propose a modeling framework called **Web Service Modeling Framework (WSMF)**. Its main elements are: Ontologies, Service descriptions, elementary and complex Web Services. The aim of **WSMF** is to enable fully flexible *and* scalable E-commerce based on web services. We achieve this goal with an architecture that is based on two complementary principles:

- Strong *de-coupling* of the various components that realize an E-commerce application.
- Strong *mediation* service enabling anybody to speak with everybody in a scalable manner.

In the paper we do not define a concrete syntax nor semantics of **WSMF**. This could be achieved in the following way. First, the **WSMF** language could be defined as an extensions of as **WSFL**, which is a language close in spirit to our framework. Also we did not define a concrete web-based syntax for **WSMF**, i.e., we did no define any web-based mark up language. Here one could take **DAML-S** as a starting point and extending it with the necessary modeling features that are missing there. Finally, an approach such as **PSL** could be used to define a formal semantics for the **WSMF**. All - three exercises look rather strait-forward in principle, however, may require serious adaptation work in detail. Finally, [Florescu & Grünhagen, to appear] provide an interesting proposal for a programming language for web services.

[Fensel et al., to appear] defines **UPML**, a conceptual model for describing problem-solving methods. As problem-solving methods and web services share many feature we could reuse many of our experience for the development **WSMF**. **UPML** uses a special class of adapters (so-called *refiners*) to express hierarchal refinement of specification components. For example, refiners are used to refine a generic local search into hill climbing or a configurational design task into a parametric design task (cf. [Fensel, 2000]). Similar refinement concepts need to be developed to structure large volumes of web service descriptions: buying a plane or train ticket are a subclasses of buying a travelling ticket. Refiners help to structure the space of descriptions preventing redundancy and inconsistency. Therefore, we expect future versions of **WSMF** will incorporate a equivalent modeling construct. This concept is also interesting because it provides a link to *invasive programming* [Aksit & Bergmans, 2001] which may help to develop web services based E-commerce solution applying concepts of modern software engineering.

Acknowledgement: The work was partially funded under the European IST project *Ibrow.*

References

[van der Aalst & Weske, 2001]
W.M.P. van der Aalst and M. Weske: In K.R. Dittrich, A. Geppert, and M.C. Norrie (eds.), *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, Lecture Notes in Computer Science LNCS 2068, pages 140-156. Springer-Verlag, Berlin, 2001.

[Allen & Garlan, 1997]
R. Allen and D. Garlan: A Formal Basis for Architectural Connection, *ACM Transactions on Software Engineering and Methodology*, 6(3):213—249, July 1997.

[Ankolenkar et al., 2001]
A. Ankolenkar, M. Burstein, T. Cao Son, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng: DAML-S: Semantic Markup For Web Services, http://www.daml.org/services/daml-s/2001/10/daml-s.html.

[Arkin, 2001]
A. Arkin: Business Process Modeling Language (BPML), Working Draft 0.4, 2001. http://www.bpmi.org/.

[Aksit & Bergmans, 2001]
M. Aksit and L. Bergmans: *Guidelines For Identifying Obstacles When Composing Distributed Systems From Components, Software Architectures and Component Technology: The State of the Art in Research and Practice*, Kluwer, 2001.

[Ballinger et al., 2001]
K. Ballinger, P. Brittenham, A. Malhotra, W. A. Nagy, and S. Pharies: Web Services Inspection Language (WS-Inspection) 1.0. http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html, 2001.

[Banerji et al., 2001]
A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, S. Williams: Web Services Conversation Language (WSCL), HP, 2001.

[Berneers-Lee et al., 2001]
T. Berners-Lee, J. Handler, and O. Lassila: The Semantic Web, *Scientific American*, May 2001.

[Beringer et al., 2001]
D. Beringer, H. Kuno, and M. Lemon: Using WSCL in a UDDI Registry 1.02, UDDI Working Draft Technical Note Document, May 5, 2001. http://www.uddi.org/pubs/wscl_TN_forUDDI_5_16_011.doc.

[Bussler, 1999]
C. Bussler: Workflow Class Inheritance and Dynamic Workflow Class Binding. In *Proceedings of the Workshop Software Architectures for Business Process Management at the 11th Conference on Advanced Information Systems Engineering CAiSE´99*, Heidelberg, Germany, June 1999.

[Bussler, 2001a]
C. Bussler: The Role of B2B Protocols in Inter-enterprise Process Execution. In *Proceedings of Workshop on Technologies for E-Services (TES 2001) (in cooperation with VLDB2001)*. Rome, Italy, September 2001.

[Bussler, 2001b]
C. Bussler: The Role of Workflow Management Systems in B2B Integration. In *Proceedings of the Fourth International Conference on Electronic Commerce Research (ICECR-4)*, Dallas, TX, USA, November 2001.

[Bussler, 2001c]
C. Bussler: B2B Protocol Standards and their Role in Semantic B2B Integration Engines, *IEEE Data Engineering*, 24(1), 2001.

[Casati & Shan, 2001]
F. Casati and M.-C. Shan: Dynamic and Adaptive Composition of E-services, *Information System*, 26, 2001.

[Chiu et al., 2000]
D.K.W. Chiu, K. Karlapalem, and Q. Li. Views for Inter-Organization Workflow in an E-Commerce Environment. In P*roceedings of the 9th IFIP 2.6 Working Conference on Database Semantics (DS-9)*, Hong Kong, April 2001, pp 151-167.

[Christensen et al., 2001]
E. Christensen, F. Curbera, G. Meredith, S. Weerawarana: Web Services Description Language (WSDL) 1.1,15 March 2001. http://www.w3.org/TR/wsdl.

[Clements, 1996]
P. C. Clements: A Survey of Architecture Description Languages. In *Proceedings of the 8th International Workshop on Software Specification and Design*, Dagstuhl, Germany, March 1996.

[Dogac & Cingil, 1993]
A. Dogac and I. Cingil: A Survey and Comparison of Business-to-Business E-Commerce Frameworks, *SIGecom Exchanges, Newsletter of the ACM Special Interest Group on E-commerce*, 2(2), Spring 2001.

[Fensel, 2000]
D. Fensel: *Problem-Solving Methods: Understanding, Development, Description, and Reuse*, Lecture Notes on Artificial Intelligence, no 1791, Springer-Verlag, Berlin, 2000.

[Fensel, 2001]
D. Fensel: *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag, Berlin, 2001.

[Fensel & Benjamins, 1998]
D. Fensel and V. R. Benjamins: The Role of Assumptions in Knowledge Engineering, *International Journal of Intelligent Systems (IJIS)*, 13(8):715-748, 1998.

[Fensel et al., 2001a]
D. Fensel, F. Baader, M.-C. Rousset and H. Wache: Special issue of the Journal *Data and Knowledge Engineering (DKE)* on Intelligent Information Integration, 36(3) 2001.

[Fensel et al., 2002]
D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster (eds.), *Semantic Web Technology*, MIT Press, Boston, to appear 2002.

[Fensel et al., to appear]
D. Fensel, E. Motta, F. van Harmelen, V. R. Benjamins, M. Crubezy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga: The Unified Problem-solving Method Development Language UPML, to appear in *Knowledge and Information Systems (KAIS): An International Journal*.

[Fensel & Groenboom, 1997]
D. Fensel and R. Groenboom: Specifying Knowledge-Based Systems with Reusable Components. In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)*, Madrid, Spain, June 18-20, 1997.

[Fensel & Groenboom, 1999]
D. Fensel and R. Groenboom: A Software Architecture for Knowledge-Based Systems, *The Knowledge Engineering Review (KER)*, 14(3), 1999.

[Fensel & Musen, 2001]

D. Fensel and M. Musen: Special Issue on Semantic Web Technology, *IEEE Intelligent Systems (IEEE IS)*, 16(2), 2001.

[Florescu & Grünhagen, to appear]
D. Florescu and A. Grünhagen: An XML Programming Language for Web Service Specification and Composition, to appear.

[Gamma et al., 1995]
E. Gamma, R. Helm, R. Johnson, and J. Vlissides: *Design Patterns*, Addison-Wesley Pub., 1995.

[Garlan & Perry, 1995]
D. Garlan and D. Perry (eds.): Special Issue on Software Architecture, *IEEE Transactions on Software Engineering*, April 1995.

[Giampapa et al., 2001]
J.A. Giampapa, O. Juarez-Espinosa, and K. Sycara: Configuration Management for Multi-Agent Systems. In *Proceedings of the 5th International Conference on Autonomous Agents (Agents 2001)*, USA, June, 2001.

[Hendler, 2001]
J. Hendler: Agents and the semantic web, *IEEE Intelligent Systems (IEEE IS)*, 16(2), 2001.

[Hofmeister et al., 1999]
C. Hofmeister, R. L. Nord, and D: Soni: Describing Software Architectures with UML. In P. Donohoe (eds.), *Software Architecture*, Kluwer Academic Publ., 1999.

[Kafeza et al., 2001]
E. Kafeza, D.K.W. Chiu, and I. Kafeza: View-based Contracts in an E-service Cross-Organizational Workflow Environment. In *Proceedings of the Workshop on Technologies for E-Services (TES 2001)*, in cooperation with VLDB-2001, Rome, Italy, September 14-15, 2001.

[Lemahieu, 2001]
W. Lemahieu: Web Service description, advertising and discovery: WSDL and Beyond, 2001. In J. Vandenbulcke and M. Snoeck (eds.), *New Directions in Software Engineering*, Leuven University Press, 2001.

[Leymann, 2001]
F. Leymann: Web Service Flow Language (WSFL 1.0), May 2001. http:// www-4.ibm.com/ software/solutions/webservices/pdf/WSFL.pdf.

[Mevidovic & Rosenblum, 1999]
N. Mevidovic and D. S. Rosenblum: Accessing the Suitability of a Standard Design Method for Modeling Software Architectures. In P. Donohoe (eds.), *Software Architecture*, Kluwer Academic Publ., 1999.

[Omelayenko & Fensel, 2001(a)]
B. Omelayenko and D. Fensel: An Analysis of Integration Problems of XML-Based Catalogues for B2B E-commerce. In *Proceedings of the 9th IFIP 2.6 Working Conference on Database (DS-9)*, Semantic Issues in e-commerce Systems, Hong Kong, April 2001.

[Omelayenko & Fensel, 2001(b)]
B. Omelayenko and D. Fensel: A Two-Layered Integration Approach for Product Information in B2B E-commerce. In *Proceedings of the Second International Conference on Electronic Commerce and Web Technologies (EC-WEB 2001)*, Munich, Germany, September 2001.

[Omelayenko & Fensel:, submitted]
B. Omelayenko and D. Fensel: Scalable Document Integration for B2B Electronic Commerce, submitted.

[Penix et al., 1997]
J. Penix, P. Alexander, and K. Havelund: Declarative Specification of Software Architectures. In *Proceedings of the 12th IEEE International Conference on Automated Software*

Engineering (ASEC-97), Incline Village, Nevada, November 3-5, 1997.

[Shaw & Garlan, 1996]
M. Shaw and D. Garlan: S*oftware Architectures. Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[Schlenoff et al., 2000]
C. Schlenoff, M. Gruninger, F. Tissot, J. Valois, J. Lubell, J. Lee: The Process Specification Language (PSL): Overview and Version 1.0 Specification, NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD (2000). http://www.mel.nist.gov/psl/.

[Thatte, 2001]
S. Thatte: XLANG: Web Services for Business Process Design, Microsoft Corporation, 2001. http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

[Trastour et al., 2001]
D. Trastour, C. Bartolini, and J. Gonzalez-Castillo: A Semantic Web Approach to Service Description for Matchmaking of Services. In *Proceedings of the Semantic Web Working Symposium*, Stanford, CA, USA, July 30 - August 1, 2001.

[Wache & Fensel, 2000]
H. Wache and D. Fensel: Special issue of the *International Journal of Cooperative Information Systems* on Intelligent Information Integration, 9(4), 2000.

[Waldt & Drummond]
D. Waldt and R. Drummond: EBXML: The Global Standard for Electronic Business, http://www.ebxml.org/presentations/global_standard.htm.

[Weikum, 2001]
G. Weikum (ed.), Special Issue on Infrastructure for Advanced E-services, *IEEE Data Engineering*, 24(1), 2001.

[Wiederhold, 1992]
G Wiederhold: Mediators in the Architecture of Future Information Systems, *IEEE Computer*, 25(3):38—49, 1992.

[WSAP, 2000]
Web Service Activity Proposal, 2000. http://www.w3.org/2001/10/ws-activity.html.

[Yellin & Strom, 1997]
D. M. Yellin and R. E. Strom: Protocol Specifications and Component Adapters, ACM Transactions on Programming Languages and Systems, 19(2):292—333, 1997.

[Zaremski & Wing, 1997]
A. M. Zaremski and J. M. Wing: Specification Matching of Software Components, ACM Transactions on Software Engineering and Methodology, 6(4):335—369, 1997.