ELSEVIER

Electronic Commerce Research and Applications 1 (2002) 113–137

Electronic Commerce Research and Applications

www.elsevier.com/locate/ecra

The Web Service Modeling Framework WSMF

Dieter Fensel[a,*], Christoph Bussler[b,1]

[a]*Division of Mathematics and Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam (VU), De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands*

[b]*Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, USA*

Abstract

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

Keywords: Web service modeling; Mediation; Semantic integration; Complex web services; Web service ontology

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[2], WSDL[3], and SOAP[4] 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 pro-

*Corresponding author. Tel.: +31-6-5185-0619; fax: +31-84-872-2722.

E-mail addresses: dieter@cs.vu.nl (D. Fensel), chris.bussler@oracle.com (C. Bussler).

[1]Tel.: +1-650-607-5684.

[2]**UDDI** provides a mechanism for clients to find web services. Using a UDDI interface, businesses can dynamically look up as well as discover services provided by external business partners (http://www.uddi.org/).

[3]**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/).

[4]**SOAP** is a message layout specification that defines a uniform way of passing XML-encoded data (http://www.soap.org/).

1567-4223/02/$ – see front matter
PII: S1567-4223(02)00015-7

viding a way of automated program communication, discovery of services, etc. Therefore, they are the focus of much interest from various software development companies.[5]

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 have a cost attached, the program knows when to search for a cheaper service and knows all the 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 automatic cooperation between enterprises. Any 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 would be supply chain relationships where an enterprise manufacturing short-lived goods has to frequently seek suppliers as well as buyers dynamically. Instead of employees constantly searching for suppliers and buyers, the web service infrastructure does it automatically within the defined constraints.

Still, more work needs to be done before the web service infrastructure can make this vision come true. Current technology around UDDI, WSDL, and SOAP provides limited support in mechanizing service recognition, service configuration and combination (i.e. realizing complex workflows and business logics with web services), service comparison and automated negotiation. Therefore, there are proposals such as **WSFL** [1] that develops a language for describing complex web services or **DAML-S** [2] that employs *semantic web* technology [3–5] for service description. The **Web Service Modeling Framework (WSMF)** follows this line of research. It is a fully-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 are 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 introduce 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 puts 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 with which web services are realized and indicate the future steps to take 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

[5] See http://www.w3.org/2001/01/WSWS/.

technical and business terms.[6] 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 with each other to carry out various commercial activities and to provide value-added services. The barriers to 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 comprehensible to the human reader), and HTTP defines a protocol for retrieving 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 look up 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 its 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. 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. Ref. [6]).

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 defines a service.

SOAP is a message layout specification that defines a uniform way of passing XML-encoded data. It 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 that

[6] See http://www.diffuse.org/WebServices.html.

allows for 'RPC *over the web*' providing a very simple one-way as well as request/reply mechanism.

2.3. Functionalities required for successful web services

UDDI, WSDL, and SOAP are important steps in 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 eventually achieve the above vision. Bussler [7] identifies the following elements as being necessary to achieve scalable web service discovery, selection, mediation and composition:

- **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 is structured according to the document types defined.
- **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 units of measure 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.
- **Transport binding**. Several transport mechanisms are available like HTTP/S, S/MIME, FTP or EDIINT. A service requester as well as a service provider has to agree on the transport mechanism to be used when service requests are executed. For each available transport mechanism the layout of the message must be agreed upon and how the document sent shall be 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, a requirement for SOAP message processing.
- **Exchange sequence definition**. Communication over networks is 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 timeouts, retry logic and upper retry limits.
- **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 goods are ordered by a purchase order and confirmed by a purchase order acknowledgment they have to be shipped and also paid for.
- **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, ensures unmodified privacy whereby non-repudiation services ensure that neither service requester nor service provider can claim not to have sent a message or to have sent a different one.
- **Syntax**. Documents can be represented in dif-

ferent syntaxes available. XML is a popular syntax, although non-XML syntax is used also (e.g. EDI[7]).

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

Current web service technology scores rather low compared to these requirements. Actually, SOAP provides support on information binding. Neither UDDI nor WSDL add any support in the terms enumerated above. Many organizations have 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 [1], XLANG [8], ebXML BPSS [9], BPML [10] and WSCL [11]. Still, there are important features missing in all of the mentioned frameworks. It is very important 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 publicly visible workflows (public processes) and internal business logics of a complex web service (private processes). Therefore, we developed a fully-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 permits the mechanization of these tasks. We will further discuss this aspect in Section 3.
- Mechanized support is needed in dealing with numerous and heterogeneous data formats. Various ‘standards’ exist for how to describe products and services, product catalogues, and business documents. Ontology technology (cf. Ref. [12]) is required to better define such standards and to map between them. Efficient bridges between different terminologies are essential for openness and scalability. We will further discuss this aspect in Section 3.
- 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 for these differences, allowing partners to cooperate properly (cf. Ref. [13]). We will further discuss this aspect in Section 3.

These 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 between internal business intelligence and public message exchange protocol interface descriptions [13]. Coupling of processes can only be achieved via interfaces to keep the amount of interactions scalable.

[7] See http://www.x12.org/.

- Strong *mediation* service enabling anybody to speak with everybody in a scalable manner. This mediation service includes the mediation of different terminologies [12] as well as the mediation of different interaction styles [13].

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 a mediation service. None of the other approaches we found in the literature take our point to such an extent. 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 different main 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. Ref. [12]) are a 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 reuse. Since the early 1990s, 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 make 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*[8] specifies objectives that a client may have when he consults a web service. A goal specification consists of two elements:

- **Pre-conditions** describe what a web service expects 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 already explained what I understand by 'best'.
- **Post-conditions** describe what a web service returns in response to its input. In response to 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 descriptions because there is an *n*2*m* mapping between them, i.e. the same web

[8] See Ref. [14].

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. Conversely, 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 that could or could not be enforced for such a relationship. Such properties determine heavily whether there is one, many or no solution for a goal. Ontologies provide reusable vocabulary with precisely defined properties. Software that handles one, many, or potentially 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 the transitivity of a nearer relationship may fall into endless sleep[9] 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.''*
>
> (Dieter Fensel)

Many web service description languages distinguish between elementary and complex web services. Elementary web services are simple input/output boxes, whereas complex web services break down the overall process into sub-tasks that may call other web services. Strictly speaking, such a distinction is wrong and may lead to mis-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 dynamic) that makes a difference. A complex web service such as a logical inference engine with a web interface can be *described* as rather elementary. It receives some input formulas and derives—after a while—a set of conclusions. A much simpler software product such as a simple travelling information system may be broken down into several web services around hotel information, flight information, 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, it has some important consequences:

(1) Many web service description approaches do not make an explicit distinction between an internal description of a web service and its external visible description. They 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 descriptions of the realization of a web service. In our framework we strictly limit ourselves to 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 rather 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 it not to make it publicly 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 [13].

(2) The dichotomy of elementary and complex web

[9] That is, it stays in an endless loop.

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 service. In any case we describe them as black boxes, i.e. we do not want to model the internal aspects of how such a service is achieved.

3.3.1. Black box descriptions

First, a web service has a **name**, i.e. a unique identifier to refer to it. 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 to be able to successfully execute it. 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 the input and output of a web service. Second, a web service can *strengthen a pre-condition* or *weaken a post-condition* of a goal.[10] Our goal may be to find the nearest restaurant to our current location. A restricted service to achieve this goal may ask for our location according to a certain standard for representing geographical information and may only provide the restaurants that are registered on 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 do not require explicit mediation because in these cases, each result of the web service fulfills the goal descriptions.

Fourth, a web service description describes the structure of its **input data** and **output data**. The elements 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 with 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 with the complex service execution. This permits, for example, making 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 (cf. Ref. [16]).

3.3.2. Gray box descriptions

These description elements still consider 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

[10] Cf. Ref. [15].

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 can not be known, then the requester would have no way of 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 did 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 before it finishes. A requester might choose to execute some local logic concurrently with 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 h 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 about 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 from an independent travel service. 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 of 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 consist simply 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 referral to a service without knowing at definition 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 look-up in, for example, 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 an invoked web service proxy's input port is a **data flow**. Data flows 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 become the 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 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', e.g. from one 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 replicas 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 typecast is necessary to make sure that the data passed between ports match according to their type. A data flow can contain a typecast 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 an invoked web service proxy. However, its implementation does not invoke a web service at runtime. Instead, execution logic is executed that is implemented, for example 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 implementation of 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, so 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, not 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 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 part of the business logic of a web service. With both sets 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 an 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 to 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 fails, the invoking web service is in an 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 has to find out how to find an alternative hotel. If this does not succeed, the requester has 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 permits definition of what happens after an invoked web service finally fails. 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 fails, 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 can 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 whether the flight should be kept or canceled.

3.3.3. Communication-oriented description elements

In any communication people have to exchange signals at three different levels:

- One has to indicate that one hears a message.[11]
- One has to indicate that one understand a message.[12]
- One has to indicate agreement about the content of a message.[13]

Chappell et al. [17] distinguish two kinds of message confirmation: ''At the message service level, guaranteed delivery means the recipient messaging service has received the message'', but at the business level, this is not enough: the receiving party is required to guarantee that the message has been archived and has passed some basic integrity checks to ensure that no matter what happens, it will be able to process them. ebXML introduces business signals to express the second type of acknowledgement. The WSMF reflects these different levels of acknowledgement. At the business layer an acknowledgement reflects a possible legally binding step that agrees on a certain content of a message. We already discussed this layer during the last section. It remains to define the **message understanding layer** and the **message exchange layer**.

Eleventh, web services need descriptions related to the acknowledgement of **message understanding**. An acknowledgement at this layer indicates that a receiver was able to correctly parse a message and achieved it properly. It indicates that a receiver has been able to process and understand the message but it does not already imply that he agrees with its content, i.e. it does not imply any legal binding which is attached to the business layer as discussed earlier.

Twelfth, web services need descriptions 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.

[11] If not, the sender should speak harder.
[12] If not, the sender should try a different language.
[13] If not, the sender has to make a different proposal or needs to look for a different partner.

Reliable networks guarantee that a sent message will be delivered. Unreliable networks do not guarantee this. Web Services assume that the transmission of messages takes place only once. In the case of reliable networks, transmitting a message only once 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 re-send of the message happens and the time-out starts again. In case the acknowledgment is again not received in the given time-out a re-send of the message happens. In order to not get into an endless loop, an upper limit restricts the number of re-sends. 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 semantic to web services. It deals with the different types of networks and their properties and provides a nice abstraction.

Example. One popular example of a message transport is SOAP. SOAP is an XML-encoding of data types in addition to a message layout definition (header and body). SOAP does specify the data type encoding in XML as well as the message layout, however, it does not provide any additional features that are required when exchanging messages. For example, in order to guarantee the asynchronous delivery of SOAP messages over the Internet using SMTP, it is necessary to build a higher level protocol around SOAP. As mentioned in the previous paragraph, acknowledgments, retry-semantics and upper limits have to be added to SOAP in order to achieve reliable message transmission. This additional functionality is not part of SOAP itself.

3.3.4. Security and trust

Thirteenth, security and trust are two fundamental properties of web services. Security is concerned about securing the message exchange as well as non-repudiation. Securing the message exchange requires that:

- **Messages are not changed**. While a message is in transit on a network between two endpoints, any attempt to change the message content needs to be detected. This is achieved by message signatures. The sender signs the message. While the message is still in clear text, a signature is added to the message that contains a unique number that is computed from the message content in encrypted form. The receiver of the message also receives the encrypted signature. It can compute the same number from the message upon receipt. Once this is accomplished, the receiver can decrypt the message, compare the unique numbers and only when the numbers are equal was the message as not modified.
- **Messages are not intercepted**. To prevent any listener on the network from interpreting a message in transit messages are encrypted. Private key infrastructures (PKI) are used to manage the public and private keys of endpoints. A sender encrypts a message with the receiver's public key. This means that only the sender can decrypt the message with its private key, and no other entity can decrypt the message for viewing.
- **Messages are from the sender**. Message encryption in the point above was used to ensure message privacy. However, this does not guarantee that the messages comes from the sender as claimed in the message header since the receiver does not have any token that uniquely identifies the sender. If the sender in addition encrypts the message with its private key then the receiver is able to decrypt the message with the sender's public key. This then allows the receiver to verify the sender.
- **Messages are not lost**. Messages must not be lost without detection. Sequence numbers are a means for sender and receiver to make sure that each message that was sent is received. A missing sequence number indicates a lost mes-

sage. Both sender and receiver can react to this by re-sending previously lost messages.
- **Messages are not artificially introduced**. Sequence numbers also ensure that messages are not artificially introduced. Any message is checked for being a duplicate and if a message is a duplicate the message is discarded.

Non-repudiation is a security mechanism for an endpoint to prove that it received or sent a message from/to another endpoint. Messages are stored as received (fully encrypted) and this enables a receiver to prove at any point in time that the message was received from indeed the sender who is identified within the message. The fact that a message was received by a receiver can be non-repudiated at a sender's site by storing the receipt acknowledgment.

Trust is a entirely different aspect. Even if messages are secured and non-repudiated, this does not imply that endpoints trust each other. A supplier might promise to deliver goods at a specific time and in a specific quality, but it is not guaranteed that this happens every single time. Endpoints therefore build up a knowledge base about their trading partners that indicates the level of trust based on past experience.

A different measure of establishing trust is to have steep fines for late or flawed delivery of services or goods. This must be part of a contract that is enforceable by law. Steep fines are incentive to comply with promised interactions.

3.3.5. Further aspects

Fourteenth, there are important **non-functional properties** that characterize 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/maximum 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. [18] 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 and Groenboom [19,20] introduced the concept of an *adapter* in architectural descriptions of knowledge-based systems to: (1) *decouple* different elements of this model; (2) *encapsulate* these different elements; and (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. Refs. [21,22]). Here, the focus is to mediate between different interaction styles of components (which we call the business logic of a web service).[14] 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* [24] 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 inherent heterogeneity. This heterogeneity can wear many clothes:

- Mediation of **data structures**. A web service may provide an input for a second service, 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.
- Mediation of **message exchange protocols**. SOAP over http is unreliable requiring trading partners to implement transport level acknowledgments as well as time-out, retry, upper resend 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

[14] Cf. Ref. [23].

done more flexibly 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 by discussing these different process models and will then look at each aspect in more detail.

3.4.1. Process models for mediation

Let us 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 is 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 is successful.

Peer-to-peer approach[15]. *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 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 an 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 enables P2P. This is like the fact that both only need an Internet provider to be able to start their P2P relation. Only the service we are talking about is at a higher conceptual level.

Real P2P communication implies that any transformation is done at either end, the sender and/or the

[15]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. Similarly with RosettaNet. Both parties open a direct http connection and send each other the RosettaNet messages. There is no third party in-between.

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 the trading partners of a trading partner need to be connected to the same third party the trading partner chooses. If one trading partner chooses 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 make 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-referenced tables have to be maintained by the third party. A trading partner must be able to modify these at any time.
- No dynamic transformations requiring access to other data sources of a trading partner may be present. Otherwise the third party needs to get access to the trading partner's data.
- Any trading partner-specific modifications of documents cause specific transformations for 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, and 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[16]. However, as shown in Ref. [25] such a flat and direct approach does not scale because of its flatness and directness. Omelayenko and Fensel [25] illustrate 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 are 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 a piece of information (address). Therefore, an XSL-T rule is concerned with extracting the actual information from a certain syntax.
- **Mapping different conceptual representation of 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.[17]
- **Representing information in a certain syntax.** The target XML dialect defines a certain syntactical representation of a piece of information (address). Therefore, an XSL-T rule is

[16] See http://www.microsoft.com/biztalk/default.asp.

[17] Giving a simple example: one standard may take a street address as street name and house number, whereas another standard may distinguish both.

concerned with exporting the actual information in a certain syntax.

Therefore, we describe Ref. [26] in a layered integration architecture that uses an intermediate data model to represent the actual information and three sub-steps in actually aligning information from different standards. Based on this approach, we are able to break up complex XSL-T rules into 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. Directly providing mappings for n standards requires n^2 mappings. This does not scale when n increases. The only viable approach to dealing with this problem is to define an intermediary conceptualization to which and from which every standard is mapped (cf. Ref. [26]). Then n mappings are enough to mediate n standards. This intermediate conceptualization (i.e. an Ontology) cannot be just a simple document collection as 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*[18] (*III*; cf. Refs. [27,28],) 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 and message understanding 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 decides 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 messages 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 messages, they just need to be restructured and re-formatted. The example given above is a case 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 from it. Once the complete purchase order is available, it is given to the seller. Analogously, once the seller publishes the purchase order acknowledgment, it is split up in to individual purchase order acknowledgements 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

[18] See http://www.tzi.de/grp/i3/, and http://www.aifb.uni-karlsruhe.de/WBS/dfe/iii99.html.

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 does not send enough data to the provider. This means that messages to the provider would be missing 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 are available the mediator can complete the message and send it to the provider. Conversely, the provider could return either too much or too few data to the requester. In the former case the mediator has to decide if the superfluous data can be dropped or if they have 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 sent by it and messages 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 the case of an 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 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 accepts 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 a 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 messages 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 successfully 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). Fig. 1 gives an example for such a mediation service where a mediator generates proxy acknowledgements to enable communication between two trading partners that follow different protocols for their business interaction. Fundamentally, a received message is passed to the back end application system (e.g. an enterprise resource planning system, ERP).

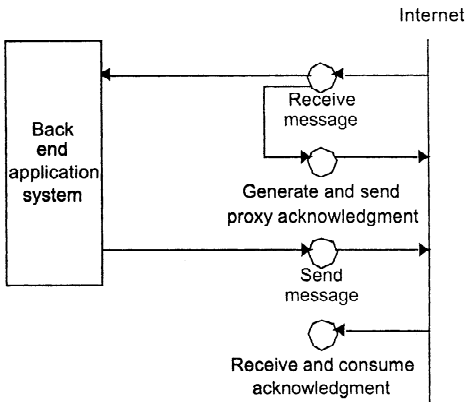

Fig. 1. Example of process mediation.

The trading partner expects an acknowledgment, however, the ERP does not generate one. Therefore, a proxy acknowledgment is generated based on the incoming message. In response to the first message, the ERP generates a response message that is sent out to the trading partner. The trading partner sends back an acknowledgment that is not further used by the ERP. Therefore it is consumed.

The four steps represent the process mediation that mediates the message sequence of the ERP with the one of the trading partners.

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 also happen. 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 re-send limits as well as duplicate detection in order to guarantee exactly once semantics. These 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 before 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 'real' business messages like purchase orders and invoices. The processing of these is the business logic. The exchange logic 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 also be necessary (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 messages like purchase orders and purchase order acknowledgements as business data), on the wire all messages 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 complex web services (and they are not exclusive at all).

3.4.5.1. 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 as a simple web service. The composition is invisible to *R*.

There is a problem in case *R* wants or 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.[19] For such cases where *R* does not need to know the execution status and can supply all input data, this approach is fine.

3.4.5.2. The intermediate case: client-based composition

P does not combine ws_1, ws_2 and ws_3 at all. Instead *R* combines them 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 its 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.

3.4.5.3. 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 the 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.

3.4.5.4. 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 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. Ref. [29]).

[19] 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_{1-3} 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).

4. Related work

There exists much related work, mainly performed in the area of web services and B2B standards and the workflow area.

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 and Cingil [30], who compare the following frameworks for B2B e-commerce: the eCo framework, RosettaNet, BizTalk, cXML, and MESCHAIN, while Weikum [31] collects 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. The Web Service Activity Proposal [32] suggests 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** [11] 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 fully-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 [33,34]. 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. Similarly, 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. Ref. [35]).

Automation of business processes based on web services requires a notation for the specification of message exchange behavior among participating web services. **XLANG** [8] 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**) [1] 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 it lacks some of the important modeling features of the latter. Examples are the difference between private and publicly visible business logic as well as the use of ontologies to keep descriptions reusable. Still, a WSMF-conforming language could be defined as an extension of WSFL.

The Business Process Modeling Language (**BPML**) [10] 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. Ref. [9]). Using ebXML, companies now have a standard method of exchanging business messages, conducting trading relationships, communicating data in common terms and defining and registering 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 [36]. Tools can interoperate by translating between their native format and PSL. Thus, 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 [7] provides an excellent survey of many of the B2B protocol standards and develops a conceptual framework for comparing the various kinds of services they provide and 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 Ref. [7] provide de-coupling of business logics, message exchange protocols, or document structures (accompanied by adequate information hiding) complemented by scalable collaboration as WFML does.

4.2. Workflow approaches to e-commerce

Workflow is the computerized automation 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 collaboration between enterprises across networks. The reality is that important modeling primitives are missing in WFMs required to achieve inter-enterprise process collaboration [13]. As Bussler [37] illustrates, naive workflow alignment would lead to an unmanageable number of workflow alternatives.[20] Conventional approaches lack three very important features: flexible de-coupling mechanisms for different work flows; scalable mechanisms to interweave different work flows; and the distinction between business logic and message exchange protocol. In this section, we will discuss some proposals to overcome these shortcomings and show how they relate to our proposed solution in the WSMF.

Workflow Management Systems are often used in the context of B2B integration as a base technology to implement B2B integration processes. Bussler [37] defines the notion of *distributed inter-organizational workflows* as private and public processes. Based on this definition, the appropriate use of WFMSs is shown in the 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 [38] discusses a set of examples that show the benefit of workflow class inheritance. Dynamic sub-workflow binding is discussed to avoid the combinatorial explosion in the number of workflow definitions in specific situations. Important modeling primitives which are missing in WFMSs require the achievement of an 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 [13] introduces an approach to bind public and private processes implemented as B2B protocols and workflow types as well as showing an approach to 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 and Weske [39], who describe the Public-To-Private (P2P) approach to inter-organizational workflows, which is based on a notion of inheritance, while Chiu et al. [40] and Kafeza et al. [41] introduce a novel concept of workflow *views* as a fundamental support for e-service workflow interoperability and for controlled visibility by external parties. Georgakopoulos et al. [42] describe a workflow-based framework for e-service integration. It is based on the extension of workflow management systems with the notion of service activities (workflow steps). A workflow step (service activity) can invoke a service in order to obtain a specific result as computed by the service. The notion of a service wrapper process is introduced in order to provide abstraction for different services that implement the same goal (similar to the concept of mediation in WSMF). However, the approach follows the invocation paradigm in terms of a remote RPC. The important concept of P2P interaction is not accounted for at all. A workflow is therefore in control of the service invocation without the service being able to asynchronously send back messages to the workflow. The distinction of public and private processes is not followed making mediation impossible to define and to execute. Chen and Hsu [43] and Chen et al. [44] introduce inter-enterprise collaborative business process management. The basic idea is to design a 'virtual' cooperative process across enterprises. Once the design is accomplished, the workflow is 'split' into individual pieces that are executed by the local process managers of the enterprises involved. Each enterprise executes its portion of the 'virtual' cooperative process. Through this approach, all pieces are executed autonomously in the enterprises and true P2P behavior is achieved. However, the approach does not distinguish between private and public processes and a binding between those. Instead, each enterprise can add sub-workflows to its local piece of the overall 'virtual' cooperative process. Furthermore, mediation is not

[20] Very similar to a naive approach to document alignment without proper abstraction steps and a lacking ontology as described in Ref. [25].

considered an issue at all. Neither data nor process mediation are recognized as important issues.

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** [3–5] will provide intelligent access to heterogeneous, distributed information, enabling software products to mediate between user needs and the information sources available. **Web Services** deal with the 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 needs be kept in the loop and the 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 to a much higher degree than the current web already has done (Fig. 2). Bussler [13] 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.

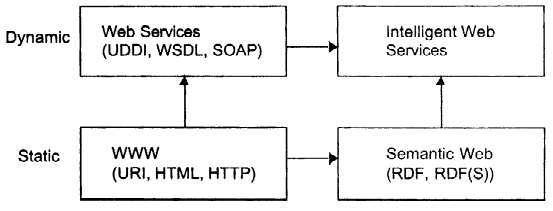

Fig. 2. Semantic web enabled web services.

Without mechanization of these processes, Internet-based e-commerce will not be able to provide its full potential in economic extensions of trading relationships. Initial attempts have already been made to apply semantic web technology to web services.[21] Trastour et al. [45] examine the problem of matchmaking, highlighting the features that a matchmaking service should exhibit and deriving the requirements on metadata for description of services from a matchmaking point of view. Hendler [46] provides a look at some potential applications of web semantics and considers some challenges the research community should be attacking. In particular, he takes 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** [2], that should make it possible to discover, invoke, compose, and monitor Web resources offering particular services and having particular properties. Ankolenkar et al. [2] 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 (i.e. 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 a 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, goal descriptions, elementary and complex web services,

[21] Lemahieu [6] provides an excellent introduction to these issues.

and mediators. 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 or the semantics for **WSMF**. This could be achieved in the following way. First, the **WSMF** language could be defined as an extension of **WSFL**, which is a language close in spirit to our framework. Also, we do not define a concrete web-based syntax for **WSMF**, i.e. we do not define any web-based mark up language. Here one could take **DAML-S** as a starting point and extend it with the necessary modeling features that are missing there. Finally, an approach such as **PSL** could be used to define formal semantics for the **WSMF**. All three exercises look rather straightforward in principle, but may require serious adaptation in detail. Finally, Florescu and Grünhagen [16] provide an interesting proposal for a programming language for web services. Fensel et al. [14] define **UPML**, a conceptual model for describing problem-solving methods. As problem-solving methods and web services share many features, we could reuse much of our experience for the development of **WSMF**. **UPML** uses a special class of adapters (so-called *refiners*) to express the hierarchical 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. Ref. [47]). Similar refinement concepts need to be developed to structure large volumes of web service descriptions: buying a plane or a train ticket are sub-classes of buying a travelling ticket. Refiners help to structure the space of descriptions preventing redundancy and inconsistency. Therefore, we expect that future versions of **WSMF** will incorporate an equivalent modeling construct. This concept is also interesting because it provides a link to *invasive programming* [48] which may help to develop web services based e-commerce solutions applying concepts of modern software engineering.

Acknowledgements

This work was partially funded under the European IST project *Ibrow*, www.ibrow.org.

References

[1] F. Leymann, Web Service Flow Language (WSFL 1.0), 2001, http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.
[2] 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, H. Zeng, DAML-S: Semantic Markup For Web Services, http://www.daml.org/services/daml-s/2001/10/daml-s.html.
[3] T. Berners-Lee, J. Handler, O. Lassila, The semantic web, Scientific American, May (2001).
[4] D. Fensel, J. Hendler, H. Lieberman, W. Wahlster (Eds.), Spinning the Semantic Web: Bringing the World Wide Web to its Full Potential, MIT Press, Boston, 2002.
[5] D. Fensel, M. Musen, Special Issue on Semantic Web Technology, IEEE Intelligent Systems (IEEE IS) 16 (2) (2001).
[6] W. Lemahieu, Web service description, advertising and discovery: WSDL and beyond, in: J. Vandenbulcke, M. Snoeck (Eds.), New Directions in Software Engineering, Leuven University Press, 2001.
[7] C. Bussler, B2B protocol standards and their role in semantic B2B integration engines, IEEE Data Engineering 24 (1) (2001) 3–11.
[8] S. Thatte, XLANG: Web Services For Business Process Design, Microsoft Corporation, 2001, (http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm).
[9] D. Waldt, R. Drummond, ebXML: The Global Standard for Electronic Business, (http://www.ebxml.org/presentations/global_standard.htm).
[10] A. Arkin, Business Process Modeling Language (BPML), Working Draft 0.4, 2001, http://www.bpmi.org/.
[11] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) 1.1, 2001, http://www.w3.org/TR/wsdl.
[12] D. Fensel, Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer, Berlin, 2001.
[13] 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, 2001.

[14] 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, B. Wielinga, The Unified Problem-Solving Method Development Language UPML, Knowledge and Information Systems (KAIS): An International Journal 5(1) (2003), to be published.
[15] D. Fensel, V.R. Benjamins, The role of assumptions in knowledge engineering, International Journal of Intelligent Systems (IJIS) 13 (8) (1998) 715–748.
[16] D. Florescu, A. Grünhagen, D. Kossmann, XL: An XML Programming Language for Web Service Specification and Composition WWW 2002, Int. World Wide Web Conference, Honolulu, HI, 7–11 May 2002.
[17] D.A. Chappell, V. Chopra, J.-J. Dubray, C. Evans, B. Harvey, T. McGrath, D. Nickull, M. Noordzij, P. Peat, P. van der Eijk, J. Vegt, Professional EbXML Foundations, Wrox Press, Birmingham, 2001.
[18] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley, 1995.
[19] D. Fensel, R. Groenboom, Specifying knowledge-based systems with reusable components, in: Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering (SEKE-97), Madrid, Spain, 1997.
[20] D. Fensel, R. Groenboom, A software architecture for knowledge-based systems, The Knowledge Engineering Review (KER), 14(3) (1999).
[21] D. Garlan, D. Perry (Eds.), Special Issue on Software Architecture, IEEE Transactions on Software Engineering 21(4) (1995).
[22] M. Shaw, D. Garlan, Software Architectures. Perspectives on an Emerging Discipline, Prentice-Hall, 1996.
[23] D.M. Yellin, R.E. Strom, Protocol specifications and component adapters, ACM Transactions on Programming Languages and Systems 19 (2) (1997) 292–333.
[24] G. Wiederhold, Mediators in the architecture of future information systems, IEEE Computer 25 (3) (1992) 38–49.
[25] B. Omelayenko, 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, 2001.
[26] B. Omelayenko, 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, 2001.
[27] D. Fensel, F. Baader, M.-C. Rousset, H. Wache, Special Issue on Intelligent Information Integration, Data and Knowledge Engineering 36 (3) (2001).
[28] H. Wache, D. Fensel, Special Issue on Intelligent Information Integration, International Journal of Cooperative Information Systems 9 (4) (2000).
[29] J.A. Giampapa, O. Juarez-Espinosa, K. Sycara, Configuration management for multi-agent systems, in: Proceedings of the 5th International Conference on Autonomous Agents (Agents 2001), USA, 2001.
[30] A. Dogac, 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) (2001) 16–27.
[31] G. Weikum, Ed., Special Issue on Infrastructure for Advanced E-services, IEEE Data Engineering 24(1) (2001).
[32] Web Service Activity Proposal, 2000, http://www.w3.org/2001/10/ws-activity.html.
[33] 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.
[34] D. Beringer, H. Kuno, M. Lemon, Using WSCL in a UDDI Registry 1.02, UDDI Working Draft Technical Note Document, 2001, http://www.uddi.org/pubs/wscl_TN_forUDDI_5_16_011.doc.
[35] K. Ballinger, P. Brittenham, A. Malhotra, W.A. Nagy, S. Pharies, Web Services Inspection Language (WS-Inspection) 1.0, http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html, 2001.
[36] C. Schlenoff, M. Gruninger, F. Tissot, J. Valois, J. Lubell, J. Lee, The Process Specification Language (PSL): Overview and Version 1.0 Specification, National Institute of Standards and Technology, Gaithersburg, MD, 2000, (NISTIR 6459; http://www.mel.nist.gov/psl).
[37] 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, 2001.
[38] 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, 1999.
[39] W.M.P. van der Aalst, M. Weske, The P2P approach to interorganizational workflows, in: K.R. Dittrich, A. Geppert, M.C. Norrie (Eds.), Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), Lecture Notes in Computer Science LNCS 2068, Springer, Berlin, 2001, pp. 140–156.
[40] D.K.W. Chiu, K. Karlapalem, Q. Li, Views for inter-organization workflow in an e-commerce environment, in: Proceedings of the 9th IFIP 2.6 Working Conference on Database Semantics (DS-9), Hong Kong, 2001, pp. 151–167.
[41] E. Kafeza, D.K.W. Chiu, 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, 2001.
[42] D. Georgakopoulos, A. Cichocki, H. Schuster, D. Baker, Process-based e-service integration, in: Proceedings of the Workshop on Technologies for E-services (TES 2000), Cairo, Egypt, 2000.
[43] Q. Chen, M. Hsu, Inter-Enterprise Collaborative Business Process Management, HP Laboratories, Palo Alto, 2000, (Technical Report HPL-2000-107).

[44] Q. Chen, M. Hsu, U. Dayal, Peer-to-Peer Collaborative Internet Business Servers, HP Laboratories, Palo Alto, 2001, (Technical Report HPL-2001-14).
[45] D. Trastour, C. Bartolini, 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, 2001.
[46] J. Hendler, Agents and the semantic web, IEEE Intelligent Systems (IEEE IS) 16 (2) (2001) 30–37.
[47] D. Fensel, Problem-Solving Methods: Understanding, Development, Description, and Reuse, Lecture Notes On Artificial Intelligence, No. 1791, Springer, Berlin, 2000.
[48] M. Aksit, 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.