Adding Semantics to Web Service Descriptions

Hafedh Mili, Guy Tremblay, Abdel Obaid, Radhouane Bentamrout, Ann-Elisabeth Caillot
LAboratoire de recherche sur les TEchnologies du Commerce Electronique
LAboratory for research on Technologies for ECommercE
www.latece.uqam.ca
Département d'informatique, Université du Québec à Montréal
C.P. 8888, Succ. Centre-ville, Montréal (Québec) H3C 3P8

Abstract

Web services are emerging as a key infrastructure for providing inter-operation between applications and systems and for providing support for the deployment of e-commerce business processes. One important issue for ensuring the growth of Web services is having ways of describing the available Web services in a precise way.

Various languages and notations are currently available for describing Web services. However, the currently available notations generally provide little semantic information pertaining to the service's behavior, or provide it in a form which is strictly operational and algorithmic.

One possible step toward providing semantic information for Web services is through the use of formal contract specification—that is, using pre/post-conditions. We present a number of ways in which pre/post-conditions could be introduced into Web services descriptions, for specification as well as dynamic verification purpose. The use of pre/post-conditions, however, is not sufficient to describe the semantic of a group of related operations, for example, to describe the legal sequences in which these operations can/should be used. We discuss various ways in which such *protocol* descriptions could be provided for Web services, including the use of path expressions that show the order in which operations can and should be invoked. In addition, we also discuss the problem of *composing* together existing web services in order to satisfy some business need.

1 Introduction

Web services are emerging as a key infrastructure for providing inter-operation between applications and systems and for providing support for the deployment of e-commerce business processes. One important issue for ensuring the growth of Web services is having ways of describing the available Web services in a precise manner.

Precise specifications of services can be used for different purposes. First and foremost, when the service or system is yet to be implemented, they serve as a precise description of the problem to be solved, providing the implementers, and testers, with a specific description of the expected behavior. For those wishing to use some systems or services, appropriate specifications also help document, thus understand, the expected behavior. When searching for Web services, specifications can thus be used to narrow the search space. Finally, specifications, when sufficiently precise and formal, can also be used for verification purposes.

Although various languages and notations are currently available for describing Web services, there exists no consensus yet, even though some standards are emerging [WSDL-2.0, 2004, WS-CDL, 2004]. The lack of consensus is particularly clear when it comes to describing the behavior of business processes and their composition, whether it be in terms of orchestration (description of a specific business process) or in terms of choreography (description of the interactions among a group of business processes) [Peltz, 2003].

More importantly, some of the available notations (e.g., WSDL [WSDL-2.0, 2004]) provide little semantic information pertaining to the service's behavior. Other notations, although they can provide precise descriptions of behavior [Andrews et al., 2003, WS-CDL, 2004], do it in a form which is operational and algorithmic, that is, by describing the expected behavior through programming language-like structures. Such descriptions, however, may not necessarily be considered as being *specifications*—"a specification is a statement of properties required of a product, or a set of products". The distinction between a model of the behavior of the system and a specification of its expected properties is one which is made explicitly in model-checking approaches, where the formalisms for describing the model's behavior (e.g., automata, transition systems, process algebras) are generally quite different from those use to express the properties (e.g., temporal logic). The same characteristic is true for abstract model or contract-based approaches to specifications [Jones86,Meyer92.2], where the properties of the abstract model are expressed in some form of first order logic.

In this report, we first present a small number of languages, all defined as XML applications, that have been proposed for describing Web services and business processes [Mili et al., 2004, MendlingNueNut, 2004]. Although they are just a few among various other similar languages, they are representative of the major concepts and most appropriate for the level of specifications we intend to discuss. In the next section, we examine how various web services could be automatically composed together in order to satisfy some business need. Then, we examine how semantic information for Web services could be provided through the use of formal contract specifications—that is, pre/post-conditions. We present different ways in which such contracts specifications could be introduced and how these contracts could also be used for dynamic verification. However, as we also show, pre/post-conditions are still not sufficient to describe the semantic of a group of related operations, for example, to describe the legal sequences in which these operations can/should be used. We then discuss how WSDL service descriptions could be augmented with path expressions specifications, as a means to describe the order in which the various operations can and should be invoked, allowing process composition to be validated. Finally, we discuss related work that has been done in the area of model-checking applied to Web service descriptions.

2 Describing inter-process behavior

In this section, we attempt to paint an accurate and hopefully not too confusing picture of the process modeling languages landscape. In [Mili et al., 2003], we presented a broad review on process modeling languages,

2.1 WSDL

The Web Services Description Language is an XML-based language for describing web services in a way that is independent of their implementation technology. The description included in this section is based on the working draft (version 2.0) of the standard dated March 2004. Given the level of detail of the current presentation, we expect the features described here to survive in the final version of the standard.

The WSDL separates the abstract functionality of a web service from its concrete implementation. The abstract functionality is described in terms of *interfaces*—called *port types* in WSDL 1.1—which are collections of related *operations*. Roughly speaking, an operation is described by a *name*, a *signature*, and an invocation *modality* (one way/asynchronous or two way/synchronous); the signature is a sequence of *messages*, which are directed data flows whose contents are described in some type system—typically XML Schemas. The abstract description makes no mention of the specific location of the service (network address), message format (SOAP or other), or transport protocol (HTTP, TCP/IP). Such information is included in the *concrete* description or *binding*, which specifies message format and transport protocols for one or more interfaces. An *endpoint* associates a network address with a binding. Finally, a *service* groups together endpoints that implement a common interface.

We illustrate the main concepts through a simple example. Additional features will be introduced incrementally.

```
<?xml version="1.0"?>
<definitions name="StockQuote"

    targetNamespace="http://example.com/stockquote.wsdl"
          xmlns:tns="http://example.com/stockquote.wsdl"
          xmlns:xsd1="http://example.com/stockquote.xsd"
          xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
              xmlns="http://www.w3.org/2000/10/XMLSchema">
           <complexType name="TradePriceRequest">
              …
           </complexType>
           <complexType name="TradePrice">
              …
           </complexType>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" type="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
```

```
            <part name="body" type="xsd1:TradePrice"/>
        </message>

        <interface name="StockQuotePortType">
            <operation name="GetLastTradePrice"
                       pattern="http://www.w3.org/2004/03/wsdl/in-out">
               <input messageLabel="In"
                      element="tns:GetLastTradePriceInput"/>
               <output messageLabel="Out"
                       element="tns:GetLastTradePriceOutput"/>
            </operation>
        </interface>

       <binding>
            …
       </binding>
       <service>
            …
       </service>
   </definitions>
```

Figure 1. Excerpts from a WSDL specification.

In this description, we have a single interface, which has a single operation with one input message and one output message, each composed of a single *part*. The types of the messages are defined under the <types> tag as XML schema descriptions. In this example, the types are in-lined, but they can be included from another file[1]. The `pattern` attribute of the <operation> element points to a description of the *message exchange pattern* that characterizes the invocation of the operation. A message exchange pattern specifies the sequence and cardinality of messages exchanged by the operation. The pattern is defined in terms of *named placeholders*, and the mapping between the pattern and the messages of the operation is assured via the `messageLabel` attribute of message references—in this case, the not very creative `In` and `Out`. The WSDL draft specification provides eight common message exchange patterns (In-Only, Robust In-Only, In-Out, In-Optional-Out, Out-Only, Robust Out-Only, Out-In, and Out-Optional-In) [WSDL-2.0, 2004]. Others may be defined at will.

Operations can throw exceptions, called faults in WSDL. A fault is characterized by 1) its name, 2) its data contents, and 3) its direction (which distinguishes between exceptions/faults raised by the operation itself and those received by it). Faults are defined per interface (omitting direction information) and then referenced in specific operations using either an <infault>[2] element or an <outfault> element, enabling sharing faults between different operations. The message exchange patterns may specify how faults are handled (e.g. returned in lieu of the output message, or as a separate message [WSDL-2.0, 2004]).

Recent versions of WSDL support the specification of *features* and *properties*—missing in WSDL 1.1. A feature describes "… an abstract piece of functionality typically associated with

[1] The standard distinguishes between *including* definitions, which are considered to belong to the same namespace, and *importing* definitions which supports types (or other constructs) belonging to different namespaces.
[2] Infaults make sense for operations that follow an out-in like exchange pattern. In this case, the operation starts out by sending a request (the out message), and then receives a response (the in message). We have an <infault> when the request may cause an exception within its receiver. In this case, the response may consist of that fault.

the exchange of messages between communicating parties" [WSDL-2.0,2004]. Features include things such "security", "reliability", "transaction". The presence of a feature means that a service supports it and requires partners to do the same (e.g. secure communications). A feature may be defined using the following structure

```
<feature
      uri="xs:anyURI"
      required="xs:boolean"? >
  <documentation />?
</feature>
```

where the URI refers to a document that includes the definition of the feature. The required attribute is self-explanatory. A feature may have accompanying documentation, like all WSDL components (interfaces, operations, messages, etc.). Features can have different *scopes*, including interfaces-level, operations-level, or message-level. The scope of a feature is inherent in the position of the <feature> element in the WSDL document tree: if the feature is a child of an <interface> element, then it is interface-wide, and so forth. Message-level features override operation-level features, and operation-level features override interface-level features.

Similarly, properties are used to describe what appear to be the service's operational parameters[3], and may be used to control the behaviour of a feature. Properties are defined as follows:

```
<property
      uri="xs:anyURI"
      required="xs:boolean"? >
  <documentation />?
  [ <value /> | <constraint /> ]
</property>
```

For a property, we can specify values or constraints on values. An example property may be the encryption algorithm, and the constraint may state that it must be a public key encryption algorithm. This property controls the behaviour of the *security* feature. Values and constraints may be defined by referring to types defined using XML Schemas. Properties have similar scopes, and a similar composition model to that of features, with property values and constraints of narrower scopes overriding values and constraints of broader ones.

Because we are mostly interested in the specification of web services, we will only briefly comment on the concrete part of WSDL specifications. WSDL enables us to separate the abstract specification of web services from their concrete implementation. The concrete part of a WSDL specification consists of the notion of *service*, which is the concrete implementation of a *single* interface. An interface may be offered at different access points (called *endpoints*). Each end point may use a different message format (e.g. SOAP) or message transmission protocol (e.g. TCP/IP or HTTP). Figure 2, taken from [WSDL-1.2,2003] illustrates this point. The bindings may specify message formats and message transmission protocols for an entire interface, or for a specific operation, or for a specific message or fault of a particular operation.

[3] The working drafts for WSDL 1.2 [WSDL-1.2,2003] and WSDL 2.0 [WSDL-2.0,2004] are vague about what properties mean, and include no examples. The definition from [WSDL-2.0,2004] says "A Property component describes the set of possible values for a particular property".

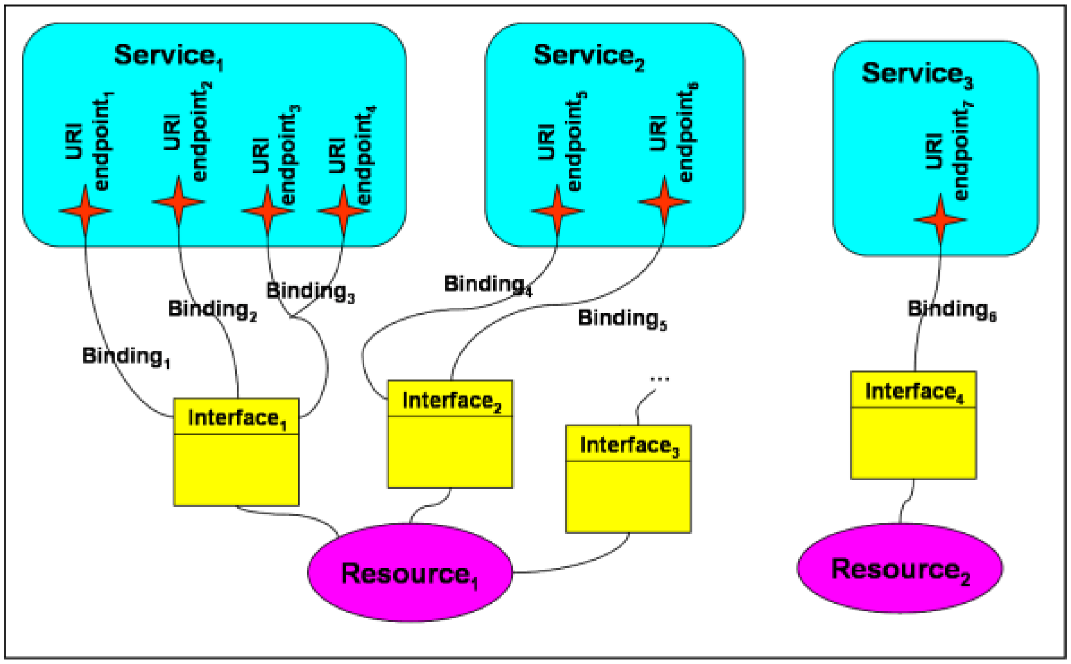

Figure 2.Showing the relationship between interfaces, bindings, end points, and services.

Figure 3 shows a first cut object meta-model for the abstract aspects of web service descriptions. Note that here we are *not* modeling the structure of WSDL documents; instead, we are presenting the *conceptual model* underlying web service descriptions, regardless of how that model is *serialized* within XML documents. For example, WSDL documents include definitions of components (e.g. types, messages, faults), and then *references* to those components. In this meta-model, the definition of the component is embodied in the class that represents the components, and references to it are simply associations to that class. This is best seen in the case of messages, faults, and message exchange patterns.

The more specific features override more general ones.
Erreur ! Aucune rubrique spécifiée.

Figure 3. A web service meta-model.

2.2 BPEL4WS

The Business Process Executable Language For Web Services (BPEL4WS, [Andrews et al., 2003]) is a language for modeling business processes, executable or not, written for the Web services context. The basic idea is that a process may be thought of as a collaboration between services or tasks described in the Web services format, more specifically, in the WSDL language. Whereas WSDL defines a syntax for expressing the interface of services (in the IDL sense, i.e., signature of operations), but says little about the interaction model (or rather assumes a simple one-way or round-trip message passing protocol), BPEL4WS allows to describe the entire interaction sequence. BPEL4WS has two target uses, which are clearly stated and explained:

1) executable processes: these describe actual business processes that are internal to an organization and are completely specified (i.e., executable);

2) abstract processes, also known as business protocols: these are the parts of a business process of an enterprise that are *exposed* to outside processes in the context of an inter-enterprise interaction.

This distinction is very helpful and not made in the case of BPML (see section 4.6). This, plus the fact that BPEL4WS makes provision for *roles* and *partners*, makes it more appropriate for describing inter-organizational processes.

Roughly speaking, a BPEL4WS process description consists of a declaration part that introduces various elements needed to describe the process, followed by the actual description of the process, i.e., its behavior. The declaration part includes the following elements:

1) *A description of the messages exchanged between services*. The message structure is similar to that used in Web services: a message consists of *parts*, each with a name and a type. The type component is described using XSD types, as with BPML. The use of XSD is not exclusive, and BPEL4WS can accommodate other type systems.

```
…
<message name="POMessage">
      <part name="customerInfo" type="sns:customerInfo"/>
      <part name="purchaseOrder" type="sns:purchaseOrder"/>
</message>
<message name="InvMessage">
      <part name="IVC" type="sns:Invoice"/>
</message>
…
```

2) *A description of the services invoked.* The description follows the WSDL standard: each service, defined as a *portType*, consists of a bunch of operations. An operation has a name and a set of parameters. The parameters are nothing but messages playing the role of inputs and outputs. A service is defined using WSDL port type (now known as interface) concept.

```
…
<portType name="purchaseOrderPT">
      <operation name="sendPurchaseOrder">
            <input message="pos:POMessage"/>
            <output message="pos:InvMessage"/>
            <fault name="cannotCompleteOrder"
                              message="pos:orderFaultType"/>
      </operation>
</portType>
…
```

3) *A description of the contracts between process participants.* Each contract—called *partner link type*—defines roles and associates them with port types (interfaces). For example, in a supply chain example, we have customers, businesses, and their suppliers. The interaction between a customer and the business to fulfill an order, and between the business and its suppliers to restock, are managed by two separate contracts/partner link types, each of which identifies the *roles* played by each service interface (port type). One of the role describes what is *provided* by the process being described, whereas the other role described what is *required* from the other partner; when no specific requirements is placed on the expected partner, a single role can be specified. For example, the following describes a contract where an `invoiceService` provides access to a `computePricePT` port type but in turn requires the other partner to provide an `invoiceCallbackPT` port type on which the answer can be sent back:

```
…
<plnk:partnerLinkType name="invoiceLT">
      <plnk:role name="invoiceService">
            <plnk:portType name="pos:computePricePT"/>
      </plnk:role>
      <plnk:role name="invoiceRequester">
            <portType name="pos:invoiceCallbackPT"/>
      </plnk:role>
</plnk:partnerLinkType>
…
```

4) *A description of partners*. While service link types define the various contracts, it does not specify which entity will play which side in the contract. A BPEL4WS process thus contains an identification of the various partners in the different contracts, the roles they play in the contract (partner link type), and the role ("myRole") the enterprise doing the modeling plays in that contract. Those partners will be referred to later in the description of the steps of the process: each step is performed by a partner (or self). Here, we show a description of two partners, where the first is the customer and the second is the `invoiceProvider` playing the role `invoiceService` in the contract (partner link type) `invoiceLT`, where I, myself, play the role of `invoiceRequester`.

```
…
<partners>
      <partner name="customer"
            serviceLinkType="lns:purchaseLT"
            myRole="purchaseService"/>
      <partner name="invoiceProvider"
            serviceLinkType="lns:invoiceLT"
            myRole="invoiceRequester"
            partnerRole="invoiceService"/>
      …
</partners>
…
```

Other elements in the declaration part include *local variables* defined within the scope of the process and exchanged as inputs/outputs between the process steps, and a description of *fault handlers*, which specify the desired response in case of a fault.

The process (i.e., its dynamic behavior) is defined using a flow, which is a partially ordered set of activities that correspond to invocation of operations, defined in the various services, that will be performed by the partners identified above. Process flow supports sequential activities (using the "sequence" activity), concurrent activities (using the "flow" activity), and arbitrary control dependencies between process steps using the "link" mechanism, which ensures that a particular process step can only be executed after another step has completed. Like with BPML processes, BPEL4WS processes include descriptions of *compensation handlers* and support the notion of *scope* (called *context* in BPML) and *correlation sets*, which are data values that uniquely identify process instances (see also BPML). The following shows excerpts from the body of the process. The (purchase order handling) process starts by receiving a purchase order (PO) from a customer. It then makes a copy of the customer info from PO into the customer info of a shipping request. Then it invokes an operation of the partner “shippingProvider” to ship the order to the customer. This example illustrates the use of control dependencies: the “requestShipping” operation is the source of a control dependency (a *link*, called “ship-to-invoice”) linking it to the operation (not shown here) “sendShippingPrice” of the partner called “invoiceProvider”.

```
…
<sequence>
      <receive    partner="customer"
```

```
                        portType="lns:purchaseOrderPT"
                        operation="sendPurchaseOrder"
                        variable="PO">
            </receive>
            <flow>
                  <links>
                        <link name="ship-to-invoice"/>
                        <link name="ship-to-scheduling"/>
                  </links>
                  <sequence>
                     <assign>
                        <copy>
                              <from variable="PO" part="customerInfo"/>
                              <to variable="shippingRequest"
                                    part="customerInfo"/>
                        </copy>
                     </assign>
                     <invoke  partner="shippingProvider"
                              portType="lns:shippingPT"
                              operation="requestShipping"
                              inputVariable="shippingRequest"
                              outputVariable="shippingInfo">
                        <source linkName="ship-to-invoice"/>
                     </invoke>
                   ...
                 </sequence>
                 …
            </flow>
            …
      </sequence>
      …
```

Figure 4 shows a meta-model of BPEL4WS. Note that we didn't add a link from invoke and service (although the XML tag for invoke requires a port type/interface name) because, in XML, two operations can use the same name, whereas here we are showing *objects* which have unique identities, but possibly the same names.

The meta-model of Figure 4 shows a process as a composite activity. The structure of a composite activity is described by a **StructuringConstruct** which, in programming languages terms, is the equivalent of a control structure. We show here two such constructs: **Flow**, whose components are activities that can proceed in parallel—barring any control dependencies, see **Link**s below—whereas the components of a **Sequence** proceed sequentially. Composite activities define scopes within which variables may be defined. Those variables will be bound to messages or parts of messages, or expressions thereof. Atomic activities can be of several kinds. One notable kind is **Invoke**, which represents an operation invocation whose actual (invocation) parameters are variables defined in the scope of the enclosing composite activity. Other basic activities include minimal operations needed to glue together operation invocations, and include things such as reception of message (**Receive**), assignments (assigning values to variables and messages back and forth to variables).

The top left corner of Figure 4 shows the association between *roles*, *services*, and *service link types*. Services correspond to WSDL 2.0's interfaces. Service link types are contracts involving one or two roles. Moving to the top center, **Partner**s may play different roles in different contracts.

BPEL4WS, unlike WSDL, makes the notion of contract *explicit*. A contract (**ServiceLinkType,** `i.e., a partnerLinkType`) is simply an association between two services. It is implicit that the operations within those services collaborate through message exchange. An example of such an exchange is given in the description of the process, but this may not be the *only* valid exchange.
Erreur ! Aucune rubrique spécifiée.

Figure 4. BPEL4WS (partial) meta-model.

.

2.3 *WSCI and WS-CDL*

The Web Services Choreography Language is the latest initiative from the *Web Services Choreography Group* of the World Wide Web Consortium (W3C). This language has an interesting history. The draft proposal for WSCI (Web Service Choreography Interface) was issued in August 2002. That effort seems to have been abandoned, and a new standardization effort launched under the name Web Service Choreography Description Language (WS-CDL), which, for all practical purposes, started from scratch. Indeed, it started with a *new* requirements document, from which the *new* language, WS-CDL, was built from the ground up, with no reference to the WSCI effort. Because we feel that there were some useful ideas in WSCI, we start by describing WSCI. We then describe novelties in the emerging WS-CDL standard.

The premise of WSCI is that WSDL descriptions are insufficient to describe the true nature of services. Indeed, WSDL defines services in terms of collections of seemingly unrelated operations (*port types, i.e., interfaces*), but has no notion of state, and does not tell us the valid sequences of operation invocations. Using the classical example of a travel agent service, a traveler can confirm an itinerary—e.g. invoke some operation `confirmItinerary(Itinerary it)`—only if that itinerary has already been *created*—e.g., by invoking an operation `createItinerary(Customer c, Date depart, Date return, Destination dest)`—and if it was created explicitly *for that same customer*. Yet, nothing in a WSDL description makes those constraints explicit. To some extent, WSCI brings statefulness into web service descriptions both explicitly by associating properties (variables) with interface definitions, and *implicitly* by stating the valid operation invocation sequences. Further, WSCI considers that the same web service can be used by several different *processes*, and thus, a given service (WSDL interface) may be associated with several valid sequences of uses, where each one would be characterized by a specific process description. These are the basic premises of WSCI.

Figure 5 shows excerpts of a simplified version of the travel agent service. Notice this example uses WSDL 1.1's syntax, and hence, web services are represented by *port types* instead of *interfaces*. The <interface> tag used here is WSCI-specific, and will be discussed further below.

```
<? xml version = "1.0" ?>
<wsdl:definitions name = "Travel Agent Dynamic Interface"
   targetNamespace = "http://example.com/consumer/TravelAgent"
   xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd = "http://www.w3.org/2000/10/XMLSchema"
   xmlns:tns = "http://example.com/consumer/TravelAgent"
   xmlns = "http://www.w3.org/2002/07/wsci10">                1.
   <!-- WSDL complex types -->                                   2.
   <!-- WSDL message definitions -->
<-- ************************************************************* -->
<-- ************TRAVEL AGENT PORT TYPES ************************ -->
<-- ************************************************************* -->
   <portType name = "TAtoTraveler">
     <documentation>
        This port type references the operations the Travel Agent
        performs with the Traveler service
     </documentation>
     <operation name = "ReceiveTrip">
        <input message = "tns:tripOrderRequest"/>
        <output message = "tns:tripOrderAcknowledgement"/>
     </operation>
     <operation name = "BookTickets">
        <input message = "tns:bookingRequest"/>
        <output message = "tns:bookingConfirmation"/>
     </operation>
     <operation name = "SendStatement">
        <output message = "tns:statement"/>
     </operation>
   </portType>
<-- ************************************************************* -->
<-- ******************** WSCI additions ************************ -->
<-- ************************************************************* -->

   <correlation name = "itineraryCorrelation"                   3.
            property = "tns:itineraryID">
   </correlation>

   <interface name = "TravelAgent">                             4.
     <process name = "PlanAndBookTrip"                          5.
          instantiation = "message">                            6.
        <sequence>                                              7.
           <action name = "ReceiveTripOrder"                    8.
                   role = "tns:TravelAgent"                     9.
                   operation = "tns:TAtoTraveler/OrderTrip">    10.
           </action>
           <action name = "ReceiveConfirmation"                 11.
                   role = "tns:TravelAgent"
                   operation = "tns:TAtoTraveler/bookTickets">
              <correlate correlation="tns:itinCorrelation"/>    12.
              <call process = "tns:BookSeats" />                13.
           </action>
            <action name = "SendStatement"                      14.
                   role = "tns:TravelAgent"
                   operation = "tns:TAtoTraveler/SendStatement"/>
           </action>
        </sequence>
     </process>
     <process name = "BookSeats" instantiation = "other">        15.
        <action name = "bookSeats"
           role = "tns:TravelAgent"
           operation = "tns:TAtoAirline/bookSeats">
        </action>
     </process>
   </interface>
```

```
</wsdl:definitions>
```

Figure 5. An example WSCI extension. Taken from [WSCI 1.0, 2002].

In this example, the travel agent web service includes two port types (*interfaces* in WSDL 2.0), `TAtoTraveler`, whose definition is shown above and which interfaces with travelers, and `TAtoAirline`, which interfaces with airlines and whose definition is not shown but is referenced in the process `BookSeats` (line 15). The valid message sequences that the travel agent web service supports are enclosed in the element **<interface>** under WSDL's top-level **<definitions>** element. Each valid interaction sequence is represented by a **<process>**, and each process is a combination of **<action>**s, where each action corresponds to the invocation of an operation (lines 8-10). The process called `PlanAndBookTrip` (lines 5 –14) is a simple **<sequence>** of actions. Other compositions include **<all>** (parallel execution), **<foreach>**, and **<switch>**. Line 3 shows the declaration of a **<correlation>** element, which is referenced in line 12. The idea here is that a travel agent service would typically be handling several "conversations" (process *instances*) with different customers or with the same customer but for different itineraries. The **<correlation>** element specifies how to uniquely identify a given process instance (or a given thread of conversation). This is identical to BPEL4WS's notion of *correlation set*. Also, processes define *scopes*—called *contexts*—within which variables may be defined—called *properties.* These properties are typically used to hold data values (i.e., state information) that are exchanged and manipulated by the different operations of a process.

Also, not shown in this example, processes can raise exceptions. Those exceptions would be specified, along with the steps to handle them. Processes can also specify transaction boundaries around a set of actions to indicate that the whole set should be treated as an atomic action, as well as compensation activities to specify what to do when transactions fail (using a **<compensate>** activity).

In addition to representing the dynamic behavior of individual services, WSCI includes the notion of a *global model*, which binds together several interfaces, one per web service. In the travel example, the travel agent service interfaces with both travelers and airlines. Similarly, airlines interface with both travel agents—for booking and pricing, say—and with travelers, for ticket delivery. In a global travel scenario, operations from the travel agent service will be talking to operations from the airline web service. The *global model* simply connects operations from the various interfaces. Figure 6 shows part of the global model for the travel scenario. A **<connect>** element associates the operation that initiates an exchange with the one that completes it.

```
<?xml version = "1.0" ?>
<wsdl:definitions   name = "GlobalModel"
   targetNamespace = "http://example.com/consumer/models"
<!--various imported namespaces -->
    …
    >
<!--various imports -->
   <wsdl:import   namespace = "http://example.com/consumer/traveler"
                location = "http://example.com/traveler.wsci" />
    …
<model name = "AirlineTicketing">

     <interface ref = "air:Airline" />
```

```
          <interface ref = "tra:Traveler" />
          <interface ref = "ta:TravelAgent" />

     <!-- Traveler / TravelAgent -->

          <connect operations = "tra:TravelerToTA/PlaceItinerary
                                 ta:TAtoTraveler/ReceiveTrip" />
           …
     <!-- Travel Agent / Airline -->

          <connect operations = "ta:TAtoAirline/CheckAvailability
                                 air:AirlineToTA/VerifySeatAvailability"/>
          …
     <!-- Traveler / Airline -->
          …
        </model>
     </wsdl:definitions>
```

Figure 6. Global model for travel scenario. Taken from [WSCI 1.0, 2002].

WSCI's meta-model is shown in Figure 7. For convenience, we showed in dashed boxes the part of the meta-model that describes the global model and the part that is inherent in WSDL. The part of the meta-model that represents processes is not much different from that of BPEL4WS—or BPML [BPMI,2003], for that matter: processes are composite activities that are aggregated from other activities using control structures of the kind found in programming and process languages. Atomic activities include operation (and process) invocations, and atomic message sends/receipts. Composite activities define scopes (*contexts*) within which local variables may be declared. Local variables are used to exchange values between operations, and are often computed from input or output messages. Further, different process instances are distinguished using correlations, which in turn are computed from messages. Unlike BPEL4WS, WSCI explicitly supports the notion of *transaction.*
Erreur ! Aucune rubrique spécifiée.

Figure 7. WSCI's (partial) meta-model.

WSCI's descendant, called *Web Services Choreography Description Language* is still very much in its infancy [WS-CDL, 2004]. The description of the *choreography* itself is not much different from that of *process* in WSCI or BPEL4WS. At the interface level, WS-CDL uses the participant, role, and relationship triad whereby participants play roles in relationships (contracts). In this regard, WS-CDL looks more like BPEL4WS than WSCI.

2.4 RosettaNet

RosettaNet is a consortium grouping more than four hundred companies from IT, electronic components, semiconductor manufacturing, and telecommunications, working "to create and implement industry-wide, open e-business process standards … [that] form a common e-business language, aligning processes between supply chain partners on a global basis" (see http://www.rosettanet.org). The RosettaNet philosophy postulates that for inter-enterprise e-business to take place, companies have to achieve compatibility at different levels of the computational infrastructure. Accordingly, the consortium has sought to standardize those levels. Figure 8 illustrates the different layers and the relevant standards. The left hand side of the figure illustrates the different layers of the standards, starting from the alphabet for communication between peer information systems, to the specific vocabulary used in transactions, to the implementation framework, dialog, process, and then applications. We will describe each layer in some detail below.

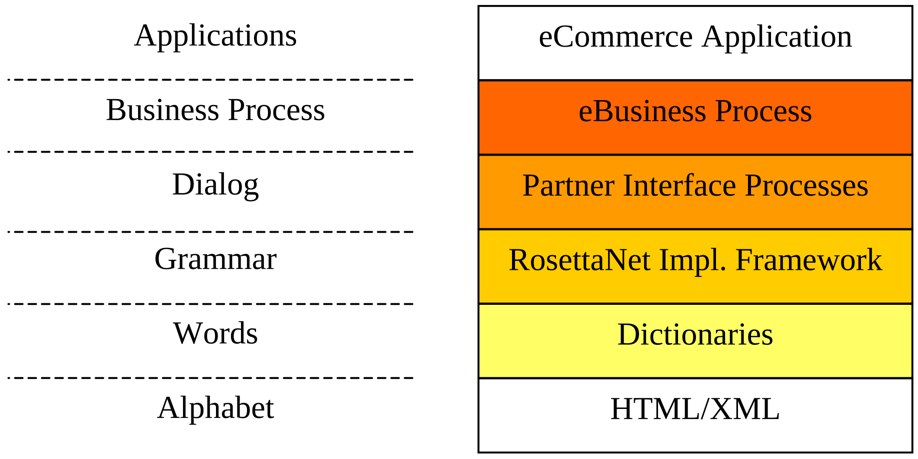

Figure 8.Structure of the RosettNet Standard.

The *dictionaries* define a common vocabulary that can be used in electronic transactions. The standard includes two kinds of dictionaries, a *RosettaNet Business Dictionary*, and the *RosettaNet Technical Dictionary*. The business dictionary includes the definition of industry-independent generic business terms (**AccountInformation**, **BillOfMaterial**, **FeeInformation**, **DeliveryException**, etc)**.** The technical dictionary provides a common vocabulary for describing *products* and *services* in target industries—mostly electronic components and information technology—with terms such as **AMPLIFIER IC – RF**, **TRANSISTOR – RF**, **ANALOG MODEM**, **PRINTER INKJET**, and **KEYBOARD**.

The Partner Interface Processes (PIPs) describe generic processes involving two—typically—or more partners. The PIPs are grouped under *segments*, themselves grouped under seven *clusters*, covering different process areas, including *Product Information*, *Order Management*, *Inventory Management*, and *Manufacturing*.

Partner Interface Processes, as the name indicates, are processes that happen at the interface between partners. A typical process starts with one or more tasks within one partner, followed by one message send, followed by more tasks in the second partners, possibly followed by sending a reply to the first partner or a message to another partner, and so forth. Figure 14 shows an example process. The process 3A8[4] is for requesting a purchase order change. It is represented using an activity diagram like notation with two "swim lanes", one for each partner. A PIP may refer to other PIPs either in its preconditions—in this case assuming that another PIP 3A4 has already taken place—or in the steps of the process—in this case the launching of a follow-up PIP, namely 3A7.

[4] The number 3 is the number of cluster (Order Management) and A the number of the segment (Quote and Order Entry).

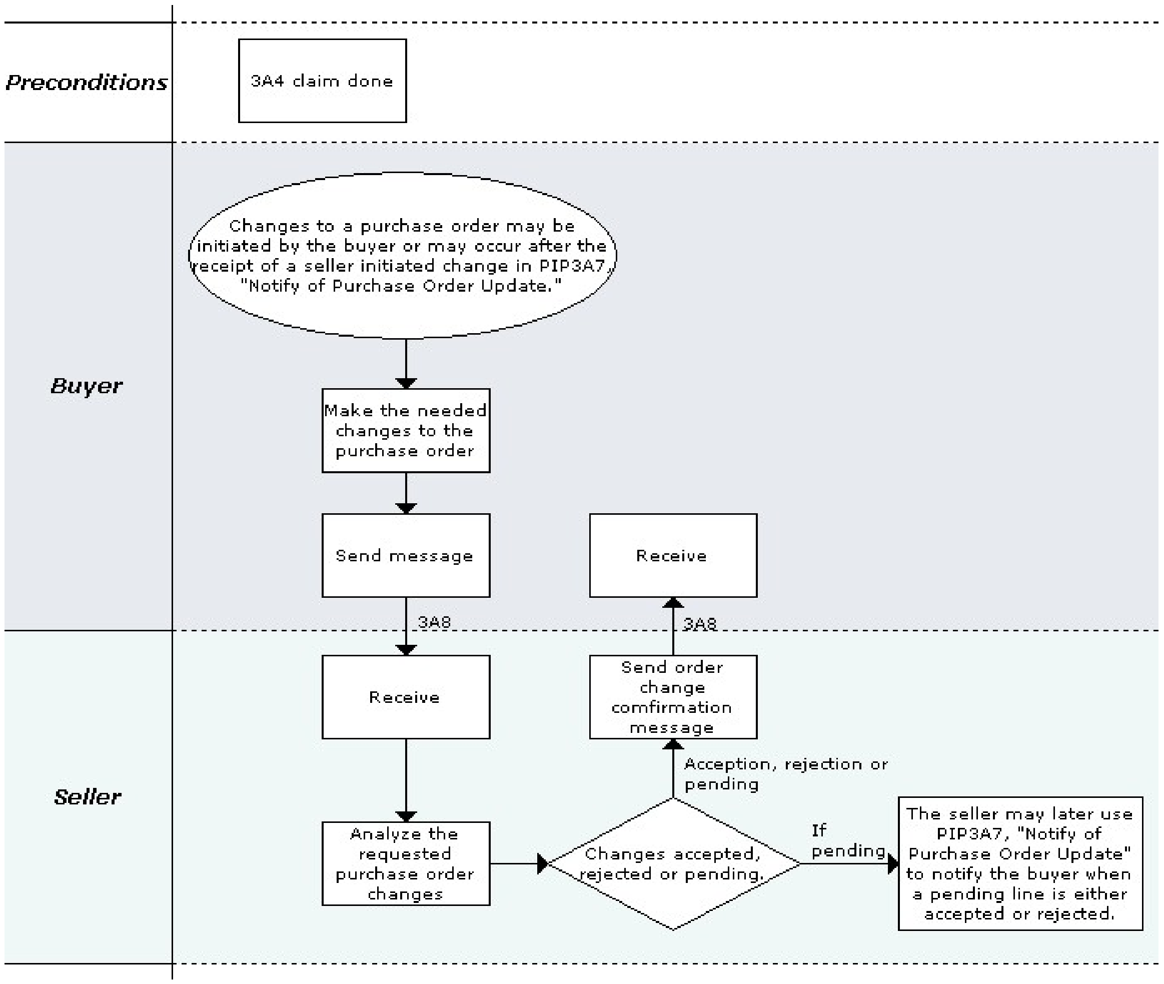
Preconditions
3A4 claim done
Buyer
Changes to a purchase order may be initiated by the buyer or may occur after the receipt of a seller initiated change in PIP3A7, "Notify of Purchase Order Update."
Make the needed changes to the purchase order
Send message
Receive
3A8
3A8
Seller
Receive
Send order change comfirmation message
Acception, rejection or pending
Analyze the requested purchase order changes
Changes accepted, rejected or pending.
If pending
The seller may later use PIP3A7, "Notify of Purchase Order Update" to notify the buyer when a pending line is either accepted or rejected.

Figure 9. PIP for requesting that a change be made to an existing order.

PIP specifications include three *views* on the process: 1) the *business operational* view, representing the business semantics as illustrated by Figure 9; 2) the *functional service view*, describing the required network components (embodying partners) and their protocols, and the mapping between PIP actions and documents; 3) the *implementation framework view*, describing the message formats and specific message formats. Message formats are provided as XML document type definitions (DTDs).

The RosettaNet Implementation Framework (RNIF) specification specifies exchange protocols for PIPs that, when followed, should enable participating supply chain members to inter-operate. The specification covers business message formats, specifying XML usage guidelines and message component formats, security provisions, procedures and rules for assembling (packing) and disassembling of messages (unpacking), message transfer protocols, and message flow semantics (one-action, request/response, handling failure, etc.).

The process that the RosettaNet organization uses to derive these specifications starts with the modeling of existing business processes by process specialists from the relevant industries or process areas. These "as-is models" are later analyzed to find commonalities, leading to generic business processes which are then used to identify PIPs to be specified. Those generic processes correspond to the upper layer of the RosettaNet standard (see Figure 8) and would typically involve more than one interaction (request/response) between partners, but also includes private tasks. We understand that, in practice, they are more of an intermediate deliverable of the process of specifying PIPs, rather than a standardized output in and of itself. To the best of our knowledge, those generic processes have not been published.

From an ontological point of view, RosettaNet does not add new concepts to the ones introduced by the standards mentioned above. The notation used for process descriptions looks closest to Ould's *role activity diagrams* [Ould, 1995]. This is illustrated in the diagram of Figure 9[5]. However, PIPs differ from the notion of a service supported by the previous three languages (WSDL, BPEL4WS, and WSCI/WS-CDL): whereas the notion of a service implied by WSDL, BPEL4WS and WSCI/WS-CDL includes only the operations that are publicly visible, RosettaNet PIPs provide ample context for the exchanges between partners, by providing: 1) preconditions for exchanges, but also 2) chains of activities preceding an actual exchange.

For our purposes, the contents of the PIPs themselves can be used as a basic business vocabulary against which process descriptions may be validated. More on this in section 4.2.

2.5 Discussion

WSDL describes web services in terms of collections of seemingly unrelated operations that users can invoke in any particular order. This may be true for services offering *basic* search capabilities into an information database, but is rarely the case with services that represent access points to elaborate business processes. More is needed.

BPEL4WS enables us to describe business processes that involve the interaction of several business partners. When the process under description is internal to an organization, thus private, the partners could be departments or divisions within the enterprise. When the process is public, the partners would be different enterprises. In either case, the functionality supported by a specific partner is expressed in terms of a WSDL web service. While the WSDL descriptions of the individual partners are stateless, the structure of the overall process shows the collaboration of and interaction between their services for the purpose of executing a global process. A BPEL4WS specification shows an overall choreography of operations from various partners in the context of an inter-organizational process.
WSCI has attempted to address WSDL's statelessness by providing valid sequences of operations within each service, *regardless of any specific global process in which a given service might be involved*. This is unique to WSCI: defining the *range of stateful behaviors* that a given service can support independently of a *usage context*. WSCI's *global model* does provide an example of such a context.

It seems to us that at least two distinct concepts are needed to capture the semantics [illegible]eb services *individually*:

1) the web service API of sorts, represented in a WSDL format,
2) the *range* of behaviors supported by the web service in terms of valid sequences of operation calls, somewhat à la WSCI.

[5] Which is an out[illegible]f a process modeling tool, called QPRProcessGuide™, by QPR Limited.

Are these elements enough to capture the semantics of web services? One of the traditional way of representing operation semantics is by using preconditions and post-conditions. In the present case, preconditions would be logical expressions about the state of the system (service) and the messages that need to be satisfied for the call to be valid. Post-conditions are logical expressions about the state of the system and the set of messages that are guaranteed to be true at the end of the operation. Note that we do not restrict preconditions to input messages and post-conditions to output, for two different reasons. First, web service operations that are of the request reply kind start with an output message (request). Second, WSDL does not support call-by-reference semantics: an operation that transforms its input is written as having an input (initial value) and an output (transformed value).

We had some debate as to whether strong enough pre/post-conditions would obviate the need for "path expressions" *à la* WSCI, which would simply be valid traces provided mainly for documentation purposes. However, if we think of operations as simply steps in overall processes, then we must have an idea about what those processes are. For example, we could model the semantics of the operation that checks itineraries with the post-condition that the returned itineraries (output message) correspond to the requested dates and destination (input message). However, this does not tell us that each enquiry should conclude either with a cancellation or with an actual booking. Path expressions *à la* WSCI enable us to say that. In conclusion, three pieces of information are required to describe the semantics of web services, taken in isolation:

1) the web service API of sorts, represented in a WSDL format,
2) pre/post-conditions on the individual operations, and
3) path expressions, which show the valid sequences of operation calls.

To describe a business process that involves the collaboration of several services—basically, what BPEL4WS is all about—we need to:

1) specify the partners, and
2) describe the orchestration of the process itself.

To some extent, BPEL4WS describes *a* process as a collaboration of web services, but does not provide (or document) the range of *all* processes that these web services can support: that information should be provided separately in the form of pre/post-conditions and valid path expressions. Conversely WSCI does provide the valid path expressions, but does not provide an explicit description of the inter-service process. Its *global model* is simply a list of bindings associating senders with receivers of messages, but does not describe an actual end to end process.

Accordingly, for the purposes of representing inter-service collaborations, we should use BPEL4WS or BPEL-like languages. We understand that WS-CDL—the descendant of WSCI—looks more like BPEL4WS than WSCI. That is too bad because WSCI provides some useful concepts that are missing from BPEL4WS. Figure 10 shows a meta-model of web services and inter-service processes that embodies the ideas discussed above.

Figure 10. A meta-model of web services.

3 Problems in service composition

In this section, we look at the problem of composing services. This problem motivated the search for ways to express the semantics of web services.

We first look at two composition scenarios, with increasing complexity. Next, we look at the potential mismatches between the composed processes, and then suggest ways of addressing the problems

3.1 Looking for a single operation

In the simplest case, a user or client service is looking for services that support a given operation. If all we have is WSDL descriptions, then this becomes a signature matching problem. The operation names have no meaning. However, the input and output messages need to be matched.

Matching input and output messages can be done using two different strategies, as is done in type-checking: name equivalence and structural equivalence. The former is used in languages that enforce strong type checking (e.g., Ada), whereas the latter is used in languages with looser type checking (e.g., C). Assuming the looser interpretation, this essentially reduces to the problem of matching data structures. The parts within a given message are not ordered. What we need to be able to do is to build trees that represent input/output messages, and try to match those trees. The intermediate nodes may not matter since this is mostly a matter of nomenclature. Further, it may also be a matter of style: because XSD enables us to use anonymous types, some schema writers will use anonymous types for those structures that are used in a single place.

The question becomes: when do we say that we have equivalence

1) isomorphic structures, through some mapping between labels
2) equivalent trace traversal (i.e., only leaf nodes, ordered in some way, matter)
3) some combination thereof

Existing efforts that aim at standardizing business documents should alleviate the problem of matching input and output messages, since they will use a common set of types, but still only up to a point: the tree structures that represent the messages would simply be shallower.

If pre/post-conditions are used to describe operations, they should be specified in queries as well. We can assume that we only look at pre/post-conditions if the message data structures match.

We assume in the above that we are not interested in the operation within the context of a process, and therefore, we won't be looking at path expressions. We also assume that the modality—one-way, request-response, solicit-response, or notification—of the operations to be matched are the same. Matching operations with distinct modalities is clearly more complex, as this is not based on a one-to-one match between operations—for instance, a request-response operation could be matched by the combination of a one-way operation together with a notification operation. (Or is this part of the operation coverage issue?)

3.2 Looking for operation coverage

The idea here is that a service client submits a query for an operation, and the output could be a single operation or a composition of operations that cover the query (see e.g. [Hall,1993], [Mili et al., 1994].). Figure 11 illustrates the problem. Figure 11 shows an instance of the problem where the functions can have any number of inputs. The mapping to web service operations' messages is fairly straightforward.

We showed in [Mili et al., 1994] that computing the coverage of a function given a library of functions has a time-complexity which is polynomial in the number of functions (?). Experiments with a C++ library yielded an overwhelming number of false positives, i.e., proposed function compositions that didn't produce the desired functionality. This is due in no small part to the nature of the C++ library that we used: it was a domain independent library providing a number of utility classes. Hence, most of the input/output types were ints, chars, or pointers. In [Hall, 1993], Hall used a similar function composer that actually applied the composition to test data to compare the obtained result with the desired one. Hall dealt with Lisp functions, which have an even looser typing system. However, Lisp supports run-time composition of functions, which made the verification of the compositions easy.

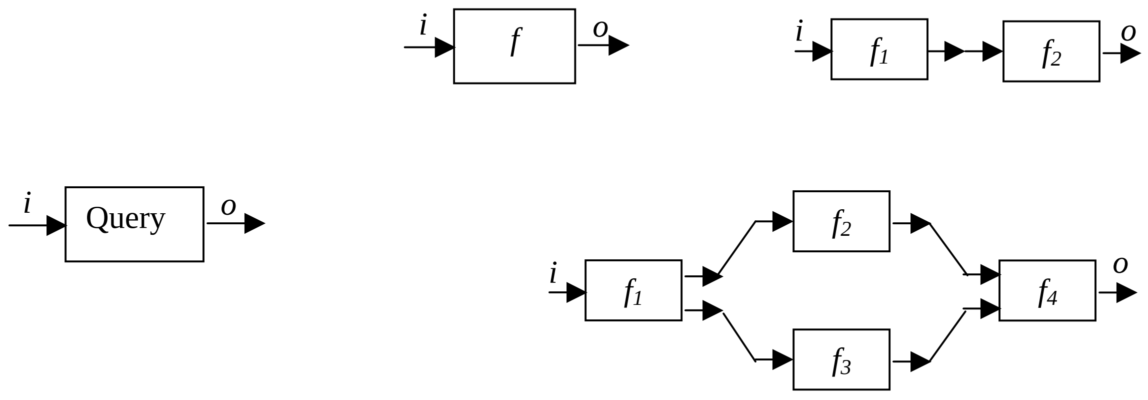

Figure 11-a. The problem Figure 11-b. Some potential solutions

Figure 11. The problem of looking for an operation coverage

In the context of web services, the exchanged messages are business documents with rich contents, and we expect type matches to be less fortuitous than was the case in [Mili et al., 1994].

One of the issues that we have to deal with is whether we should be looking for compositions of operations across services or within the same service. There are plausible cases for both. Consider the case where we try to a book a flight from an airline. Assume that the user submits an itinerary (with preferred hours of day) with seating preferences, and expects a reservation and seat assignment in return. Most airline reservation systems would check seat availability first, and if confirmed, deal with seat assignment. Thus, the search for an operation or service that does both might combine two operations of the same service, to be executed in succession, the first to reserve a place in the airplane, and the second to pick a seat, given a reservation. If we take into account the path expressions, the composition also needs to satisfy the path expressions of the service.

We can also think of a query that looks for an operation that takes an itinerary as input, and then returns flights, hotel reservations and car rentals at the destination. In this case, the flights would need to be reserved from an airline web service, the hotel reservations from a hotel web service, and the car rentals from a car rental web service. In this case, the same inputs (destination and duration of stay) may be used by all three services.

3.3 Looking for an inter-organizational partner

This is the case where we have a process defined in terms of a collaboration between several roles and where we are looking for a partner that can fulfill one of the roles. A plausible scenario is a case where some manufacturer needs parts A, B, and C to manufacture some finished product P, and wants to keep as low an inventory of parts as possible. Just in time manufacturing would require close collaboration between the manufacturer and the suppliers of parts A, B, and C. Suppose part A supplier P_A got together with part B supplier P_B and started looking for a supplier of part C that can help them bid jointly for the manufacturer's business. (?)requests a joint bid by suppliers of products A, B, and C.

The inter-organizational process may be given as a BPEL4WS specification. In this case, the web services associated with the various roles do not correspond to actual services known to be offered, but correspond to *required* services that potential partners will have to support. Further, the partners do not correspond to actual known partners (e.g. suppliers P_A and P_B) but to virtual partners that can play certain roles in certain contracts (partner link types). For example, the manufacturer should be equally happy if parts A, B, and C were supplied by the same or by different suppliers: certain parts have to be delivered in determined quantities at specific times to make the process work, and that is all that matters. Internally, a given partner may optimize its process to handle all the orders coming from the same manufacturer together (e.g. combined shipping, possibly with temporary holding locations) but those internal optimizations should have no bearing on the inter-organizational behavior[6].

Finding partners that can play certain roles in an inter-organization process should then look for web services that support, minimally, the set of operations that are invoked by the inter-organizational process. Further, out of the candidate web services, we should check that their acceptable path expressions (à la WSCI) are consistent with the inter-organizational process. Say that we are building a trip planner that combines the services of an airline, a land travel service, and a hotel. The process starts with a customer submitting an itinerary. The airline service kicks in first, followed by car rental and hotel (in parallel). Assume that the process uses a two-phase commit protocol to confirm a booking. Each service must support at least two sequences of two operations: the first sequence starts with an operation that makes a tentative reservation, followed by another operation that confirms the reservation. The second sequence starts with the operation that makes the reservation and ends with an operation that cancels it. If a given service does not support both sequences, then we have a problem. Figure 12 illustrates this problem.

[6] This suggests that when we write a generic inter-organizational process using BPEL4WS, we should use as many partners as the most general case requires: we let specific instantiations of the process merge partner definitions, if needed (e.g. same supplier supplying two parts, or the same department doing both invoicing and shipping).

Figure 12.Given an inter-organizational process specified in BPEL4WS, find appropriate partners.

Notwithstanding the signature matching aspects to this problem, this is a problem of matching what amounts to two partial traces: a partial trace of the inter-organizational process—where we don't care about the nature of the process steps that fall between calls to operations from the same potential partner—and a partial trace of the *internal process* to the potential partner[7]

3.4 Summary of composition problems

- Data equivalence problems (whether we are using high-level domain specific documents or not)
- Satisfiability of pre/post-conditions
- Operation coverage
- Partial trace equivalence

4 Specifying the pre/post-conditions of WSDL operations

The specification of explicit pre/post-conditions requires using an appropriate formal specification language. Although many such notations and languages exist (e.g., Z, VDM, Spec, OCL, etc.), for our purpose (description of WSDL-based web services), the interesting ones are those that could be easily integrated into existing WSDL descriptions, in the spirit of Meyer's Design by Contract.

In what follows, we describe two possible approaches to the specifications of pre/post-conditions in WSDL: Larch/WSDL and WSDL-Contract. But first, we briefly present an existing approach proposed by AI researchers, DAML-S.

4.1 DAML-S

DAML (DARPA Agent Markup Language) is an AI-inspired language, building on RDF and RDFS, for defining ontologies to be used in the development of the so-called *Semantic web*. Building on DAML, a group of researchers has developed a markup language to more specifically describe the semantic of Web services:

> "We present an approach to Web service markup that provides an agent-independent declarative API capturing the data and metadata associated with a service together with specifications of its properties and capabilities, the interface for its execution, and the prerequisites and consequences of its use." [McIlraith et al., 2001]

The proposed DAML-S ontology contains the following three classes of concepts:

[7] These are dual problems. Given the definition of an internal process to an organization, and the designation of some exposable (public) steps (operations) of the process, find the valid path expressions.

A) Service profile: concepts in this category describe "what the service does", that is, by describing its functionalities in terms of input and output types, effects (events caused by the execution of the service), etc. Logical properties are supposed to be used to described such conditions, although how exactly this will be expressed is not yet specified. A specific type of precondition, called an access condition, can also be specified; such a condition, which must be true for the service to be executed, must remain true once the service has been performed.

B) Service model: concepts in this category describe " "how the service works"; that is, it describes what happens when the service is carried out.". More specifically, a particular subclass of service model has been defined, namely, the process model, which in fact consists of two aspects: the *process model* itself—which describes the process behavior in terms of its composite and atomic actions, that is, à la BPEL4WS—and the *process control model*—which allows agents to monitor and control the execution of a process (this part has not yet been defined).

C) Service grounding: describes how the service can be accessed by an agent, for example, using a specific communication protocol.

Because of DAML-S roots in ontology languages, the syntax presented in the various papers describing it appears to be based on RDF. A number of elements of DAML-S appear not yet to be fully specified (at least from the examples presented in those papers), for instance, how exactly the preconditions and effects (post-conditions) would be described, what kind of logical language would be used, etc.

Probably because it predates the work on BPEL4WS, the DAML-S papers do refer to WSDL, but not to BPEL4WS. However, they do claim to inherit, for their process model description, from previous work on PSL (NIST's Process Specification Language) and various workflow languages.

4.2 Larch/WSDL

The Larch family of specification languages is based on a two-tier approach. The first tier is called the Larch Shared Language (LSL). Essentially, this is a library of (pure) abstract data types described using an algebraic approach—i.e., in terms of signatures (syntax) and recursive equations (semantics) relating the various operations—and which is *independent of any specific programming language*. This tier describes an ideal mathematical world consisting strictly of (immutable) values and functions on such values, where notions such as side-effects or exceptions do not exist. Clearly, this is not the world in which software artifacts live.

Software systems are instead described using a second-tier. It is at this level that software operations and their associated pre/post-conditions are described. Although such operation descriptions can refer to the types and functions provided by the language-independent LSL library, this second tier, because it aims at describing the real *interface* of software systems and software units, is *language specific*. Thus, different variants of Larch interface languages do exist—for example, Larch/Clu, LCL (Larch C Language), Larch/Ada, Larch/C++, etc.—, each tailored to the specific interactions mechanisms provided by the target language—for example, LCL does not provide for the specification of exceptions whereas Larch/C++ does.

Although a Larch interface language is specific to a target language, the operations' specifications clearly cannot be expressed in the target language syntax. Thus, an appropriate syntax must be developed for each interface language. Specifications written in this language can then be analyzed and some of their properties can be checked using the Larch Prover. Contrary to the approach described below, although the specifications could be included as comments in the target language based descriptions, there is no specific support for the dynamic evaluation of the pre/post-conditions, as suggested by Meyer's DBC approach and exemplified by the Eiffel language. Thus, the interface specification, although expressed with concepts specific to the target language, does not really get integrated into the associated implementations.

Defining a Larch/WSDL interface language could be a possible direction for integrating semantics into web services description. However, as mentioned earlier, the integration of the interface specification into the target language (WSDL specification in our case) is not necessarily natural or seamless. Furthermore, LSL is a rich and complex language, whose good use requires a lot of knowledge and expertise. Enriching this library is not easy either, because of the reliance of the algebraic specification method on recursive equations.

4.3 WSDL-Contract

The Eiffel language, developed by Meyer, popularized the use of pre/post-conditions for both the specification of operations' contracts and their *dynamic (run-time) verification*. More precisely, in Eiffel, appropriate mechanisms for the specification of pre/post-conditions and other assertions (including loop variants/invariants) have been integrated directly into the language definition. Such specifications can be interpreted logically, that is, as formal specification (and documentation) of the operations. In addition, these assertions can also be interpreted operationally, that is, as verifications that must be performed at run-time, to ensure the program is in a correct state relative to its intended specification.

Although Eiffel is, as of our knowledge, the only language to directly integrate such formal specification and verification of contracts into the language definition, other strategies have been proposed to obtain an equivalent effect—formal documentation and specification together with run-time verification of assertions—in other languages. The most common approach, now available for a wide variety of languages (for example, Java, C, Python, etc.), is the use of special comments together with a pre-processor (à la JavaDoc). The fact that the pre/post-conditions are specified using comments mean that the source program can be handled normally by the regular compiler. On the other hand, if required by the user, the pre-processor can also be used to analyze those special comments and generate, in the executable code, appropriate run-time checks.

Java is one language for which such pre-processors have been developed, e.g., iContract. In this case, the syntax of the special comments is based on the JavaDoc style comments, whereas the syntax of the specification expressions is inspired from OCL. The following is a simple example of an operation to add an element into a collection:

```
/**   Add an element o to collection c.

      @pre !c.contains(o)
      @post c.size() = c@pre.size()+1
      @post c.contains(o)
*/
public void addElement( Collection c, Object o ){ … }
```

Using this strategy for integrating pre/post-conditions into WSDL descriptions would have a number of advantages. First of all, such an approach seems more *lightweight* than a Larch based approach, as it does not need to rest on the understanding of a complex library (LSL) of algebraic types; instead, knowledge of the key collection operations, à la OCL, is generally sufficient. Also, such an approach to the integration of contracts is becoming fairly common, as it is now available for use with many programming languages. Finally, such an approach would also allow for the integration of dynamic verification of assertions associated with Web services operations, an interesting way to verify the behavior of complex business processes

In terms of concrete syntax, how this integration would be done still remains to be defined. If we refer to the new WSDL (2.0) specification, the feature/property mechanism will probably be used to specify special properties associated with contract specification (using an XML-based syntax). .How to develop a dynamic contract evaluation engine, in order for the various contract conditions to be verified dynamically still remains to be examined, so that appropriate assertion violation faults are defined and get generated when pre/post-conditions fail to hold.

5 Validating process compositions

One possible perspective from which to examine the validation of process composition might be termed the orchestration/choreography perspective. Orchestration refers "to an executable process that can interact with both internal and external Web services [where] interactions occur at the message level" [Peltz, 2003]. Orchestration always represents a specific party's perspective. By contrast, choreography "is more collaborative and allows each involved party to describe its part in the interaction" [Peltz, 2003]. Choreography thus tracks the various messages exchanged among different parties. Given a specific executable process, it would be useful to validate that behavior exhibited by this process indeed conforms to the overall choreography in which the process participates.

In a similar way, abstract BPEL4WS processes can be used to represent abstract business protocols. Such protocols, as illustrated in the example presented in Section 16.1 of the BPEL4WS document [Andrews et al., 2003], can play roles similar to those of WSCI interfaces, namely, described the order in which the various operations of a Web service should be invoked. Given an executable process that uses that Web service, validating its behavior relative to the Web service abstract business protocol would involve ensuring that the operations it uses indeed obey the protocol.

More precisely, using Web services to integrate various business applications and processes requires being able to describe long sequence of interactions between partners, that is, requires defining appropriate *business protocols*—"formal description of the message exchange protocols used by business processes in their interactions" [Andrews et al., 2003]—that is, specifying a business protocol aims at describing "the observable behavior of a service and the rules for interacting with the service from the outside […] [such] that external actors will know at each stage in the given process which messages that service may or must send or receive next" from [WSCI 1.0, 2002].

One possible way to describe protocols is by using path expressions, which were introduced to express synchronization of concurrent processes [Campbell and Habermas, 1974, Andler 1979]. A path expression can be used to describe, and restrict, the allowable sequences of operations on an entity, thus can be interpreted as a specification of the allowable "transactions" associated with a system.

More precisely, path expressions are a kind of *regular expressions* where basic path expressions have the following form [Andler, 1979]—parenthesis can be used, as necessary, to create sub-expressions:

- `A;B`: `A` can be executed followed by `B`.
- `A+B`: either `A` or `B` can be executed.
- `A*`: 0, 1 or more copies of `A` can be executed consecutively.

For example, the following regular expression denotes a protocol for a service that must first be initialized with the `init` operation, then must be followed by any number (possibly 0) of calls to `inc` or `dec` operations (in any order), and then must be concluded by a call to the `demandVal` operation:

```
  init; (inc + dec)*; demandVal
```

Note that regular expressions in various forms can also be found in other languages for specifying properties of processes' behavior. For instance, it is possible to extend modal and temporal logic operators with such regular expressions. This is done, for example, in the *regular alternation-free mu-calculus* [MateescuSig00], where various useful properties of processes can be expressed quite intuitively and concisely using those regular expressions, instead of using the more complex fixed point operators. Of course, since a reactive system generally exhibits a non terminating behavior, regular expressions must be generalized to omega-regular expressions that can deal with infinite sequences of computation.

Given a protocol description expressed as an appropriate path or (omega-)regular expression, we are interested in checking whether a specific executable business process that uses this web service do obey the indicated protocol. One possible line of research would be to develop a tool that would perform such an analysis. This tool could be built around the CADP toolbox (Construction and Analysis of Distributed Processes, formerly known as "`CAESAR/ALDEBARAN Development Package`" [Fernandez et al., 1996]).

More precisely, the goal would be to analyze an executable process description written in BPEL4WS and check whether the process do conform to the protocol of the various web services used by the process. This tool could work as follows:

- Using an appropriate tool, a business processes *P* expressed in BPEL4WS would be parsed and analyzed and the various Web services used by *P* would be identified.
- For each key service, the appropriate (slice of) flow of control within *P* would be analyzed and an algebraic process model representing this flow would be generated---LOTOS, one of the key language supported by the CADP toolbox, could be used as a possible target algebraic process model.
- The path expression describing the protocol of a service would be translated into an appropriate regular alternation-free mu-calculus expression, one of the logic notation supported by the CAPD toolbox.
- Conformance of the abstract model of *P* with the service's protocol would then be verified using the evaluator model-checking tool of CADP.

6 Verifying properties of business processes

Associating a formal semantic to web services and having business processes defined in terms of an executable language, thus also providing a formal semantics, has a number of interesting implications. Among them is the possibility of verifying various properties of business processes. In what follows, we present some work that has been done in this regard.

6.1 Model checking of business processes (pre-web services)

Janssen et al. [JanssenEtAl98] present an application of the SPIN model-checker in Testbed, a framework for business process reengineering. Business processes are described in AMBER, a graphical process description language. The AMBER graph is automatically translated in PROMELA, SPIN's input process language. Properties are expressed using LTL formula. Two kinds of temporal properties can be expressed:

a) Behavioral properties, which concern the execution of actions in a process description.
b) Data-based properties, which concern the evolution of the data variables defined in a process description.

The translation of the behavior description from the Amber process language to PROMELA requires defining an appropriate *state automaton* where all nodes in the Amber model (based on a causality semantics) correspond to transitions in PROMELA (based on an interleaving semantics).

The main limitations of this work, as described by the authors themselves, are the followings:

- Only a restricted subset of PROMELA had to be used in order to make the model tractable.
- Complex data structures lead to high memory requirements.
- Linear time logic is sometimes restrictive.
- Whereas SPIN supports only weak fairness, AMBER assumes strong fairness.

Wang et al. [WangEtAl00] present an application of model checking to a ticket sales system first implemented in C, first using VeriSoft, then with PROMELA/SPIN. The verification using VeriSoft was done using assertions (VS_assert()) placed in various places in the C code. The key advantage of this tool is that it makes it possible to verify existing programs with only minor modifications. Its key drawbacks, however, is that it does not record the states it has visited and it does not perform a complete exploration of the state space (only up to a certain depth specified at run-time). This stems from the use of C as a process description language, which may lead to large and complex state spaces. To avoid these drawbacks, the C code was then translated into PROMELA, a relatively easy task because PROMELA is also a C-based language. Standard model checking was then applied.

To make verification possible, a number of simplifications had to be made: the system consisted of only two agents and two costumers, there was only one ticket for sale, etc. Still, various bugs were detected, both using VeriSoft and SPIN, including detection of a deadlock situation using the latter tool.

6.2 Model-checking of Web services

A number of recent papers address directly the verification of processes involving Web services.

Nakajima [Nakajima02] applies model-checking to web services described in WSFL (Web Services Flow Language), a net-oriented specification language based on a workflow description language. It is further stated that "WSFL is meant to be a behavioral extension of WSDL, and is equipped with an XML-based concrete syntax", but the relationships with WSDL is not further explained. The verification is performed by translating WSFL descriptions into PROMELA/SPIN, where properties to be checked are expressed in LTL.

One of the key problem mentioned in the paper appears to be that WSFL's operational semantics does not correctly handle some types of dataflow, which can be corrected using an appropriate form of dead path elimination (DPE); this seems to have been taken care in BPEL4WS, since DPE is explicitly mentioned. The conclusion of the paper mentions the emergence of BPEL4WS as the convergence between WSFL and XLANG.

The starting point of the work by Narayanan and McIlraith [NarayananMcI02] is the DAML-S ontology for Web services, which "provide[s] semantic markup of the content and capabilities of Web services". A formal semantics for DAML-S is first provided, most precisely to define the semantics of atomic and composite processes introduced in the DAML-S ontology. This semantics is defined by mapping DAML-S into PSL (Process Specification Language), "a process specification ontology described in the situation calculus, a (mostly) first-order logical language for reasoning about dynamical processes". The situation calculus semantics expressed in PSL is then translated into Petri nets. This representation using Petri nets has been chosen because Petri nets provide an executable semantics and also offer techniques for quantitative analysis; furthermore, mappings between PSL and Petri nets are already available.

Three verification problems are addressed by their approach using Petri nets: reachability, liveness, and deadlock detection. Theoretical complexity of these various tasks are briefly discussed. General model-checking of temporal properties is not addressed.

Foster et al. [FosterEtAl03] describe an approach to the model-based verification of Web service compositions using BPEL4WS. More precisely, LTSA-MSC (Labelled Transition System Analyzer extended with Message Sequence Chart) is first used to describe the intended behavior of the workflow using scenario-based specifications, which are then compiled into FSP (Finite State Processes), a textual notation for process calculus. A BPEL4WS implementation is then written and is also translated into FSP. Verifying that the implementation satisfies the specification is then done through trace equivalence analysis of the resulting FSP representations (which appears to be similar to a form of bisimulation checking). The types of properties that can be checked using this approach are not clearly explained.

Rather than studying BPEL4WS in full, Koshkina and van Breugel [KoshkinaBre03] introduce the BPE-calculus, which targets the essence of BPEL4WS focusing on the flow of control, but abstracting from data and ignoring fault and compensation handlers. The formal syntax and (structural operational) semantics of the BPE-calculus are defined using the notation supported by PAC (Process Algebra Compiler), which from a formal syntax and semantics generates a front-end (written in SML) that interfaces with CWB (Concurrency WorkBench). Model-checking is then performed using CWB, where properties to be checked are expressed in the mu-calculus (branching time temporal logic) extended with some CTL (Computation Tree Logic) operators (A, E, G and F). CWB also supports equivalence checking and preorder checking.

The main advantage claimed for this approach is that, contrary to the previous ones, a business process does not have to be translated into a generic process language, which generally results in a loss of abstraction when counter-examples are found while performing model-checking. Of course, the main future work mentioned in the conclusion is to include fault and compensation handlers, as well as include the handling of time – handling of data, however, is not mentioned.

7 Conclusion

In this technical report, we have presented some proposals toward adding semantic information to Web service descriptions, including specification of contracts and protocols. We have also shown how these specifications could be used for verification purposes. In addition, we have examined the issue of composing existing services together in order to satisfy some business need.

The key goal of this technical report was, admittedly, to identify a number of possible interesting research directions. Thus, what we have been presenting is still work in progress, so that no result are available yet.

However, we do think that the questions raised in this report—how to add semantics information into web service descriptions—are important, especially since a number of standards or proposals for describing Web services have recently been, and still are, emerging. Understanding the strengths, and limits, of these proposals is important. Furthermore, we think that the development of Web services should also benefit from the same kind of conceptual tools, such as contracts, now available in other areas of software development.

8 Bibliography

[v.d. Aalst, 1999] W.M.P. van der Aalst, “Formalization and Verification of Event-driven Process Chains,” *Information and Software technology*, vol. 41 no. 10, 1999, pp. 639-650.

[Altschul et al., 1997] Altschul, S., Madden., T., Scha_er, A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D., Gapped BLAST and PSI-BLAST: a new generation of protein database search programs, *Nucleic Acids Research*, 25(17):3389-3402, 1997.

[Amir & Keselman, 1997] Amir, A. and Keselman, D., Maximum agreement subtree in a set of evolutionary trees: metrics and efficient algorithms, *SIAM Journal of Computing*, 26(6):1656-1669, 1997.

[Andrews et al., 2003] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Aivana Trickovic, Sanjiva Weerawarana, Business Process Execution Language for Web Services (BPEL4WS) version 1.1 , 5 May 2003.

[Andler, 1979] S. Andler, “Predicate path expressions,” in Annual Symposium on Principles of Programming Languages, 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, Association for Computing Machinery. San Antonio, Texas: ACM Press, 1979, pp. 226–236.

[Aversano et al., 2004] L. Aversano, G. Canfora, and A. Ciampi, An Algorithm for Web Service Discovery through their Composition, *Proceedings of the 2004 International Conference on Web Services*, San Diego, July 2004, 332-341.

[Bolognesi & Brinksam, 1987] T. Bolognesi and E. Brinksma. “Introduction to the ISO specification language LOTOS”. *Computer Networks and ISDN Systems*, vol. 14, pp. 25-59, 1987.

[BPMI,2003] *Business Process Modeling Language*, Business Process Management Institute, January 24, 2003, 96 pages

[Brooks, 1987] Brooks, F. "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer* 20, 4 (April 1987): 10-19

[CampbellHab74] R. Campbell and A. Habermann, “The specification of process synchronization by path expressions,” in *Operating Systems*, E. Gelenbe and C. Kaiser, Eds. Springer-Verlag, LNCS-16, 1974, pp. 89–102.

[Carlson, 1979] W.M. Carlson, "Business Information Analysis and Integration Technique (BIAIT) - the new horizon." Data Base, 10 (4), p. 3-9, Spring 1979.

[Coad & Yourdon, 1988] P. Coad & E. Yourdon, *Object-Oriented Analysis*, Prentice-Hall, First edition, 1989.

[Constantinescu et al., 2004] I. Constantinescu, B. Faltings, and W. Binder, Large Scale, Type-Compatible Service Composition, in *Proceedings of the 2004 International Conference on Web Services*, San Diego, July 2004, 506-513.

[Curtis et al., 1992] Curtis, B., Kellner, M.I. and Over, J.: “Process modeling”, *Communications ACM*, 35, (9), pp. 75- 90, 1992

[DAML, 2002] DAML Services Coalition. DAML-:Web Service Description for the Semantic Web. In The First International Semantic Web Conference (ISWC), June 2002.

[DAML+OIL,2001] I. Horrocks, F. van Harmelen, P. Patel-Schneider, T. Berners-Lee, D. Brickley, D. Connoly, M. Dean, S. Decker, D. Fensel, P. Hayes, J. Hein, J. Hendler, O. Lassila, D. McGuinness, and L. A. Stein. DAML+OIL, 2001. http://www.daml.org/2001/03/daml+oil-index.html

[Dayal et al., 2001] U. Dayal, M. Hsu, and R. Ladin, “Business Process Coordination: State of the Art, Trends and Open Issues,” in *Proceedings of the 27th Very Large Databases Conference*, VLDB 2001, Roma, Italy, pp. 3-13.

[Dogac et al., 2002] Asuman Dogac, Yusuf Tambag, Pinar Pembecioglu, Sait Pektas, Gocke Laleci, Gokhan Kurt, Serkan Toprak, and Yildiray Kabak, “An ebXML Infrastructure Implementation through UDDI Registries and RosettaNet PIPs, in *proceedings of ACM SIGMOD 2002*, June 4-6, Madison-Wisconsin, pp. 512-523.

[Dussart et al., 2002] Aymeric Dussart, Benoit Aubert & Michel Patry, "An Evaluation of Inter-Organizational Workflow Modelling Formalisms," technical report 2002s-64, CIRANO (http://www.cirano.qc.ca) , July 2002.

[ebXML,2003] see http://www.ebxml.org

[Edmonds & Matula, 1968] Edmonds, J. and Matula, D., An algorithm for subtree identification, SIAM Rev., 10:273-274, 1968.

[Eertink et al., 1999] H. Eertink, W. Janssen, P.A. Luttighuis, W. Teeuw, and C. Vissers. "A business process design language". In FM'99 --- Formal Methods: World Congress on Formal Methods in the Development of Computing Systems, pp. 76--95. Springer-Verlag, LNCS-1119, 1999.

[Fernandez et al., 1996] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu. "CADP: A protocol validation and verification toolbox". In Computer Aided Verification, pp. 437--440. Springer-Verlag, LNCS-1102, 1996.

[Finden & Gordon, 1985] Finden, C.R. and Gordon, A.D., Obtaining common pruned trees, Journal of Classification, 2:255-276, 1985.

[FosterEtAl03] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based verification of web service compositions," in *18th IEEE International Conference on Automated Software Engineering*. Montreal, QC, Canada: IEEE Computer Society Press, 2003, pp. 152–163.

[Gale & Eldred, 1996] Thornton Gale and James Eldred, *Getting Results with the Object-Oriented Enterprise Model*, SIGS Boooks, ISBN 1-884842-16-X, January 1996.

[Garey & Johnson, 1979] Garey, M., and Johnson, D., *Computers and Intractability*, Freeman, San Francisco, 1979.

[Geerts & McCarthy, 1997] Geerts, Guido L. & William E. McCarthy, "Modeling Business Enterprises as Value-Added Process Hierarchies with Resource-Event- Agent Object Templates," in *Business Object Design and Implementation* J. Sutherland and D. Patel (eds.), 1997, Springer-Verlag, pp. 94-113

[Glikas & Valiris, 1999] Michalis Glykas and George Valiris, "Formal methods in object-oriented business modelling," *The Journal of Systems and Software*, vol. 48 (1999), pp. 27-41.

[Gruhn & Wellen, 2001] Volker Gruhn and Ursula Wellen, "Analyzing a process landscape by simulation," *The Journal of Systems and Software*, vol. 59 (2001), pp. 333-342.

[Hall, 1993] Robert J. Hall, ''Generalized Behavior-based Retrieval,'' in *Proceedings of the 15th International Conference on Software Engineering*, pp. 371-380, ACM Press, Baltimore, Maryland, May 17-21, 1993.

[Hammer, 1990] M. Hammer, "Re-engineering work: don't automate, obliterate", *Harvard Business Review*, July-August 1990, pp. 104-112.

[Hammer & Champy, 1993] Hammer M., and Champy, J., *Reengineering the Corporation*, Harper Business, 1993, New York.

[Haughen & McCarthy,2000] Haugen, R & McCarthy W.E., "REA, a semantic model for Internet supply chain collaboration", OOPSLA Workshop on *Business Object Components: Enterprise Application Integration*, OOPSLA 2000, Minneapolis, Minnesota.

[Henderson-Sellers & Edwards, 1990] Brian Henderson-Sellers and Julian M. Edwards, "The Object-Oriented Systems Life Cycle," *Communications of the ACM*, vol. 33, no. 9, September 1990, pp. *143*-159

[Horowitz & Sahni, 1978] Horowitz, E. and Sahni, S., *Fundamentals of Computer Algorithms,* Computer Science Press, 1978.

[Ijiri, 1975] Ijiri, Y., *The Theory of Accounting Measurement*, American Accounting Association, Sarasota, Florida, 1975.

[Isoda, 2001] Sadahiro Isoda, "Object-Oriented real-world modeling revisited," *Journal of Systems and Software*, vol. 59 (2001), pp. 153-162.

[Jackson & Twaddle, 1997] Michael Jackson and Graham Twadlle, *Business Process Implementation: Building Workflow Systems*, Addison Wesley, 1997, ISBN 0-201-177684.

[Janssen et al., 1998] W. Janssen, R. Mateescu, S. Mauw, and J. Springintveld. "Verifying business processes using SPIN". In G. Holzman, E. Najm, and A. Serhrouchni, editors, *Proceedings of the 4th International SPIN Workshop*, pp. 21--36, Paris, France, 1998.

[Jiang et al., 1995] Jiang, T., Wang, L., and Zhang, K., Alignment of trees - an alternative to tree edit, *Theoretical Computer Science*, 143:137-148, 1995.

[Jones86] C. Jones, *Systematic Software Development using VDM*. Prentice-Hall, 1986.

[Keller et al., 1992] Keller, G; Nüttgens, M, Scheer, A.-W, "Semantische Prozebmodellierung auf der Grundlage, publication of the Institut für Wirtschaftsinformatik, paper 89, Saarbrucken, 1992.

[Kindler, 2003] Kindler, E., "On the semantics of EPCs: A framework for resolving the vicious circle" technical report, Reihe Informatik, University of Paderborn, Paderborn, Germany, August 2003.

[KoshkinaBre03] M. Koshkina and F. van Breugel, "Verification of business processes for web services," Deparment of Computer Science, York University, Toronto, Ontario, Tech. Rep. CS-2003-11, 2003.

[Lee et al., 1996] J.Lee, M.Gruninger,Y.Jin, T.Malone, A.Tate, G. Yost, and other members of the PIF Working Group, *The PIF Process Interchange Format and Framework (May 24,1996)*,1996, http://ccs.mit.edu/pif8.html.

[Lin et al., 2001] Lin, G.H., Ma, B., and Zhang, K., Edit distance between two RNA structures, *Proceedings of the Fifth Annual International Conference on Computational Biology*, ACM Press, 211-220, 2001.

[McIlraithEtAl01] S. McIlraith and H. Z. T. C. Son, "Semantic web services," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 46–53, March 2001.

[McCarthy, 1980] William E. McCarthy, "Construction and Use of Integrated Accounting Systems with Entity-Relationship Modeling", in P. Chen eds., *Entity-Relationship Approach to Systems Analysis and Design*, North Holland Publishing, 1980, pp. 625-637.

[McCarthy, 1982] William E. McCarthy. "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment," *The Accounting Review* (July 1982) pp. 554-78

[Malone et al.,1999] T.W. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C.S. Osborn, A. Bernstein, G. Herman, M. Klein, and E.O'Donnell, "Tools for inventing organizations: Toward a handbook of organizational processes," *Management Science* 45(3) pp 425-443, March, 1999.

[MateescuSig00] R. Mateescu and M. Sighireanu, "Efficient on-the-fly model-checking for regular alternation-free mu-calculus," INRIA, Rocquencourt, Tech. Rep. 2899, Mars 2000.

[Mayer et al., 1995] Mayer, R.J., C. P. Menzel, M.K. Painter, P.S. deWitte, T. Blinn, and B. Parakath, *Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report*, Knowledge Based Systems Inc., September 1995.

[MendlingNueNut04] J. Mendling, G. Neumann, and M. Nuttgens, "A comparison of XML interchange formats for business process modelling," in *Proc. of EMISA 2004 "Informationssysteme im E-Business und E-Government"*, ser. Lecture Notes in Informatics (LNI), F. Feltz, A. Oberweis, and B. Otjacques, Eds., vol. 56, Oct. 2004, pp. 129–140.

[Meyer92] B. Meyer, "Applying "design by contract"," *IEEE Computer*, vol. 25, no. 10, pp. 40–51, 1992.

[Mili et al., 1994] Hafedh Mili, Odile Marcotte, and Anas Kabbaj, ''Intelligent Component Retrieval for Software Reuse,'' in *Proceedings of the Third Maghrebian Conference on Artificial Intelligence and Software Engineering*, pp. 101-114, Rabat, Morocco, April 11-14, 1994.

[Nakajima02] S. Nakajima, "Model-checking verification for reliable web service," in *OOPSLA Workshop on Object-Oriented Web Services*, 2002.

[NarayananMcI02] S. Narayanan and McIlraith, S. A., "Simulation, verification and automated composition of web services," in *Proceedings of the eleventh international conference on World Wide Web*. Honolulu, Hawaii, USA: ACM Press, 2002, pp. 77–88.

[NIST,2002] *The Process Specification Language* (PSL 2.0), September 2002, NIST, http://ats.nist.gov/psl/

[OASIS, 2001] Business Process and Business Information Analysis Overview v1.0 (ebXML), May 11, 2001, http://www.ebxml.org/specs/bpOVER.pdf

[OMG, 2001] OMG, *EDOC: UML Profile for Enterprise Distributed Object Computin*g, Document ptc/2001-12-04, December 2001

[Ould & Roberts, 1987] M.A. Ould & C. Roberts, "Defining Formal Models of the Software Development Process," in *Software Engineering Environments*, ed P. Brereton, Ellis Horwood, 1987.

[Ould, 1995] M.A. Ould, *Business Processes: Modelling and Analysis for Re-engineering and Improvement*, 1995, John Wiley & Sons.

[OWL, 2002] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. Web Ontology Language (OWL) Reference Version 1.0. W3C Working Draft 12 November 2002, http://www.w3.org/TR/2002/WD-owl-ref-20021112/

[Peltz, 2003] C. Peltz, "Web Services Orchestration and Choreography," *IEEE Computer*, vol. 36, no. 10, pp. 46-52, October 2003.

[Peterson, 1981] J. L. Peterson, *Petri Nets Theory and the Modeling of Systems*, Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

[Petrie et al., 2004] Charles Petrie et al.: "Adding AI to Web Services", in *Agent Mediated Knowledge Management,* LNAI 2926, Springer-Verlag, 2004.

[Phalp, 1998] Keith Phalp, "The CAP framework for business process modeling," *Information and Software Technology*, vol. 40 (1998) , pp. 731-744.

[Phalp & Shepperd,2000] Keith Phalp and Martin Shepperd, "Quantitative analysis of static models of processes," *The Journal of Systems and Software*, vol. 52, pp. 105-112, 2000.

[Prieto-Diaz & Freeman, 1987], Ruben Prieto-Diaz and Peter Freeman, ''Classifying Software for Reusability,'' *IEEE Software*, pp. 6-16, January 1987.

[Reenskaug, 1996] Trygve Reenskaug, *Working with Objects: The OORAM Software Engineering Methodology*, Manning, 1996.

[RosettaNet,2003] http://www.rosettanet.org

[Ross,1977] Ross, D., and K. Schoman, "Structured Analysis for Requirements Definition," *IEEE Trans. Software Engineering*, vol. 3, no. 1, January 1977.

[Salton & McGill, 1983] Salton G., and McGill, M, *Introduction to Modern Information Retrieval*, 1983.

[Sycara et al., 2003] Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N.: Automated Discovery, Interaction and Composition of Semantic Web Services, *Journal of Web Semantics*, Volume 1, Issue 1, December 2003

[Tai, 1979] Tai, K., The tree-to-tree correction problem, *Journal of the ACM*, 26:422-433, 1979.

[Turban et al., 1999] E. Turban, J. Lee, D. King, et H.M. Chung, *Electronic Commerce: A Managerial Perspective*, Prentice-Hall, 1999.

[WS-CDL, 2004] N. Kavantzas, D. Burdett, and G. Ritzinger, "Web services choreography description language version 1.0," April 2004, http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427.

[WSCI, 2002] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic, and S. Zimek, "Web service choreography interface (WSCI) 1.0," August 2002, http://www.w3.org/TR/wsci.

[WSDL,2001] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[WSDL,2004] R. Chinnici, M. Gudgin, J.-J. Moreau, C. Schlimmer, and S. Weerawarana, "Web services description language (WSDL) version 2.0 part 1: Core language," August 2004, http://www.w3.org/TR/2004/WD-wsdl20-20040803.

[WFMC,1995] Workflow Management Coalition, *The Workflow Reference Model*, document no TC00-1003, January 1995, see http://www.wfmc.org

[Wu et al., 2003] Wu, D., Sirin, E., Hendler, J., Nau, D., and Parsia, B. 2003. Automatic Web Services Composition Using SHOP2. Twelfth World Wide Web Conference.

[Yourdon,1989] Yourdon, E. N., *Modern Structured Analysis*, Prentice-Hall, 1990.

[Yu, 1976] Yu, S. C., *The Structure of Accounting Theory*, The University Presses of Florida, 1976.

[Zaremski & Wing, 1993] Amy Moormann Zaremski and Jeannette M. Wing, ‘‘Signature Matching: A Key to Reuse,’’ *Software Engineering Notes*, vol. 18, no. 5, pp. 182-190, 1993. First ACM SIGSOFT Symposium on the Foundations of Software Engineering.