BPEL4WS Document Management with an Active Deductive XML Database

[1]Jose Oscar Olmedo-Aguirre, [1]Giner Alor-Hernandez, [2]Juan Miguel Gomez

Research and Advanced Studies Center of IPN (CINVESTAV).
Electrical Engineering Department. Computing Section.
Mexico City, Mexico.
[2]Digital Enterprise Research Institute (DERI). National University of Ireland, Galway
Galway, Ireland.
e-mail: oolmedo@delta.cs.cinvestav.mx, gineralor@computacion.cs.cinvestav.mx,
juan.gomez@deri.org

Abstract. BPEL4WS (Business Process Execution Language for Web Services) is a process modeling language for representing compositional workflow structures that coordinate elementary Web service invocations. Nonetheless, a major problem in BPEL4WS modeling is that the language does not support generic definition of workflows. The importance of generic workflow definitions relies on describing increasingly complex forms of recurring situations abstracted from the various stages of business collaboration. In this paper, the problem of defining and instantiating generic BPEL4WS workflows is addressed by using ADM, an active deductive model for XML database management. The ADM management system extends the BPEL4WS workflow engine with an inference engine to develop a rational behavior hidden in the Web services provided. Among the salient contributions of this work, the brokering architecture developed so far comprises a business agency, a repository of workflow patterns of interaction describing common business practices that can be instantiated with appropriate business partners.

1 Introduction

Web services are emerging as a suitable approach to support business collaboration. Web services are standardized mechanisms for integrating applications based on an open programming interface capable of being described and discovered as XML documents. Once deployed, Web services can be invoked by application programs or other Web services leading to the definition of composite Web services. Compositional languages and models provide the means to describe workflow structures to coordinate Web service invocations. An important compositional language is Business Process Execution Language for Web Services (BPEL4WS) that orchestrates diverse applications to conduct a business process [2]. As other compositional languages, BPEL4WS provides the means to specify the order in which the services are to be combined as indicated by a composition schema. Therefore the schema can be seen as a kind of program written in a programming language. Nonetheless,

BPEL4WS cannot be used as generic business process description language because it does support the notion of procedures and parameters.
In this paper, the problem of managing generic BPML process descriptions is addressed by using ADM, an active deductive XML database model for managing XML databases that are augmented with rational and reactive behavior. By extending XML with logical variables and logical procedures, the ADM programming model is not only able to represent generic BPEL4WS definitions but also it provides the means to instantiate and enact process descriptions either by submitting the process instance to a BPEL4WS compositional engine or by interpreting the BPEL4WS language constructs according to the ADM rule semantics. The support of Web service composition grounds on the fact that the composition engine can be considered as a kind of reactive system. ADM reacts to messages received from clients or other services as it progresses through the coordination schema, possibly undertaking some actions. Since rules are always selectable for execution, they can model the start-up of activities signaled by the occurrence of message operation events during the enacment of a business process. Exception-handling, a domain that characterizes by its asynchronous and reactive nature, can be conveniently managed by rule-based systems. Though rule-based systems do not impose any order in selecting rules, in the ADM model, sequential and parallel constructs are proposed to restrict the valid sequences of actions.

2 BPEL4WS Compositional Language

BPEL4WS builds on Microsoft's XLANG (Web Services for Business Process Design) [3] and IBM's WSFL (Web Services Flow Language) [4] combining block structured language constructs borrowed from XLANG with a graph-oriented notation originated from WSDL. BPEL4WS closely follows the WS/Coordination and WS/Transaction specifications. The former describes how Web services may use predefined coordination contexts to be associated to a particular role in a collaboration activity [5]. The latter provides an infrastructure that provides transaction semantics to the coordinated activities [6]. A BPEL4WS document consists of three parts describing data, coordination activities and communication activities [7]. Data tags are used to define a set of external partners and the state of the workflow. Coordination activity tags define the process behavior by means of conventional control flow structures. Finally, communication activity tags define communication with other Web services through coordination activities by sending and receiving information.

3 Management of BPEL Documents

Orchestration of Web services using BPEL4WS consists of building at design time a fully instantiated workflow description where connections to external Web services are defined to be executed later on. In this schema, workflows are completely determined since business partners are known before hand.

3.1. Defining generic workflow definitions

Each template describes generic commercial activities. As an example, if purchasing a book, the client might be interested in buying the book in the store that offers either the lowest price or the minimum delivery time. If the client wants to buy several books at the lowest price, BPM will retrieve from a database the location of the BPEL4WS workflow template that uses the purchase-criteria selected. Once the template is located, ADM uses some related configuration files in order to instantiate them. ADM organizes a collection of XML documents in extensional and intentional databases. Extensional databases are collections of standard XML documents that does not require of additional mark-up. Intentional databases are collections of logical procedures and ECA rules called programs. In order to introduce generic BPEL4WS definitions, ADM extends XML elements with logical variables. Logical variables are used to define the integrity constraints imposed on the document contents and to formulate queries to bind values to variables. Logical variables can be of either type string (prefixed by a '$') or term (prefixed by a '#') and they may occur in elements and in attributes. Integrity constraints are introduced by means of logical procedures. Fig. 1 shows an example of a generic BPEL4WS definition that includes logical variables.

3.2 Instantiating the workflow

We have developed a BPM (Business Processes Manager) which obtains the templates that can be used to find the suppliers that offer the product required by the client. A query to a database containing the WSDL documents provided by BPM can retrieve the appropriate Web services to obtain price, delivery time, quantity, and purchase access point of the product. An instance of the generic BPEL4WS document is obtained by substituting each logical variable by its bound value that appears next to the variable in Fig. 1. The logical variable $PWSDL is used to place pieces of information related to the WSDL for the Web service provided whose behavior is given by this template. If variable $PWSDL is bound to string value "http://www.cinvestav.com/getPriceandDeliveryTime.wsdl", after completion of full instantiation of this document, the value will appear instead of logical variable $PWSDL. These services are getPriceandDeliveryTime, getProviderQuantity, and getProviderURLBuy, respectively, that are based on UNSPSC and RosettaNet ontologies.

3.3 Enacting the workflow

After the workflow has been fully instantiated, the workflow can be executed. The instantiated templates are allocated in a BPEL4WS engine for execution. To communicate with the running workflow, BPM builds SOAP messages containing the information provided by the client.

```
<process name="lowpriceprocess" ...
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns: $PN pn0= $PWSDL "http://www.cinvestav.com/getPriceandDeliveryTime.wsdl"
```

```
xmlns:tns="http://www.cinvestav.com/getLowPrice.wsdl"/>
<partnerLinks>
<partnerLink name="User" partnerLinkType="tns:UserSLT"
partnerRole="InformationProvider"/>
<partnerLink name="partner0LT" partnerLinkType="tns:partner0PLT"
partnerRole="partner0Provider"/>
...
</partnerLinks>
<variables>
<variable name= $UReq "uRequest" messageType="tns:getLowPriceRequest"/>
<variable name= $URes "uResponse" messageType="tns:getLowPriceResponse"/>
<variable name= $PReq "partner0Request" messageType="pn0:getArrayRequest"/>
<variable name= $PRes "partner0Response" messageType="pn0:getArrayResponse"/>
...
</variables>
<sequence name="sequence"/>
<receive name="receiveRequest" operation="getLowPrice"
partnerLink="User" portType="tns:getLowPricePortType"
createInstance="yes" variable= $UReq "uRequest"/>
<assign name="CreateLowPriceInput">
<copy>
<from variable= $UReq "uRequest"/>
<to variable= $PReq "partner0Request"/>
</copy>
...
</assign>
<invoke name="invokeService" inputVariable= $PReq "partner0Request"
operation="getArray" outputVariable= $PRes "partner0Response"
partnerLink="partner0LT" portType="pn0:getArrayPortType"/>
...
<reply name="sendInformation" operation="getLowPrice"
partnerLink="User" portType="tns:getLowPricePortType"
variable= $URes "uResponse"/>
</sequence>
</process>
```

Fig. 1 In a generic BPEL workflow definition, the document contains logical variables that appear prefixed by a dollar sign. In an instance of the workflow, the document contains only the values bound to the variables (shown here next to the variables).

Once the workflow has been successfully terminated, it sends back to the client the list of suppliers satisfying client's conditions. Then, the workflow is deallocated from the workflow engine. After the client selects the suppliers, a BPWL4WS template for placing a purchase order is now retrieved from the repository, completed and executed as described before.

By enacting this workflow the purchase orders are sent to the suppliers and the corresponding answers from each supplier are eventually received. This example has shown that ADM provides the means not only for defining generic BPEL4WS descriptions but also for producing instances and send the instance for execution to the BPEL4WS workflow engine. In that follows, we show that ADM can also interpret the BPEL4WS language constructs if the basic model described in [9] is extended with basic communication operations.

4 Operational Semantics

4.1. Basic Definitions

In this section a few basic definitions for XML terms are given to formally introduce the operational semantics of ADM. The XML term <a a_1 = T1 ... $a_m = T_m$ > ... </a> is *normalized* if its list of attribute-value pairs is lexicographically ordered by the name of the attribute in increasing order: $a_i \leq a_j$ if $i \leq j$, $\forall\, i, j \in \{1, \ldots, m\}$. The set of *substitutions* Σ consists of the partial functions XMLVariable $\rightarrow$ XMLTerm. Computed answers to queries are obtained by the *composition* $\sigma\sigma'$ of substitutions σ and σ'. The *null substitution* ε is the identity for substitution composition $\varepsilon\sigma = \sigma\varepsilon = \sigma$. A *ground substitution* does not introduce any variable. The natural extension σ:: XMLTerm $\rightarrow$ XMLTerm of a substitution defined over variables to XML terms is denoted by the same name. The instance of an XML term x under substitution σ is denoted by $x\sigma$ and its inductive definition on the structure of the XML term is given in Fig. 2. Instances of XML terms under ground substitutions are called ground instances. A *unifier* σ for the XML terms T and S is a substitution that produces syntactically identical instances $T\sigma = S\sigma$. The *most general unifier (mgu)* is a unifier that cannot be obtained as the composition of any other. Unification of XML terms provides a uniform mechanism for parameter passing, construction and selection of the information contained in the XML document.

$$\frac{\texttt{<a}\,\mathtt{a}_1 = \mathtt{T}_1 \cdots \mathtt{a}_n = \mathtt{T}_n\texttt{>}\mathtt{B}_1 \cdots \mathtt{B}_m\texttt{</a>} \in P \qquad \exists\sigma'.\ \sigma' = mgu(head(\mathtt{A}_1), \texttt{<a}\,\mathtt{a}_1 = \mathtt{T}_1 \cdots \mathtt{a}_n = \mathtt{T}_n\texttt{>})}{(\texttt{<query>}\mathtt{A}_1\mathtt{A}_2 \cdots \mathtt{A}_n\texttt{</query>}, \sigma) \rightarrow (\texttt{<query>}\mathtt{B}_1\sigma' \cdots \mathtt{B}_m\sigma'\mathtt{A}_2\sigma' \cdots \mathtt{A}_n\sigma'\texttt{</query>}, \sigma\sigma')}$$

$$\frac{(\texttt{<query>}\mathtt{C}_1 \cdots \mathtt{C}_k\texttt{</query>}, \epsilon) \rightarrow^{*} (\texttt{<query> </query>}, \sigma)}{P \models_{\sigma} \texttt{<query>}\mathtt{C}_1 \cdots \mathtt{C}_k\texttt{</query>}}$$

Fig. 2 SLD Inference Relation

4.2 Deductive Database Model

A logical procedure has the form <b b1 = T_1 ... $b_n = T_n$> B_1 ... B_m </b> comprising a *head* and a *body*. The head <b $b_1 = T_1$ ... $b_n = T_n$ > consists of a procedure name b and a list of parameter names b_1 ... b_n associated with their respective terms T_1 ... T_n. The body B_1 ... B_m consists of a sequence of procedure calls. A procedure call has either the form <b $b_1 = S_1$ ... b_n = Sn/> or the form <b $b_1 = S_1$ ... $b_n = S_n$ >A_1 ... A_k </b> where S_1 ... S_n are the XML terms passed to the procedure and $A_1, \ldots, A_m$ are

XML terms. The former case corresponds to a call to a defined procedure, whereas the latter case corresponds to a consult to the extensional database. In any case, unification is used for parameter passing, and equality comparison between documents with complex structure. The function *var* : XMLTerm → **P** XMLVariable is defined to obtain the set of variables occurring in a term (here **P** stands for the power set operator).
The meaning of a logical procedure is given by the interpretation of an SLD-resolution [6]. Conversely, the declarative reading of a logical procedure establishes the validity of the call to the procedure head whenever all the conditions forming the procedure body are valid. An SLD resolution step of the logical procedure <b $b_1 = T_1$ … $b_n = T_n$ >B_1 … B_m </b> in the query <query>A_1 … A_n </query> is obtained by replacing an instance of the body $B_1\sigma'$ … $B_m\sigma'$ under the most general unifier σ of the first call A_1 in the query and the head <b $b_1 = T_1$ … $b_n = T_n$ /> of the procedure. An SLD resolution step is thus defined as the relation $\Rightarrow \subseteq$ (XMLTerm × Σ) × (XMLTerm × Σ) that transforms the query (<query>A_1 A_2 … A_n </query>; σ) into the query (<query>$B_1\sigma'$.. $B_m\sigma' A_2\sigma'$ … $A_n\sigma'$</query>; $\sigma\sigma'$) by replacing the procedure call $A_1\sigma'$ by the instance of the procedure body $B_1\sigma'$ … $B_m\sigma'$ under σ'.

4.3 Active Database Model

The consistency-preserving and reactive behavior are described by *ECA rules*. An alerter mechanism notifies the system when a collection of XML documents changes by inserting or deleting a document. The event section E uses a document template to retrieve the required information sent by the alerter. Then one or more rules may be selected and tested against the content of the document E_{ext} associated with the event. In this case, condition C is checked and if the condition holds, the actions of deleting B or inserting A the argument document are executed. Ordering of actions can be defined by composing rules R either sequentially or in parallel.
The operational semantics of rule execution, given in Fig. 3, defines the reduction relation: **M** XMLTerm × Σ → (**M** XMLTerms × Σ ∪ **M** XMLTerms) where **M** XMLTerm denotes multisets of XML terms, including logical procedures and ECA rules. The notation /P/D denotes a document D located at path P, possibly including the host name. After observing event E_{ext} , if there is a computed answer σ' for both extended program P ∪ {E_{ext}} and condition <query>E C_1 … C_k </query>, then the collection of documents in both extensional and intensional databases are modified by deleting the instance of documents $B_1\sigma'$;…; $B_n\sigma'$ and inserting the instance of documents $A_1\sigma'$; … ; $A_m\sigma'$ under σ'. In the rule, E_{ext} denotes the external document observed by the event handler, whereas E denotes sub-element rule/on of the document that contains the rule. As indicated by including E in the query, the structure of document E_{ext} must match the structure given by E. The difference ⊖ and union ⊕ operators for multisets of documents are used instead of the corresponding operators over sets due to the multiplicity of documents is important. The order in that documents are deleted or inserted is not specified. By using instance of documents under substitutions, the semantics captures the propagation of values among logical variables across documents. The operational semantics for the termination condition of rule execution is given also in Fig. 3. After observing the event E that triggers a rule, the

rule does not execute if the current contents of the XML databases does not entail the rule condition. In this case, the reduction relation $\rightarrow$ leads to the program singleton P.

$$\boxed{\begin{array}{l}\texttt{<rule>}\\ \quad\texttt{<on>E</on>}\\ \quad\texttt{<if>}C_1\cdots C_k\texttt{</if>}\\ \quad\texttt{<do>}\\ \qquad\texttt{<delete>}B_1\cdots B_m\texttt{</delete>}\\ \qquad\texttt{<insert>}A_1\cdots A_n\texttt{</insert>}\\ \quad\texttt{</do>}\\ \texttt{</rule>}\end{array}} \in P$$

$$\frac{\exists\sigma'.P \cup \{E_{ext}\} \models_{\sigma'} \texttt{<query>}E\,C_1\cdots C_k\texttt{</query>}}{(P.\sigma) \longrightarrow (P \ominus \{B_1\sigma',\ldots,B_n\sigma'\} \oplus \{A_1\sigma',\ldots,A_m\sigma'\}, \sigma\sigma')}$$

$$\frac{\neg\exists\sigma'.P \cup \{E_{ext}\} \models_{\sigma'} \texttt{<query>}E\,C_1\cdots C_k\texttt{</query>}}{(P.\sigma) \longrightarrow P}$$

Fig. 3 Reduction Relation of Active Rules

As Web services based business collaborations are long-running interactions that exchange documents enclosed in SOAP messages, rules for receiving and sending documents have been included to extend the programming model of ADM as presented in [9]. The rules coordinate the interaction between internal activities and external protocol messages. Basic workflow activities are modelled by the notification (or the receipt) of a message to (from) a Web service:
- Notifications of messages to other Web servers are modeled by means of non-blocking *send* operations.
- Invocations of synchronous request/reply operations are modeled by means of blocking *invoke* operations.
- Reception of messages invoking composite services are modelled by means of blocking receive operations. If the received message is invoking a composite service, the response to the invoking client is sent by a *reply* operation.

The operational semantics of the basic message exchange operations for receiving and sending documents are given next.
The rule for receiving documents D from either a directory or a document server located at path P is given in Fig. 4. If the document collection, including document /P/D_{ext} , entails the rule condition, an instance of the document collection is obtained by applying the actions given by A (by inserting or removing instance of documents). Otherwise, rule does not apply leaving the document collection unchanged.
The rule for sending local document D_{loc} to either a directory or a server located at P is given in Fig. 5. If local document D_{loc} exists and if collection $P \cup \{D_{loc}\}$ of documents entails the rule condition, an instance of D_{loc} is placed at /P/, keeping un-

changed the local document. Otherwise, the rule terminates maintaining the entire collection with no change.

```
<rule>
 <on>
  <receive>
   <doc>D</doc><from>P</from>
  </receive>
 </on>
 <if>C</if>
 <do>A</do>
</rule>
```
$\in P$

$$\frac{\exists\sigma'.\, P \cup \{\texttt{/P/D}_{ext}\} \models_{\sigma'} \texttt{<query>D C</query>}}{(P \oplus \{\texttt{/P/D}_{ext}\}, \sigma) \longrightarrow (A(P\sigma' \oplus \{\texttt{/P/D}_{ext}\sigma'\}), \sigma\sigma')}$$

$$\frac{\neg\exists\sigma'.\, P \cup \{\texttt{/P/D}_{ext}\} \models_{\sigma'} \texttt{<query>D C</query>}}{(P \oplus \{\texttt{/P/D}_{ext}\}, \sigma) \longrightarrow P \oplus \{\texttt{/P/D}_{ext}\}}$$

Fig. 4 Reduction rule for receiving a document

```
<rule>
 <if>C</if>
 <do>
  <send>
   <doc>D</doc><to>P</to>
  </send>
 </do>
</rule>
```
$\in P$

$$\frac{\exists\sigma', \texttt{D}_{loc}.\, P \cup \{\texttt{D}_{loc}\} \models_{\sigma'} \texttt{<query>D C</query>}}{(P \oplus \{\texttt{D}_{loc}\}, \sigma) \longrightarrow (P \oplus \{\texttt{D}_{loc}, \texttt{/P/D}_{loc}\sigma'\}, \sigma\sigma')}$$

$$\frac{\neg\exists\sigma'.\, P \cup \{\texttt{D}_{loc}\} \models_{\sigma'} \texttt{<query>D C</query>}}{(P \oplus \{\texttt{D}_{loc}\}, \sigma) \longrightarrow P \oplus \{\texttt{D}_{loc}\}}$$

Fig. 5 Reduction rule for sending a document

Though an underlying model of distributed, shared repository of documents is fundamental in the design of ADM, the decision for using the same addressing scheme for both the collection and the document server greately simplifies the programming model since ultimately the active server provides the means for accessing the collection of documents it holds. An unpredictable behavior may arise due to rules may interfere with each other preventing their execution. In order to reduce the non-determinism in the ordering of actions, the sequential and parallel composition of rules are introduced in [9] to schedule their execution.

Nonetheless, the purpose of composition of Web services contrasts with coordination protocols. Coordination protocols, similar to speech acts communication of multi agent systems, makes possible for agents to synchronize their exchange of messages. Conversation compliant with a coordination protocol are supported by conversation controllers, whose purpose is not to execute any business logic, but to control messages to internal objects and to verify protocol compliance. The coordination protocol imposes requirements on how the composition is to take place, since the order in which are invoked has to be compliant with the protocol definition. In comparison, the composition logic determines the conversation that a composite service is able to execute.

5 Related Works

In [10] a far different solution to the problem addressed in this paper is proposed. Executable BPEL4WS documents are not created from a repository of templates. Instead, a dynamic binder and invoker module communicates with a generic Web service proxy to dynamically bind external Web services. Despite the elegance of this solution, the generic Web service proxy becomes a bottleneck because it coordinates all kind of interactions occurring in the business process, from finding suitable candidate services to binding compatible services and invoking them. In comparison, our approach produces separate workflows that run independently and more efficiently from each other without recurring to a central invoker. In [11] mechanisms for dynamically discovering Web services using semantic extensions to UDDI are suggested. In [12] a different approach is presented. They semi automatically generate process composition by using semantic capabilities of Web services. Finally, in [13] another approach is presented combining Semantic Web technology and BPEL4WS to achieve dynamic binding. In comparison to all these approaches, we have not considered any form of semantic matching to automatically select the suitable service required. We believe that the use of the rational aspects of ADM can provide a solution for reasoning about the semantic matching of web services.

6 Conclusions

In this paper we have discussed the need for automating the process of binding dynamically business partner information into a workflow template. As a solution to this problem we have proposed a repository of generic business process definitions using the ADM system. These business processes are widely used in commercial organizations to achieve particular business targets. A simple and uniform model for active and deductive XML databases has been proposed in this paper. The ADM language extends the XML language by introducing logical variables, logical procedures and ECA rules. An experimental distributed system with layered architecture has been implemented that maintains the openness of the XML data by keeping apart the language extensions. As future work, we plan to develop some support for the execution of BPEL4WS workflows.

References

1. J. Bailey, A. Poulovassilis, P. T. Wood, Analysis and optimization of event-condition action rules on XML. Computer Networks, 39:239-259. 2002.
2. N. W. Paton, O. Diaz, Active database systems. ACM Computing Surveys, 31(1): 64-103. 1999.
3. M. Bambara, N. Tamura, Translating a linear logic programming language into Java. In Proceedings of ICLP'99 Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages. pp. 19-39, Dec 1999.
4. O. Lassila, R. Swick, Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommentation, http://www.w3.org./TR/REC-rdf-syntax
5. The DARPA Agent Markup Language Homepage, http://www.daml.org/
6. J. W. Lloyd, Foundations of Logic Programming. Springer-Verlag, 1987.
7. M. Liu, G. Dobbie, T. W. Ling, A logical foundation for deductive object-oriented databases, ACM Transactions on Database Systems, 27(1), pp. 117-151. 2002
8. A. Martelli, U. Montanari, An efficient unification algorithm, ACM Transactions on Programming Language and Systems, 4(2): 258-282, April, 1982.
9. Oscar Olmedo-Aguirre, Karina Escobar-Vázquez, Giner Alor-Hernández, Guillermo Morales-Luna, ADM: An Active Deductive XML Database System. Lecture Notes in Artificial Intelligence Vol. 2972, pp.139-148. April, 2004.
10. Verma K., et al. "On accomodating inter service dependencies in Web process flow composition".
11. Paolucci M., et al. "Importing the semantic web in UDDI", in Web Services, E-Business and Semantic Web Workshop.
12. Sirin E., et al. "Semi automatic composition of web services using semantic descriptions", Web Services: Modeling, Architecture and Infrastructure workshop in conjunction with ICEIS2003, 2003.
13. D. J. Mandell, S. A. McIlraith, Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. The Proceedings of the Second International Semantic Web Conference (ISWC2003).