Applying model checking to BPEL4WS business collaborations

Jesús Arias Fisteus  
Carlos III University of Madrid  
Avda. Universidad, 30  
28911 Leganés, Madrid  
(Spain)  
jaf@it.uc3m.es

Luis Sánchez Fernández  
Carlos III University of Madrid  
Avda. Universidad, 30  
28911 Leganés, Madrid  
(Spain)  
luis@it.uc3m.es

Carlos Delgado Kloos  
Carlos III University of Madrid  
Avda. Universidad, 30  
28911 Leganés, Madrid  
(Spain)  
cdk@it.uc3m.es

Categories and Subject Descriptors
H.4.1 [Information Systems Applications]: Office Automation—Workflow management; D.2.4 [Software Engineering]: Software/Program Validation—Model checking

Keywords
Web services, BPEL4WS, model checking

1. INTRODUCTION

Web services are a key technology for the execution of distributed applications. However, the core Web services standards (SOAP, WSDL and UDDI) are not enough for several application domains, in which a more complex middleware infrastructure is needed for the development of services. One of these domains is B2B E-commerce, in which systems of different enterprises can automatically perform transactions using Web services. B2B collaborations are often complex, in the sense that they are implemented by combining the functionality provided by several Web services. These collaborations are normally coordinated by a business process. As stated in [4], the specification and implementation of these processes is far from trivial. For example, there are often involved several call-back Web services, and processes can have a very long duration (even days or weeks) compared to other application domains. A middleware infrastructure can facilitate their development. BPEL4WS [5] provides such an infrastructure. It is a language for defining complex processes implemented as compositions of Web services. Execution of BPEL4WS processes can be automated and managed by execution engines.

Abnormal situations can occur during the execution of a BPEL4WS process, such as network problems, faults in remote systems, call-back operations that should be invoked but they are not, etc. Process definitions have to deal properly with them, because a failure can cause high economic losses, especially in E-commerce applications. To increase the reliability of processes at design time, we have developed VERBUS (VERification for BUSiness processes), a system for verifying business processes using model checking. Our prototype receives an input BPEL4WS process model and a specification, that is the set of properties that the designer wants to verify. Then it compiles them into a formal specification language and performs the verification using model checkers. The main difference between VERBUS and other similar proposals is its modularity and extensibility: VERBUS is designed to allow the integration of other business process definition languages and verification tools.

This paper is related to two Web services issues necessary for B2B environments: coordination and composition. The coordination concept deals with specifying complex interactions among Web services. The composition concept deals with the implementation of complex Web services that in-
Several languages have been developed for specifying coordination protocols [3], like WSCL, WSCI, or WS-CDL [12], and compositions of Web services, like WSFL. They are often known as choreography languages. BPEL4WS can deal with both coordination and composition. Complex e-commerce collaborations are normally based on the execution of business processes. Web services are a key technology for supporting and performing these business processes [3, 11]. There are many similarities among them and choreographies. Therefore, languages and formalisms developed for choreographies of Web services are very similar to those previously developed for business process management, like XLANG, BPML and ebXML BPSS.

2.1 An Introduction to BPEL4WS

BPEL4WS [5] (Business Process Execution Language for Web Services) is an XML notation for specifying business processes as compositions and coordinations of Web services. It can describe both the external behaviour of a service and its internal implementation.

To illustrate BPEL4WS we will briefly explain a simple example: a company that buys high quantities of olive oil from a provider and bottles it. The company communicates with the oil provider by invoking a Web service in the provider. It has two operations: one for asking the price and one for ordering the oil. The price of olive oil can change from day to day. In order to pay a price as low as possible, the company implements a complex process. The company buys it when the price is low enough (less than a specified price limit). If the price is not lower than the limit before a specified deadline, then the oil is bought with the price it has at this moment.

Fig. 1 shows a simplified version of this process, implemented with BPEL4WS. A new instance of the process starts when an OliveOilRequest message is received. The main attributes of this request are the price limit and the maximum number of days that the process can wait before buying the oil. The complete BPEL4WS and WSDL files for this process are published in the VERBUS home page1.

2.2 Verification of Business Processes

The designer of the previous example process could be interested in verifying several properties about it. For example: “when the oil is ordered the number of days passed is always less or equal than the deadline”. Or: “when the process is completed, the price of the ordered olive oil is less or equal than the price limit, or the number of days passed is equal to the deadline”. In the following sections we show how VERBUS can prove if these properties are true or not.

Several works have been done previously on business processes verification. Woflan [1] is a Petri–Net based verification tool. It can perform verifications on workflow definitions, and was integrated with several commercial workflow management systems. In [7] formal semantics are defined for UML activity diagrams that allow the verification of workflow processes defined with this language. It uses the SMV model checking tool to perform verifications. A framework for the verification of Web services is proposed in [13]. It can perform analysis on a Web service described with DAML-S by translating the description to a Petri–Net based model. A first approach for the verification of BPEL4WS compositions is done in [9]. It proposes a system that verifies processes by translating them to the LTSA system.

3. THE VERBUS FRAMEWORK

Verification frameworks proposed in previous works for the verification of business processes and Web services compositions use a specific definition language and verification tool. Our proposal, VERBUS, is a modular and extensible framework. Its main objective is the integration of languages used to define business processes with verification.

1 http://www.it.uc3m.es/jaf/verbus/
tools. Note that VERBUS does not define these languages and does not implement verification tools: it proposes a framework in which existing ones may be integrated.

To fulfil this objective, VERBUS defines an intermediate common formal model that disconnects definition languages and verification tools. Instead of compiling directly process definitions to verification specifications, the compilation in VERBUS is performed in two steps. Definitions are compiled first into specifications based on the common formalism. Then, these specifications are compiled to verification specifications. The VERBUS framework can be viewed as a layered architecture, represented in Fig. 2:

- The process definition layer (layer 3) deals with designing business processes compositions, using specification languages like BPEL4WS or BPML.

- The common formal model layer (layer 2) is made up by a formalism for modelling processes and a language for describing these models. The formalism is based on transition systems. It is the core of VERBUS, and its main contribution.

- The verification layer (layer 1) deals with verifying business processes. Several general-purpose verification tools can be placed in this layer.

Each process definition language is integrated by defining its semantics in terms of the common formal model, and implementing a compiler based on them. Each verification tool is integrated by implementing a compiler from layer 2 formal specifications into specifications expressed with the input language of the verification tool.

The layer 2 formalism was presented and formally defined in [8]. Following, a brief overview of it is provided. A process is composed by a set of attributes and a set of transitions. Each attribute has a value within a set of possible values. The value of all the attributes of the process at a given moment establishes its state. The process progresses from one state to another by means of transitions. A transition is a pair of states (origin and destination) that defines a possible progress of the process. The process starts at an initial state. Then it fires transitions, until it reaches a state that is not in the origin of any transition. This state implies the completion of the process and is called a final state.

The FSM of a business process may have many transitions. To avoid defining them explicitly, VERBUS uses a compact representation based on functional transitions. A functional transition is defined by two predicates: domain and action. The domain defines a set of states that are origin of transitions. The action defines how these origin states change to obtain the final states. Therefore a functional transition represents a set of transitions that model a similar behaviour.

The concepts of entity, entity type and activity are introduced as notational elements, to make specifications more readable. However, they do not affect the basic formalism. An entity type is a group of typed fields (boolean, enumerated or integer). An entity is an instance of an entity type. A field of an entity type generates an attribute each time this entity type is instantiated. An activity is a logical unit for grouping related functional transitions. Sample fragments of a layer 2 specification are showed in next section.

Figure 3: Mapping of BPEL4WS variables (up) to CFM entities (bottom).

4. COMPILING BPEL4WS TO THE COMMON FORMAL MODEL

BPEL4WS is integrated in VERBUS by defining its semantics in terms of the common formal model (CFM), as described in this section. The description is centred in, as representative features, variable declarations, the sequence activity, the fault-handling mechanism and reception of messages. Complete examples of these and other BPEL4WS features can be found in the Web page of VERBUS.

4.1 Variables

BPEL4WS variables are mapped to CFM entities. Each variable is an instance of a data type defined by a WSDL message, an XML element or an XML Schema type. First, its data type is defined as an entity type. Then, it is instantiated as an entity. Simple data types are transformed to CFM data types if possible (boolean, enumerated and integer), or declared as abstract otherwise. Complex data types are transformed by recursively transforming their components. Fig. 3 shows the mapping of the message OilOrder. It has three simple parts. The part quantity is defined as an CFM integer type, customerId as abstract and oilType as enumerated. Abstract types that are compared to constant values anywhere in the process are mapped as enumerated types based on these constants.

4.2 Activities

The execution of each BPEL4WS activity instance is controlled by a life-cycle. Two different life-cycle types were identified in this work, depending on the type of activity (Fig. 4). The general life-cycle is used for activities that can have handlers (process, scope and invoke). The simple life-cycle is used for the other activities. Each activity is mapped to an entity and several functional transitions. The entity represents the state of the activity in its life-cycle. The functional transitions represent the way the activity can progress through its life-cycle and how this progress affects to the attributes of the process. The actual functional transitions of each activity depend on its activity type. However, there are several rules that are common to almost all activities. Activity instances have always a begin transition, that represents the beginning of their execution. Its domain
Figure 4: Life-cycle for activities. The general life-cycle is on the left and the simple life-cycle is on the right. States are labelled with uppercase letters and transitions with lowercase letters. States containing a black dot can be final states of the life-cycle.

represents the preconditions of the activity. Normally it is a condition that checks the state of other related activities, and its own state (it should be \textit{not started}). Its action changes the state of the activity to \textit{running}. Activity instances have normally a \textit{complete} transition also, that represents the end of their execution. Its domain checks that the activity is \textit{running}. Depending on the activity type, it may include other conditions. Its action puts the activity in \textit{complete} state, and can change also attributes of the process to represent the effects of the activity execution.

If an activity has a non-deterministic behaviour then it is modelled with several mutually-exclusive functional transitions. Each of them represents a different behaviour of the activity. Samples of this include the \texttt{pick} activity in which several messages or alarms can occur, activities that can either complete normally or throw a fault, activities \texttt{receive} or \texttt{invoke} that can receive messages with different data, etc.

The BPEL4WS sequence activity contains one or more inner activities, that are executed in sequential order: an activity can start when the previous one has been completed. The sequence itself is completed when its last inner activity has been completed. This behaviour is modelled by adding a condition to the domain of the \texttt{begin} transition of each inner activity. The condition of the first inner activity states that the sequence activity should be \textit{running}. The conditions of the other inner activities state that the preceding activity should be \textit{completed}. In order to complete the \texttt{sequence} activity, its \texttt{complete} transition states that its last inner activity should be \textit{completed}. Fig. 5 shows an example.

4.3 Receiving messages

A BPEL4WS process can receive input messages in \texttt{receive} or \texttt{pick} activities, and output messages in \texttt{invoke} activities. However, the designer is not normally interested in modelling completely the behaviour of the external entities that send these messages. VERBUS permits the definition of a simple abstract model of them, normally sufficient for verification purposes. It is constructed as a list of alternative messages that the process could receive in a given activity. Each of them is mapped to a transition in the CFM. Messages can be specified in two ways: literally or with BPEL4WS activities. They are specified literally by giving the whole message as a constant. If they are specified with BPEL4WS activities, the modeller can construct them dynamically from the value of variables of the process.

4.4 Fault handling

BPEL4WS has a powerful fault-handling mechanism similar to the Java exception mechanism. The scope-like activities (\texttt{process}, \texttt{scope} and \texttt{invoke}) may provide fault handlers. When a fault is thrown in a given activity, a handler in the immediately enclosing scope is selected, based on the fault name and variable type. Before the handler is executed, all the running inner activities of this scope are cancelled. If no handler is appropriate, then the whole scope is cancelled, and the fault is re-thrown to the next enclosing scope. A fault that reaches the \texttt{process} level causes the cancellation of the whole process. This mechanism is mapped to CFM by adding fault specific attributes to the complex life-cycle and defining new functional transitions in activities that perform cancellation and handling tasks.

4.5 Prototype Implementation

The current prototype of VERBUS is mainly composed by three compilers: from BPEL4WS into the CFM, and from CFM into Promela (Spin [10]) and SMV/nuSMV [6]. It is
5. VERIFICATION OF PROCESSES

The main lack of the current prototype is the BPEL4WS compensation mechanism, because of the complexity associated with it: a copy of all the variables must be saved for each completed activity instance, because activities should be compensated using the value that variables had when they were completed. While loops make this even more complex, because multiple instances of each inner activity can be created. This feature will be handled in future versions of VERBUS, by storing a copy of attributes for each completed activity. This will increase the complexity of the verification, and therefore a configurable parameter will be added to limit the maximum number of activity instances.

6. CONCLUSIONS

In this work we have presented VERBUS, our proposal for business processes and Web services compositions verification. The main difference between VERBUS and other previous works is its modularity and extensibility, provided by a common intermediate formal model. The implementation of a prototype of VERBUS has demonstrated the feasibility of the framework. It integrates the BPEL4WS language and the Spin and SMV model checkers. Although the common formal model is simple, it was found expressive enough even for modelling the complex fault handling and cancellation mechanisms of BPEL4WS. Future work includes integrating other business process definition languages and analysing the expressivity of the CFM by applying workflow patterns [2].

Acknowledgements

This work is partially supported by the Spanish Science and Technology Ministry, project TIC2003-07208 “Infoflex”.

7. REFERENCES