Software Process Modeling and Execution: The UML4SPM to WS-BPEL Approach [1]

Reda Bendraou[1], Andrey Sadovykh[2], Marie-Pierre Gervais[1, 3] and Xavier Blanc[1]

[1] Laboratoire d'Informatique de Paris 6, 104 Av. du Président Kennedy - F75015 PARIS
[3] Université Paris X
{Reda.Bendraou, Marie-Pierre.Gervais, Xavier.Blanc}@lip6.fr
[2]Softeam
27 Av. Victor Hugo F75008 PARIS
Andrey.Sadovykh@softeam.fr

Abstract. *Over the two past decades, the software process modeling community is being confronted to the following dilemma: how a Software Process Modeling Language (SPML) can be sufficiently abstract to hide the increasing complexity of development processes while being precise enough to be executed? Since no SPML succeeded in satisfying these apparently conflicting requirements, in this paper we propose to combine two languages: UML4SPM, an UML2.0-based Software Process Modeling language and WS-BPEL (Web Services Business Process Execution Language). While UML4SPM brings expressiveness, understandability and abstraction in modeling software development processes, BPEL provides a semantically rich set of concepts for process executions. The mapping between the two languages, how do they complement each other, some issues and the value of the approach are discussed.*

Key Words: Software Process Modeling, Process Enactment, UML, WS-BPEL, PML, Workflow, MDD.

1. Introduction

The standardization of UML (Unified Modeling Language) [OMG 05a] and its successful adoption by the industry and academia has naturally attracted the attention of the software process modeling community. The principal ingredients that participate in the success of UML, among others, are its ability of abstracting the complexity of systems under specification and the use of an intuitive and understandable set of notations and diagrams. Therefore, the possibility of using UML as a software process modeling language has been largely explored in the literature [Jäger 98] [OMG 02] [Di Nitto 02] [Chou 02]. However, whether UML provides a high-level of abstraction and understandability in representing process models, it lacks of some semantics, concepts and tools for their execution [Rumpe 02]. On the other hand, in the Business Process Management (BPM) domain, recently, a consolidation has led to a single language for business process executions: the Business Process Execution Language for Web Services (WS-BPEL, BPEL for short) [WSBPEL 07]. Rapidly, BPEL gained importance and became the "Language" for business process orchestrations. Many tool vendors already provide training supports and process engines for this standard [ActiveBPEL].

In this paper we explore the possibility of combining both standards for the purpose of software process modeling and execution. In addition to the fact that UML and BPEL share the common point of being standard, widely adopted and many people are familiar with their use, they can be used to complement each other. While UML comes with a high degree of abstraction, expressiveness and notations suitable for modeling software processes, BPEL provides concepts and precision required for their execution support. In this context, we use our UML2.0-based Language for Software Process Modeling (UML4SPM) [Bendraou 05] [Bendraou 06] as a high-level language for modeling software processes. UML4SPM process descriptions will be then mapped to BPEL specifications in order to be executed. Our main motivations for combining both languages are first, to keep a clear separation between the business concerns of software process descriptions (i.e., Phases, Activities, Roles, etc.) and all the technical and organizational features needed for their execution support (Task sequencing, Artifacts assignment, alarms, events and exception handling, etc); second, to leverage the maturity level of the BPM field and the bunch of existing tools instead of starting from scratch. This approach will reinforce the connection between process modeling tools and process execution tools.

In the following, we start by introducing UML4SPM, our UML-Based Language for software process modeling. To demonstrate the feasibility of the approach, in Section 3, we present a software process example, which we model using the UML4SPM notation. After a brief presentation of WS-BPEL in Section 4, the process example is used in Section 5 for demonstrating the mapping between UML4SPM and BPEL. We will discuss this mapping but most of all; we will share some of the feedbacks and issues we had while experimenting the approach. Section 6, concludes this work and draws some future perspectives.

[1] This work is supported in part by the IST European project "MODELPLEX" (contract no IST-3408).

33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007)
0-7695-2977-1/07 $25.00 © 2007 IEEE

2. UML4SPM

UML4SPM is a UML2.0-based Language for Software Process Modeling. Expressiveness, understandability, precision and executability were our main requirements while designing UML4SPM. The language comes in form of a MOF-Compliant metamodel, a precise semantics and a simple yet expressive graphical notation and diagrams. Hereunder, we start by presenting the UML4SPM metamodel.

2.1. UML4SPM Metamodel

The UML4SPM language is defined as a MOF-compliant metamodel. It contains two packages: 1) the *UML4SPM Process Structure* package, in which we defined the set of primary process elements with a semantic proper to software process modeling (see figure 1.); 2) the *UML4SPM Foundation* package, in which we reuse *Activities* and *Actions* elements from UML2.0 Superstructure [OMG 05a]. These elements provide UML4SPM with coordination mechanisms, and executability semantics for the enactment of process's activities. Herein, we start the UML4SPM presentation by the *Process Structure* package:

UML4SPM Process Structure Package

The building block of any UML4SPM process model is the *Software Activity* element. It describes any effort or piece of work to be performed during the development process. It has a *description* property that briefly outlines what has to be done by *Responsible Roles* of the activity, a *priority* ranging from *low* to *high* to highlight its importance within the process and a *complexity* property to show its degree of difficulty (i.e., *easy*, *medium* or *difficult*). The *isInitial* property is to tell whether the activity is the initial one within the process or not, i.e. it plays the role of a container that will encapsulate all process activities. A special behavior is assigned to it and it is considered as the current context of the process.

Thus, any UML4SPM process model should have an outermost *Software Activity* with its *isInitial* property set at "true" and that encapsulates all subsequent activities. A *Software Activity* may be totally executed by a machine. Then, the *kind* property is set to "machine execution". Otherwise, it is fixed at "human execution" if a human expertise is required. A *Software Activity* has a *SoftwareActivityType*, which can be for instance a *Phase*, an *Activity*, a *Sprint* (term used in the Scrum agile process), etc.

In order to realize a *Software Activity*, one or more *Role Performers* are assigned however this is not mandatory at process specification time. The *Responsible Role* element defines *responsibilities* and *qualifications* required from the *Role Performer* to realize the activity. A *Role Performer* may be a *Tool,* an *Agent* with a *name* and *skills* or a *Team*, which in its turn may be composed of *Agents* or *Teams*. In order to help the *Role Performer* to perform the activity, *Guidance* may be provided. *Guidance* may be guidelines, checklists, tool tutorials, etc.

Another essential element is the *WorkProduct* element. It represents any physical piece of information consumed, produced or modified during the software development process. A *WorkProduct* has a unique identifier specified by the *idWorkProduct* property. A *WorkProduct* may be either a process *deliverable* or not, and an *uriLocalization* property that serves at determining the *WorkProduct* location during process execution. Finally, the *Version* and *lastTimeModified* properties were defined in order to help developers in avoiding confusion while manipulating different versions of the same *WorkProduct* during development activities. The UML4SPM meta-classes we introduced in this section represent the constructs and semantics required to represent primary process model elements. However, this is insufficient. Coordination of *Software Activities* (i.e., control and data flows), the ability to express events, decisions, iterations, exceptions, and interactions is still lacking. This is where the *UML4SPM Foundation* package comes into action.

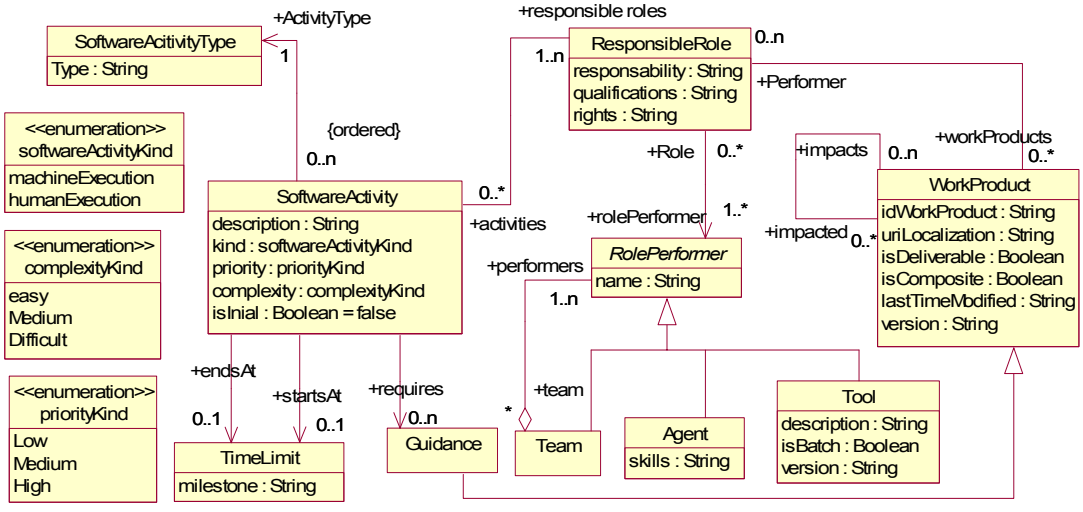

Figure 1. UML4SPM Process Structure package elements

UML4SPM Foundation Package

In this section, we give a brief description of the UML2.0 subset that we identified as a basis of UML4SPM:

- Activity*:* in UML2.0, an *Activity* is the specification of a parameterized behavior defined in terms of a coordinated sequencing of *Actions* [OMG 05a]. The sequencing of these actions is ensured using an *Object* and *Control flow* model. The former is used to sequence data produced by one action that are used by other actions. The latter is used to explicitly sequence the execution of actions. Activities also include *Control Nodes*, which structure *control* and *object* flow between actions. In addition to *Initial* and *Final Nodes*, these include *Decision* (to express choices), *Fork* (parallelism), *Join* (synchronization) and *Merge Nodes* (to accept one among several alternate flows). *Object Nodes* in activities are to represent objects and data as they flow in and out of invoked behaviors (Activities) or Actions. As UML4SPM *Software Activity* element extends UML2.0 *Activity*, we take advantage of all its properties and associations. Thus, a *Software Activity* can be composed by other *Software Activities* and may contain *Actions* (the hierarchy dimension). An UML2.0 *Activity* being indirectly a *Classifier*, the possibility to specify new *properties* and new *operations* is then offered to *Software Activities*. A *Software Activity* being now a specialization of UML2.0 *Activity* meta-class, the specification of *pre* and *post conditions* on the execution of a *Software Activity* is also rendered possible.

- Artifact: The UML2.0 standard defines an *Artifact* as a *Classifier* that represents a physical entity. It may have *Properties* that represent its features, and *Operations* that can be performed on its instances. It can be involved in associations to other *Artifacts* (e.g., composition associations). Examples of *Artifacts* include model files, source files, scripts, and binary executable files, a development deliverable. The UML4SPM *WorkProduct* element extends UML2.0 *Artifact*. An *Artifact* being a *Classifier*, *WorkProducts* can be defined as *InputPins* and *OutputPins* of *Software Activity's Actions*. They can also have additional properties and operations than those we explicitly defined. It is possible to specify composite *WorkProducts* thanks to the "nested artifact" association. Finally a *WorkProduct* may be associated with a state machine that defines its allowable states and operations to switch between them.

Due to space restrictions, we cannot present in details the precise set of UML2.0 *Actions* and *Activity* elements (control flow, object flow, events, exception handling, etc.) we identified as a basis for software process modeling as well as those we newly defined. Hereunder is a table (see table 1) that just enumerates them. Their definitions are given in the standard [OMG 05a]. Their use, notation, new elements we defined and a discussion on how UML4SPM reaches the expressiveness, understandability and precision requirements are given in more detail in [Bendraou 06].

Actions	***Activity Elements***
AcceptEvenAction, Action, CallBehaviorAction, CallOperationAction, SendSignalAction, RaiseExceptionAction	Activity, ActivityFinalNode, ActivityParameterNode, ConditionalNode, ControlFlow, DataStoreNode, DecisionNode, ExceptionHandler, FinalNode, ForkNode, InialNode, InterruptibleActivityRegion, JoinNode, MegeNode, Pin (Inputpin, Outputpin), LoopNode, ObjectFlow, StructureActivityNode

Table 1. The identified set of UML2.0 Actions and Activity elements suitable for process modeling

In the next section, we present the software process example that we will use for demonstrating the UML4SPM to WS-BPEL approach. The example is then modeled using the UML4SPM notation.

3. The Software Process Example

In this section we introduce a simple yet representative example of a portion of a software development process. This process example was provided by our industrial partners within the IST European Project MODELPLEX, which this work is part of [MODELPLEX 06]. The process example will be first described in natural language and then represented using UML4SPM.

The process is composed of two phases: "Inception" and "Construction" phases. The "Inception" phase is composed of two activities. The "Elaborate Analysis Model" activity and the "Validate Analysis Model" activity. The "Elaborate Analysis Model" activity takes as input "Work Specifications" (i.e. requirement documents) and produces an UML "Analysis Model". The "Analysis Model" is then taken as input by the "Validate Analysis Model" activity which is composed of the following steps: 1) Get the "Analysis Model" (which is in this example a UML Model); 2) Submit the UML model for validation to an UML Checker Tool which will emit a validation report; if the "Analysis Model" is valid then send an email to the development team and go to the next phase. If the "Analysis Model" is invalid, then send an email to the development team and comeback to the "Elaborate Analysis Model" activity. The role in charge of both activities of this phase is ensured by the "Analyst". For brevity reasons, the "Construction" phase is skipped.

Looking at the process description we can notice some aspects that characterize software development processes. The first one is the hierarchy of the process. We have a Phase, which may contain Activities, which in their turn may contain steps. The second aspect is the presence of both human activities and automated activities, which makes it difficult to automate the entire process. Finally, the transformation process of artifacts

from one activity to another and the necessity to know the artifact's states at any time of the process.

3.1. Software Process Example Description Using UML4SPM Notation

The graphical representation of a UML4SPM *Software Activity* is given in figure 2. As we can notice, it differs slightly from the one proposed by the UML2.0 standard. This is because it has new properties and associations specific to software process modeling that we newly defined. Precision was a major requirement for this notation. At a glance, the *Agent* or the customer can know the name of the activity, its input and output parameters, its priority in the process, its duration, the assigned roles, the tools used for performing the activity, accepted and triggered events. Post and pre conditions may be expressed in natural language or by means of OCL2.0 constraints (Object Constraint Language). More details on the notation are given while commenting the process description.

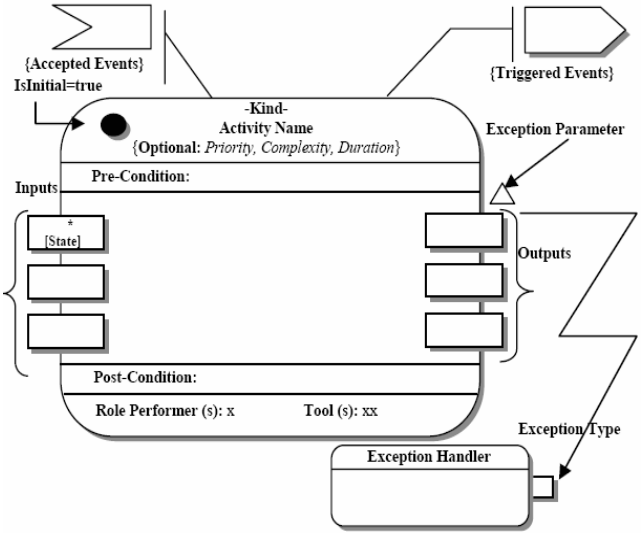

Figure 2. UML4SPM Software Activity Notation

The process description focuses on the "Inception Phase" and activities it owns (figure 3).

As we can notice, the "Inception Phase" activity represents the context of this process. This is indicated by the start-blob in the top-left corner. It is used to coordinate between different activities and workproducts of the process. The "M" letter is to indicate that the activity is machine-executable (H for Human execution). One important aspect is the use of *CallBehaviorActions* in order to initiate/call process's activities (e.g., "Elaborate Analysis Model" call). In the call, we have to precise 1) whether the call is synchronous (use of a compete arrow in the top-left corner) or not (half arrow, e.g., "Construction Phase" call); 2) the parameters of the call, which represent workproducts inputs/outputs of the activity. The parameter types may be *in*, *out* or *inout*. Another aspect is the use of *Decision* and *Merge* nodes. The decision node allows expressing a choice of actions to do depending on a condition (in this case whether the analysis model is valid or not). The merge node here is used to express that the "Elaborate Analysis model" activity may be triggered by one of the two possibilities. The first one is when the "Inception Phase" activity is lunched. The second one is when the analysis model validation fails. In the next section we introduce BPEL and how this orchestration language can be used as a support for UML4SPM process model executions.

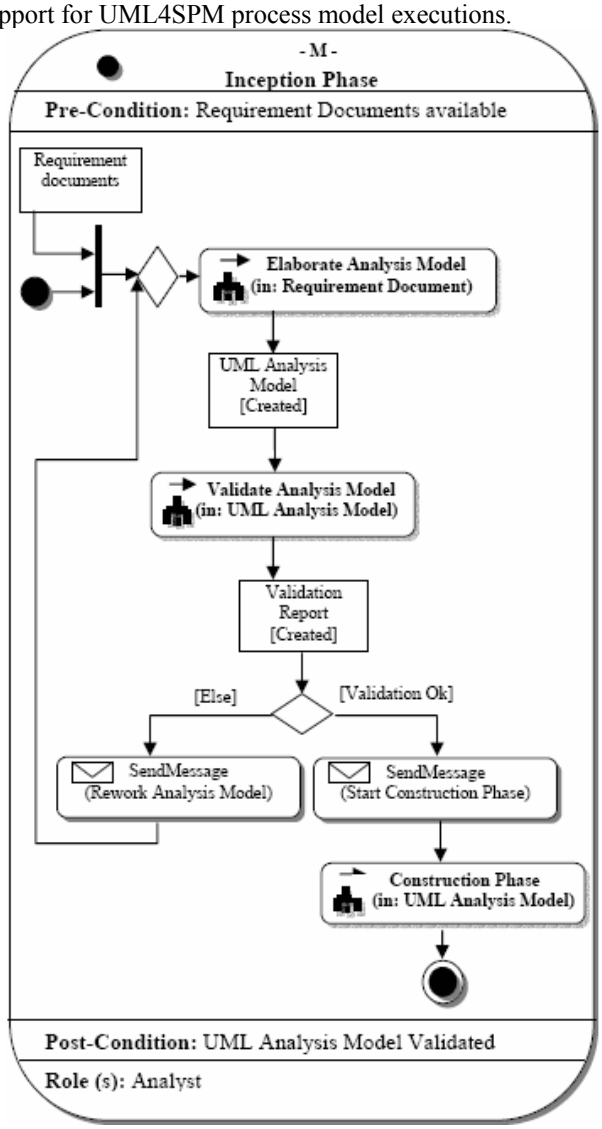

Figure 3. "Inception Phase" Activity

4. WS-BPEL2.0

BPEL is an XML-based standard for defining how a set of Web services can be orchestrated (i.e., combined) in order to implement business processes [WSBPEL 07]. It is built upon WSDL (Web Services Definition Language) and XML Schema. A BPEL process definition is serialized in XML and owns a number of activities. Activities fall into two categories: *Basic Activities* and *Structured Activities*. Basic activities correspond to atomic actions such as: *invoke*, invoking an operation on a Web service; *receive*, waiting for a message from a partner; *reply*, replying to a partner;

assign, assigning a value to a variable; *exit*, terminating the entire process instance; *empty*, doing nothing; and etc. In WS-BPEL2.0, new activities were introduced such as *if-then-else*, *repeatUntil*, *validate*, *forEach* (parallel and sequential), *rethrow* and *extensionActivity*. Structured activities impose behavioral and execution constraints on a set of activities contained within them. These include: *sequence*, for defining an execution order; *flow*, for parallel routing; *switch*, for conditional routing; *pick*, for capturing a race between timing and message receipt events; *while*, for structured looping; and *scope*, for grouping activities into blocks to which event, fault and compensation handlers may be attached [Ouyang 06] [Dobson 06]. BPEL processes are closely coupled with WSDL. A BPEL process provides a web service interfaces described in WSDL and at the same time deals with services that also have to be described in WSDL. From this point of view, a BPEL process represents a compound web service. In the next section, we present mapping rules from UML4SPM to WS-BEPL.

5. From UML4SPM to WS-BPEL

In this section, we address the mapping between UML4SPM and BPEL. We will start by introducing the mappings between concepts of both languages. Then, we will discuss some obstacles we faced while establishing these mappings. A discussion on the human interaction is also addressed. Finally, a brief description of the transformation is given in natural language.

5.1. Mapping Rules

Table 2 lists major mappings between UML4SPM and WS-BPEL2.0. UML4SPM proposes new concepts that deal with the modeling of software process concerns (.i.e., *Roles*, *Guidance*, *Artifact*, *TimeLimit*, etc.) and reuses UML2.0 *Activity* and *Action* package elements, which deal with actions sequencing and synchronization, exceptions, events, invocation, etc. In the following, we propose mappings for both concepts.

In the literature, we can find some work done for mapping UML activity diagrams to BPEL. In [Mantell 05], the author maps UML1.4 Activity diagram elements to BPEL1.1. In UML1.4, Activity diagrams were completely different from UML2.0 ones. They were a special case of state diagrams and no actions with executable semantics were provided. This resulted to a very coarse-grained mapping with only few correspondence rules proposed (e.g. A UML Class maps to a BPEL Process, UML Activity to a BPEL Activity, and so on). With the adoption of UML2.0, Activity diagrams are enriched with executable semantics actions. These actions reduced the gap between both languages (i.e.,UML2.0 and BPEL.) In [Korherr 06], authors define a UML2.0 Profile for BPEL1.1 and propose a mapping between the two formalisms. However, this was only restricted to actions and did not cover activity elements such as Fork node, Decision node, Control Flow, etc. Similarly, in [Bodbar 04], author concentrated on UML2.0 actions. Mappings for *Control Nodes* (fork, join, merge, etc.), *Loops*, and *Exception constructs* were not defined. Moreover, authors map the UML2.0 *Control Flow* as a BPEL1.1 *Sequence* activity. However, a UML *Control Flow* can only link two activities (i.e., When activity A finishes, B starts). While the BPEL *Sequence* activity defines a block where one or more activities are to be performed sequentially.

UML4SPM	BPEL
Software Activity	BPEL Process
SoftwareActivityType	BPEL Variable with name = "ActivityType" and type = "String"
Software Activity's attributes and associations	BPEL Variable with name = "attributeName" (respectively "associationEndName") and type = "XSD_Type". The type may be simple or complex and defined in a xsd file
Software Activity hierarchy and enclosing elements (actions, inputpins and ouputpin, control nodes, etc)	BPEL Sequence or Flow elements
Pre and Post Conditions of a SA	BPEL Transition Condition element
Value of the Pre/Post Condition	The text element of the BPEL Transition Condition
WorkProduct input or output of Actions	BPEL Variable with attribute MessageType equals to the WorkProduct Type. If the Action has more than one WorkProduct than one WSDL Message Part (name=workProductName) with its type is to be defined for each WorkProduct whithin the MessageType. The attributes of the WorkProduct have to be defined in the type of the WorkProduct in an xsd file
Responsible Role	BPEL Variable
TimeLimit of a SA	BPEL Variable
Guidance	BPEL Variable
Team	BPEL Variable
Agent	BPEL Variable
Tool	BPEL Variable
AcceptEventAction	BPEL Receive Activity
AcceptEventAction that waits for an event among a list of possible events	Pick activity. Accepts a message among a list of possible expected messages
AcceptCall Action et ReplyAction to model synchronious calls	BPEL Receive activity with a Reply and input and output specification
Variable (in the context of a StructuredActivity)	BPEL Variable with name and type
ReadVariableAction followed by a WriteVariableAction	BPEL Assign with From (for reading) and To (for writing) within the Copy element
CallBehaviorAction (Sync / Async)	BPEL Invoke activity (with input and output specification / Only input specification)
CallOperationAction (Sync / Async)	BPEL Invoke activity (with input and output specification / Only input specification)

RaiseExceptionAction. The exception type is defined by the action's InputPin	BPEL Throw activity. Throw has a FaultVariable attribute that corresponds to the exception type
An AcceptEventAction that wait for a TimeEvent	BPEL Wait activity. Waits for a deadline (use of Until element) or a duration (use of For element)
AcceptEventAction	onEvent in the EventHandlers section
InitialNode	BPEL Receive with a CreateInstance=true
FinalNode	BPEL Exit activity may be used to abort the process
ControlFlow	BPEL Link element combined with Source and Target elements
ObjectFlow	BPEL Assign with From (the source) and To (the target) within the Copy element
DecisionNode	BPEL IF activity witht Condition element to express the condition
ExceptionHandler	BPEL FaultHandlers with Catch
ForkNode to express parallelism.	BPEL Flow Activity
JoinNode	BPEL Link element combined with Source and Target elements
LoopNode with Test expressed via the association test:ExecutableNode	While activity with element Condition
	RepeatUnitl activity
	ForEach activity
StructuredActivity (defines an activity with its actions, control nodes, variables limited to the activity scope, etc.)	BPEL Scope Activity with all its partnerlinks, variables, faulthandlers, etc

Table 2. UML4SPM to WS-BPEL2.0

While establishing these rules we have noticed many observations. The most important one relates to the fact that all elements in UML4SPM that provide semantics proper to software process modeling have no equivalent in BPEL. All elements such as *Responsible Role*, *Guidance*, *Time Limit*, etc are converted to BPEL process variables. On the other hand, all elements that deal with the coordination of activities, events, exception handling, etc. map easily to BPEL concepts. This observation comforted us in our choice of combining the two standards, one for process modeling and communication, and the other one for process execution. The second observation is that there is no one-to-one correspondence between UML4SPM elements and BPEL elements. As we can see in the table an UML4SPM element (e.g., *LoopNode*) can be mapped into different BPEL elements (i.e., *While*, *Repeat Until*, or *ForEach* activities). This implies that during the transformation phase, the process modeler has to choose one mapping rule among those proposed (if multiple choices) and always apply the same one along the process specification. On the other hand, there are some BPEL concepts that have no equivalent in UML4SPM such as *Validate*, *Empty*, or *Extension Activities*.

Another important aspect relates to the impossibility of BPEL to support some Control Flow patterns, more commonly known as workflow patterns [Van der Aalst 03]. Indeed, BPEL lacks support of multiple merges pattern (merge many execution paths without synchronizing) and discriminators pattern (merge many execution paths without synchronizing. Execute the subsequent activity only once). It also does not allow the synchronization of multiple instances of the same activity and lacks support of arbitrary cycles (e.g. in our process example, there is a cycle between "Send Message" activity and "Elaborate Analysis Model" activity). Similarly, in [Wohed 04], authors evaluated UML2.0 Activity diagrams against workflow patterns. Activity Diagrams succeeded in fulfilling sixteen of the twenty patterns proposed. Among those that were not satisfied, the *Synchronizing Merge* and the *Milestone* patterns. Details about the patterns can be found in [WfP]. These lacks then have to be taken into account while modeling software processes with UML4SPM in order to avoid the use patterns that are not supported by BPEL and vise versa. To avoid for instance arbitrary cycles we propose to combine the use of a *SendSignalAction* and an *AcceptEventAction*. These concepts can be an alternative to cycles and map to BPEL concepts (*Invoke* and *Receive* activities).

5.2. Human interactions

While some business processes can be fully automated, software processes are composed of creative activities (i.e., modeling, checking, communicating, decisions, etc.) that make them need a support for human interactions. Even, in the field of BPM, it has been recognized that the human dimension is essential for process realization. We can notice in Table 2 that BPEL does not provide any support for this kind of activities. In UML4SPM, we have the possibility to express that an activity is automated or has to be carried out by a human. This data can then be mapped as BPEL process variable that the process engine can take into account at enactment time. In order to deal with this issue, we decided to reuse a very interesting work done by industrials known as "BPEL4PEOPLE" [Kloppmann 05]. In BPEL4PEOPLE, a new BPEL activity called *People* activity is introduced. A *People* activity is a basic activity, which is not realized by a piece of software but an action performed by a human being. It can be associated with a group of people, a generic role, etc. The extended BPEL engine creates for each *People* activity - depending on its contents - a list of tasks, also called work items ("to-dos") and affect them to the appropriate process participants. A generic user interface is associated with each task of the activity in order to highlight inputs/outputs of the activity, deadlines, to add the possibility to attach other materials (e.g., guidelines) and to ease communication between agents. In UML4SPM, each human software activity will be mapped to a *People* activity. Each action within the activity will be considered as a task (a work item). *InputPins* and *OutputPins* of actions will be used as I/O of tasks. Regarding the implementation of tasks, BPEL4PEOPLE leaves the choice to the modeler between five possible configurations. These five configurations, that we will not detail here, fall roughly into two kinds: *Inline Tasks* and *Standalone Tasks*. Inline tasks are defined as part of the *People* activity or

of the BPEL process (they have access to the process context, variables, etc.) while standalone tasks are defined outside the process. Standalone tasks may be accessed through 1) implementation-specific invocation mechanisms (i.e., no WDSL), 2) a Web service interface defined with WSDL or 3) a BPEL *Invoke* activity that calls a Web service implemented by the task (WSDL + binding). We opted for the latter configuration. Main reasons are: 1) to promote reusability of standalone tasks by other processes, 2) to use tasks in a distributed environment since they offer a WSDL interface, 3) to avoid BPEL engine extensions, since that solution is generic and does not need a support of the new *People* activity kind. However, process modeler can decide to use another configuration among the five that BPEL4PEOPLE proposes if needed.

5.3. Transformation

For experimentation purposes, the transformation of UML4SPM process models into BPEL code is currently carried out by a Java program. However, we plan to formalize the transformation with a model transformation language such as ATL [ATL 06]. Hereunder, we present in natural language main steps of the transformation algorithm:

1) The creation of an empty BPEL process definition; 2) Generation of the "import" and "variable" section. All UML4SPM elements in table 2 that map to a BPEL variable are processed here; 3) then, the "flow" section is created followed by the "links" declaration. All UML4SPM control flows are generated as BPEL "links" and the *Source* and *Target* elements are documented; 4) The BPEL "flow" starts with a “receive” activity, which is used for communicating input Work Products to the BPEL process. This activity should also contain "createInstance" attribute equals to "True" to indicate that the process is instantiable; 5) “Human” activities are transformed into a pair of linked "invoke" / "receive" activities implementing an asynchronous call of "Workflow Administration" Web service which is a service offering a GUI we defined. Other activities are transformed into synchronous "invoke" activities that have to be completed after the BPEL code generation in order to indicate web services to be used. The remaining UML4SPM elements are transformed according to what was defined in Table 2; 6) Finally, the "import" section is filled manually in order to document the "partner link" and the WSDL location of Web services the process uses, in particular here, the "Workflow Administration" Web service. The generated BPEL process is to be deployed with a conventional BPEL engine, ActiveBPEL in our case. Then, the process is run according to the BPEL process definition. All human tasks are to be redirected to the "Workflow Administration" Web Service which provides a console for guiding the agent in performing the task. Listing 1 gives a sample of the generated process example defined with UML4SPM in section 3. Due to space restrictions, we skipped some process's activities of the example.

```
  <bpel:process  xmlns:bpel="http://docs.oasis-
open.org/wsbpel/2.0/process/executable"
  xmlns:ns1="http://www.softeam.fr/WorkflowAdmi
nistration/"
xmlns:ns2="http://www.example.org/orchestration
/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
name="Inception"      suppressJoinFailure="yes"
targetNamespace="http://Inception">
     <bpel:import
importType="http://schemas.xmlsoap.org/wsdl/"
location="WorkflowAdministration.wsdl"
namespace="http://www.softeam.fr/WorkflowAdmini
stration/"/>
        .......
     <bpel:partnerLinks>
        <bpel:partnerLink
myRole="HumanActivityFacade"
name="HumanActivity"
partnerLinkType="ns1:HumanActivity"
partnerRole="HumanActivityFacade"/>
              .......
     </bpel:partnerLinks>
     <bpel:variables>
        <bpel:variable
messageType="ns1:HumanActivityRequest"
name="InceptionRequest"/>
        <bpel:variable
messageType="ns1:HumanActivityRequest"
name="ElaborateAnalysisModelRequest"/>
              .......
     </bpel:variables>
     <bpel:flow>
        <bpel:links>
           <bpel:link name="L1"/>
           <bpel:link name="L2"/>
              .......
        </bpel:links>
        <bpel:receive      createInstance="yes"
name="StartInception"
operation="HumanActivityRequest"
partnerLink="HumanActivity"
portType="ns1:WorkflowAdministrationPT"
variable="InceptionRequest">
           <bpel:sources>
              <bpel:source linkName="L1"/>
           </bpel:sources>
        </bpel:receive>
        <bpel:invoke
inputVariable="ElaborateAnalysisModelRequest"
name="ElaborateAnalysisModelRequest"
operation="HumanActivityRequest"
partnerLink="HumanActivity"
portType="ns1:WorkflowAdministrationPT">
  <bpel:targets>
         <bpel:target linkName="L1"/>
       </bpel:targets>
           <bpel:sources>
              <bpel:source linkName="L2"/>
           </bpel:sources>
        </bpel:invoke>
        <bpel:receive
name="ElaborateAnalysisModelResponse"
operation="HumanActivityResponse"
partnerLink="HumanActivity"
portType="ns1:WorkflowAdministrationPT"
variable="ElaborateAnalysisModelResponse">
           <bpel:targets>
              .......
        </bpel:receive>
                    .......
     </bpel:flow>
  </bpel:process>
```

Listing 1. A Sample of the generated BPEL code

6. Contributions & Conclusions

Whether WS-BPEL provides a rich set of concepts for executing processes, it lacks of the abstraction and expressiveness needed in modeling human-readable and

understandable process definitions. Its deficiency in supporting some workflow patterns, the lack of graphical notation and its no support for human interactions and arbitrary cycles makes it inappropriate for the modeling and understanding of software processes. On the other hand, our UML2.0-based language for software process modeling namely, UML4SPM, provides a high level of abstraction, expressiveness, notation and a set of elements and concepts with executable semantics; however it lacks of enactment support. In this paper we demonstrated how the two languages are combined in order to complement each other and to fully support both process modeling and execution. We defined a set of mapping rules between UML4SPM and WS-BPEL and we proposed to reuse the BPEL4PEOPLE proposition in order to deal with human interactions. The mapping rules we proposed are not only UML4PSM-to-BPEL specific since all rules that deal with UML2.0 concepts can be reused by any UML2.0-Based language or profile for software or business process modeling.

However, even if this approach presents the advantage of leveraging existing BPEL process engines and takes advantage of the execution support, it still suffers from some issues. The first one deals with the fact that during the transformation process all the aspects and semantics proper to software process activities (roles, guidance, deadlines, etc) are lost or scattered as BPEL variables. The only concepts that have equivalents in BPEL are those that deal with the sequencing of activities, events headlining, etc and which already have executable semantics (i.e.,UML2.0 *Activities* and *Actions*). This has as direct effect the loss of data needed for process measurement and improvement. Another issue is that process modeler has to choose the right concepts, which can be mapped in BPEL while modeling the process, otherwise there will be no support for them. Finally, the last issue relates to the fact that the generated BPEL is not usable straightforward after the transformation. A configuration step is needed in order to set *Partner Link* properties (service locations that have to be combined). This step can be automated during the transformation and process modeler would be asked to enter these information However, if the process modeler adds new elements or variables for execution aims after the transformation, this would raise the issue the issue of traceability between UML4SPM process definition and the generated BPEL process, and how coherence between the two definitions can be preserved.

This approach is currently evaluated within the MODELPLEX [MODELPLEX 06] project as well as in the MDDi Eclipse project (http://eclipse.org/mddi). Future perspectives of this work are the formalization of the transformation (currently in Java) by means of well-established model transformation languages such as ATL [ATL 06] or QVT [OMG 05b]. This will reduce human intervention and ambiguities due to multiple mappings that one UML4SPM element may have to BPEL. In addition, the support of OCL2.0 as a language for the specification of Pre and Post condition is underway.

7. References

[ActiveBPEL] ActiveBPEL at http://www.active-endpoints.com/active-bpel-engine-overview.htm, last time page visit: February 2007

[ATL 06] Atlas Group LINA and INRIA Nantes, Atl: Atlas transformation language. atl user manual, February 2006.

[Bendraou 05] Bendraou R., Gervais M.P, Blanc X. *"UML4SPM: A UML2.0-Based Metamodel for Software Process Modelling"*. In Proc. of the ACM/IEEE 8th Intern. Conf. on Model-Driven Engineering Languages and Systems (MoDELS'05), Montego Bay, Jamaica, Oct. 2005, LNCS, Vol. 3713, 17-38

[Bendraou 06] Bendraou R., Gervais M.P, Blanc X., *"UML4SPM: An Executable Software Process Modeling Language Providing High-Level Abstractions"*, Tenth IEEE Intern. EDOC Conf. (EDOC 2006), IEEE Computer Society Press, Oct. 2006, pp. 297-306

Bordbar B., Staikopoulos A.: "On Behavioural Model Transformation in Web Services", Proc. of the ER 2004 Workshops CoMoGIS, COMWIM, ECDM, CoMoA, DGOV, and ECOMO, Shanghai, China 2004, Springer.

[Chou 02] S.-C. Chou, *"A process modeling language consisting of high level UML diagrams and low level process language"*, Journal of Object Technology 1, 2002, 4, pp. 137–163

[Di Nitto 02] Di Nitto E. et at. *"Deriving executable process descriptions from UML",* in Proc. of the 24th Inter. Conf. on Software Engineering (ICSE'02), Florida, USA, 2002, ACM Press.

[Dobson 06] Dobson G., *"Using WS-BPEL to Implement Software Fault Tolerance for Web Services"*, in Proceedings of the 32nd EUROMICRO-SEAA'06 conference, IEEE Computer Society, 2006.

[Jäger 98] Dirk Jäger, Ansgar Schleicher, and Bernhard Westfechtel. *"Using UML for Software Process Modeling"*. Number 1687 in LNCS, pages 91-108, 1998.

[Kloppmann 05] Kloppmann, M. et al. *"WS-BPEL Extension for People BPEL4People"*, Joint white paper, IBM and SAP, July 2005.

[Korherr 06] Korherr B. and List B. "*Extending the UML 2 Activity Diagram with Business Process Goals and Performance Measures and the Mapping to BPEL*" in Proc. Of UML (BP-UML'06), Nov. 2006.

[Mantell 05] Mantell, K. *"From UML to BPEL"*. URL: http://www.ibm.com/developerworks/webservices/library/ws-uml2bpel, September 2005.

[MODELPLEX 06] MODELPLEX IST European Project, Contract N° IST-3408 *at http://www.modelplex-ist.org, page last visit: Feb. 20, 2007*

[OMG 02] OMG SPEM1.1, *"Software Process Engineering Metamodel"*, OMG document formal/02-11/14, November 2002, at http://www.omg.org.

[OMG 05a] OMG UML2..1.1 Superstructure, *"Unified Modelling Language", OMG document formal/07-02-03, Feb. 2007.*

[OMG 05b] OMG MOF QVT final adapted specification, ptc 05-11-01 (2005).

[Ouyang 06] Ouyang, C., Dumas, M., Breutel, S., ter Hofstede, A.H.M.: *"Translating Standard Process Models to BPEL"*. In Pohl, K., ed.: 18th Conf. on Advanced Information Systems Engineering, Luxembourg, Springer (2006) forthcoming

[Rumpe 02] Rumpe, B.: *"Executable Modeling with UML. A Vision or a Nightmare?"* In: Issues & Trends of Information Technology Management in Contemporary Associations, Seattle. Idea Group Publishing, Hershey, London, pp. 697-701. 2002.

[Van der Aalst 03]Van der Aalst W.M.P. *"Don't go with the flow: Web services composition standards exposed"*. IEEE Intelligent Systems, 18(1):72-76, 2003.

[Van der Aalst 03] Van der Aalst, W.M.P., ter Hofstede, A.H.M., et al..: "*Workflow Patterns"*. Distributed and Parallel Databases 14 (2003) 5–51

[WSBPEL 07] Web Services Business Process Execution Language Version 2.0. Working Draft. WS-BPEL TC OASIS, January 2007. URL: http://www.oasis-open.org/committees/download.php/12791/

[WfP] Workflow Patterns at www.workflowpatterns.com

[Wohed 04] Wohed P. et al.., *"Pattern-based Analysis of the Control-Flow Perspective of UML Activity Diagrams"*, in L. Delcambre et al., editors, Proc. of the 24th Int. Conf. on Conceptual Modeling (ER 2005), volume 3716 of LNCS, pages 63-78. Springer-Verlag, Berlin, 2005