On Behavioural Model Transformation in Web Services

Behzad Bordbar and Athanasios Staikopoulos

School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK
B.Bordbar@cs.bham.ac.uk, A.Staikopoulos@cs.bham.ac.uk

Abstract. Web Services are seen as one of the most promising solutions for the integration of autonomous, heterogonous e-business systems. Today's e-commerce systems often involve a combination of multiple Web Services, which are implemented via a mix of technologies such as Business Process Markup Language (BPML), Business Process Execution Language for Web Services (BPEL4WS), and Web Service Choreography Interface (WSCI).

Recently, the application of Model Driven Architecture (MDA) to Web Services has received considerable attention. However, most of existing literature deals with the static aspects of Web Service modelling. This paper focuses on the behavioral aspect of the composition of Web Services using a Meta Object Facility (MOF) compliant metamodel for BPEL4WS. The paper presents a transformation of the Unified Modelling Language (UML) Activity diagram to the BPEL4WS.

1 Introduction

In recent years, the Internet has evolved from a simple storage of information into a provider of different kind of e-commerce services, ranging from travel booking, shopping to more complex e-business systems involving complex transactions. One of the main challenges of the design of such systems is to integrate autonomous, heterogonous and distributed components [21]. Currently, *Web Services* [26] are seen as one of the most promising approaches to solve the above problems [20][21]. Web Services are a set of technologies that allow applications to communicate with each other in a platform and a programming-language independent manner. Extensible Mark-Up Language (XML) [23], Simple Object Access Protocol (SOAP) [24], and Web Service Description Language (WSDL) [27] are among the technologies used in Web Service. Developing Web Services by applying Model Driven Architecture (MDA) [10][12][17] has recently received considerable attention [4][8][12]. In particular, [4][12] study the transformation of Web Services and present a set of case studies involving the transformation of Web Services models to various implementation platforms such as Java, WSDL and Enterprise Distributed Object Computing (EDOC) [14]. However, most of existing research focuses on the transformation of models that express the static structure of the system, i.e. models describing what the system contains and how various parts are related together. In this paper, we shall study transformations, which deal with the *dynamic* aspects of the system, which are modeled via *behavioral models,* expressing the way the various components collaborate in order to manage a task and fulfill the system functions.

Defining and supporting business processes and collaborations between Web Services is a more complex problem than defining and supporting individual Web Services, see page 43 in [8]. Currently, the composition of Web Services is carried out via a mix of concrete technologies such as BPML [5], BPEL4WS [2], WSCI [25] and other [8]. Considering the existence of various technologies and languages, there is a clear scope for defining the business processes via behavioral Platform Independent Models (PIM) and providing methods of translation of PIM to Platform Specific Models (PSM) [8][10]. In this paper we shall present a method of transformation of the behavioural models from the UML Activity diagram to BPEL4WS.

The paper is organized as follows. We shall start by a brief introduction on Web Services, business processes and the model transformations in the MDA. Section 3 discusses the transformation of Business processes from Activity diagrams to BPEL4WS. First, we shall presents an example of a Stock Quote Web Service, which serves as our running example. The behavioural aspect of the Stock Quote system is modeled as a UML Activity diagram. Next, we shall sketch a metamodel for the UML Activity diagram and BPEL4WS. To translate the Activity diagrams to BPEL4WS, a set of transformations is introduced. The final part of section 3 applies the transformations to the running example. Section 4 reflects on the lessons learnt and discusses some of the issues regarding the model transformation of the behavioural aspects of systems. Finally, section 5 presents a conclusion.

2 Preliminaries

The Web Services introduce a new paradigm for enabling the exchange of information across the Internet. They can be characterised as self contained, self-describing that can be published, located and invoked across the Web. Various e-business applications can be encapsulated and published as Web Services allowing them to interoperate through standard communication and messaging mechanisms [26]. In general, Web Services are based on the Extensible Mark-Up Language (XML) [23] as the fundamental mechanism for describing protocols, structured data and messages, the Web Service Description Language (WSDL) [27] for describing the exposed interfaces and operations of a Web Service and the Simple Object Access Protocol (SOAP) [24] for providing the communication protocol.

Business integration and collaboration requires more than the ability to conduct simple interactions between Web Services. This has resulted in the creation of the notion of *business process*. A business process can be viewed as a composite activity, which defines a complicated behaviour expressed as a workflow [9]. Workflow specifies the control and data flow among sub activities of an activity. There are several different implementations for representing business processes [13], like the Business Process Execution Language for Web Services (BPEL4WS) [2]. The BPEL4WS provides an XML notation and semantics for specifying business process behaviour based on Web Services and defines, how Web Services can be combined to implement a business process.

In MDA each model is based on a specific metamodel, which defines the language that the model is created in. All metamodels are based on a unique metamodel called Meta Object Facility (MOF) [15]. As a result, model transformations can be carried via defining Transformation Rules between two MOF compliant metamodels [4][8][10], the source and the destination. In this paper, the source metamodel is the UML Activity modelling language and the destination is the BPEL4WS. Fig. 1 depicts the use of transformation rules for model transformation [4]. The transformation rules define a mapping between a source and a destination metamodel that preserves equivalent or isodynamic *(similar)* semantics. A transformation engine executes the transformation rules on the source model (acting as the input) in order to generate its equivalent destination model (output).

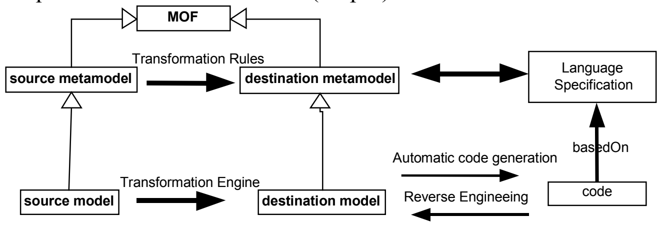


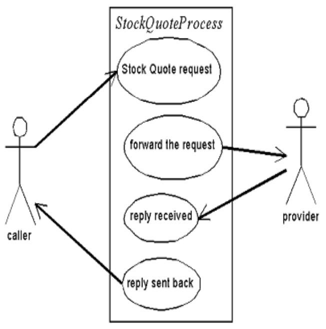
Fig. 1. Transformation in the MDA based on [4]

3 Business Processes Transformation in Web Services

This section demonstrates how a general business process can be modelled as a UML Activity diagram and how it can be mapped to an equivalent MOF based model representing a BPEL4WS. We shall start by presenting an example of a business process, which serves as our running example.

Example: Fig. 2 depicts two UML actors and a business process. The actors, *caller* and *provider*, represent two roles played by Web Services. They are composed together with the help of the *StockQuoteProcess* to create a composite Web Service. The *caller* makes a Stock Quote request to the business process. The process receives the request and forwards it to the *provider*. The *provider* replies with the value of the requested stock quote, which the process sends back to the *caller*.

Fig. 3 depicts a UML Activity Diagram model of the process in terms of workflow, coordination and interaction between the involved Web Services. It can be seen that the three services have been separated via Activity diagram swimlanes [19] and are stereotyped accordingly to their roles. In addition, the service location is indicated by the *external* stereotype regarding the business process's perspective. Next, five tasks *receive request*, *prepare invocation call*, *invoke provider service*, *prepare response message*, and *reply* are modelled as actions and are connected in a sequential order within a process. This order can be regarded as the execution path for handling the *caller*'s request. Furthermore, there are four variables involved; *request*, *invocation* *request*, *response*, *invocation response*, which are depicted as *object flow*, indicating how objects can be passed around via operation calls.



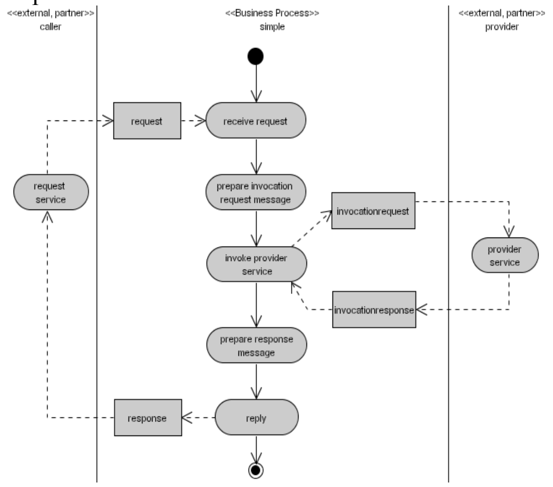


Fig. 2. A stock quote business process

Fig. 3. Activity diagram for the process

UML Metamodel for activity diagram: Metamodels play an important role within MDA, as they provide the language and the rules for describing models**.** Fig. 4 depicts a part of the metamodel for Activity diagram, which includes metamodel elements representing *workflow*, *object flow*, *activities*, *actions*, *operation calls* and other various modelling elements expressing control nodes. To create this metamodel, we have abstracted and combined various activities and action modelling elements defined in the UML 2.0 Superstructure Specification [19].

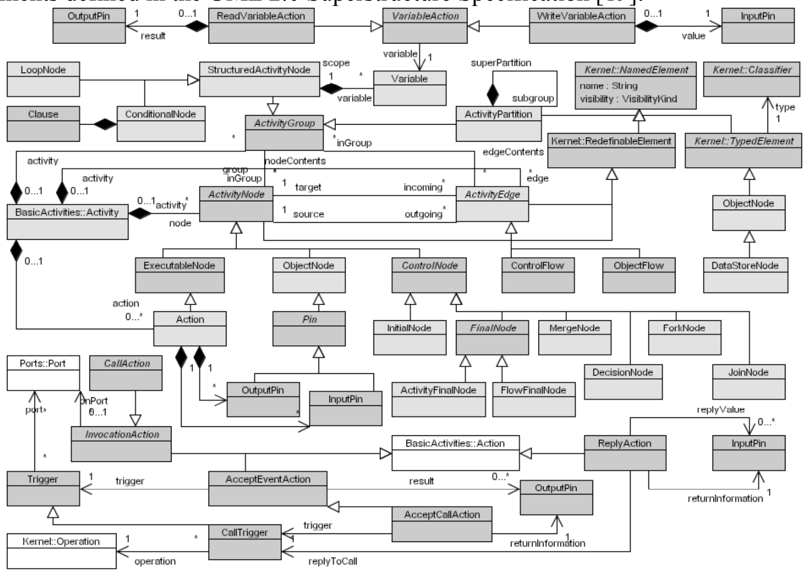


Fig. 4. UML Activity diagram metamodel

Metamodels create a clear view of available model elements. As a result, using the metamodel, it is possible to refine a model and provide a more elaborate representation. For example, consider the *receive request* action of Fig. 3. There are various types of *actions* specified in the metamodel of Fig. 4. For example, *receive request* can be modelled as an *AcceptCallAction* meaning the receipt of a synchronous call request. Such model refinements is a result of implementing new requirements of the system or by making additional assumptions regarding the model elements involved, such as *action types*, *sub activities* and *variables*. Consequently, we have refined the activity diagram of Fig. 3 to the new activity diagram of Fig. 5 by including a set of stereotypes. We have also included *sub activities* to illustrate the internal *variable* manipulation through read and write actions. For more information regarding the Activity model elements refer to chapter 12 of [19].

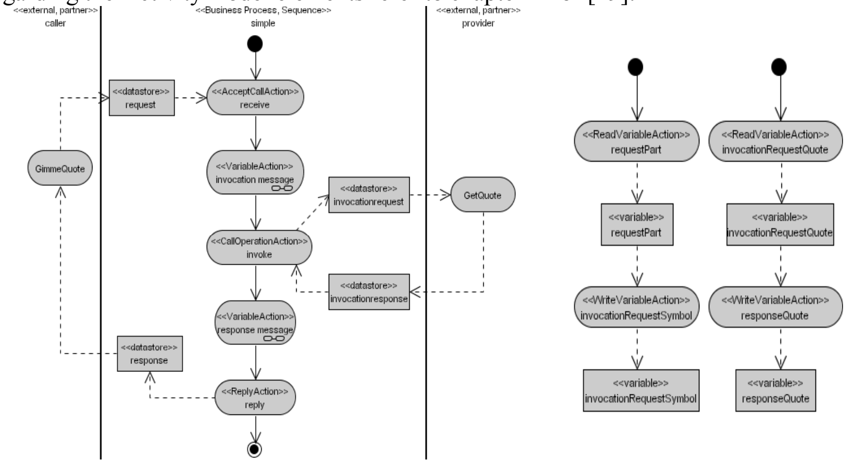


Fig. 5. The refined Activity diagram

BPEL4WS Metamodels: I this section we shall present the metamodel for BPEL4WS; the destination language, see Fig. 1. The BPEL4WS can be seen as an extension of the WSDL [27] supporting the collaboration of Web Services. In other words, the WSDL describes stand alone Web Services and their structure. Thus UML class diagrams, expressing the static aspects, are sufficient to capture and represent the semantics of the WSDL. Currently, there are approaches [4] for specifying and mappings metamodels among UML static aspects with its relevant WSDL elements. Our research can be seen as an extension of the existing work on static modelling by WSDL, to include methods of the mapping of the dynamic aspects.

Fig. 6 depicts the metamodel for BPEL4WS based on BPEL4WS version 1.1 specification and the XML Schema as published in [2]. Since, BPEL4WS is dependant upon a number of WSDL elements, there are references to *port types*, *messages* and *operations*, which are WSDL model elements. The central element of the metamodel is the business *process,* which captures both the structure and the workflow (controlled order of actions) of a business model.

A *process* consists of a number of *variables* for holding messages and representing the states of a process. A *partner links* is used to identify the associated Web Services through their *roles* and their *port types.* A *port type* represents the exposed interfaces of a Web Service via the WSDL specification. The metamodel also includes *fault handlers* for dealing with errors and *event handlers* for reacting to the events triggered. The BPEL4WS also includes *sequence*, *flow*, *while, switch and scope* as *structured activities*. *Sequence* activity defines sequential execution of actions. *Flow* provides parallel execution and synchronisation between actions. *While* specifies iterative activities. *Switch* supports conditional behaviour *and Scope* defines the execution context in which local variables and handlers can be defined for an action. Other basic activities for calling Web Service operations are: *Receive* to provide the business process services to Web Service partners. *Reply* to answer an accepted request. *Invoke* to either perform a request/response or a one way operation on partner Web Services. *Assign* to copy message parts and data from one variable to another. For further details, we refer the reader to the BPEL4WS specification [2].

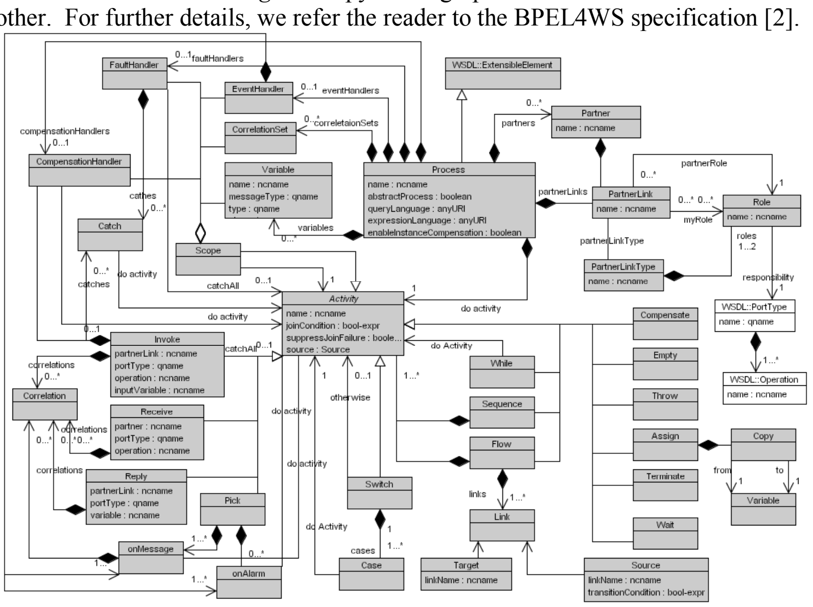


Fig. 6. A metamodel for BPEL4WS

Transformation Mapping: A model transformation, which defines the generation of a target model from a source model, is described by a transformation definition, consisting of a number of transformation rules that are executed by a transformation CASE tool. There are various methods of specifying model transformation [1][4][10]. Currently, there is no standardised language for defining transformations definitions. Consequently, the OMG has issued a request for proposal for a new standard called, *Query Views and Transformations* (QVT) [18], which has received considerable attention [17]. In this paper, we demonstrate the mapping among meta-elements with a table and express their transformation rules with the Object Constraint Language (OCL) [16]. Table 1 depicts a mapping, between the model elements of the metamodel of the Activity diagram and BPEL4WS 1.1. The mapping is based upon the descriptions given in the specifications [19] and [2]. Because of space restrictions, we have only included the relative elements demonstrated in the example

UML 2.0	*BPEL 1.1*
Class as BehavioredClassifier, Activity, StructuredActivityNode	**<process>**
A BehavioredClassifier is a class acting as a container and where behaviour specifications can be defined. It is also an activity that is expressed as a flow of execution via subordinate units.	The BPEL4WS process description.
Datastore, StructuredNode Variable, ObjectNode, Class Attributes	**<variable>**
A datastore node is a central buffer node for non-transient information. An object node is an abstract activity node for defining object flow. Variables are elements for passing data between actions indirectly.	They provide the means for holding messages that constitute the state of a business process and the exchanging of messages between partners.
ControlFlow	**<sequence>**
It contains one or more activities that are performed sequentially in the order. It completes when the final activity has been completed	An activity that contains one or more other activities and which executes them in a serial order.
AcceptCallAction	**<receive>**
Is an accept event action representing the receipt of a synchronous call request. The action produces a token that is needed later to supply the information to the ReplyAction.	A business process provides services to its partners through receive activities and corresponding reply actions.
ReadVariableAction , WriteVariableAction	**<assign>**
Variable actions support the reading and writing of variables. The VariableAction metaclass specifies the variable being accessed.	It copies data from one variable to another.
Variable , ObjectNode , OutputPin	**<from>**
An output pin is a pin that holds output values produced by an action. Output pins are object nodes that deliver values to other actions .	It specifies the source variable name to be used for an assignment expression.
Variable, ObjectNode , InputPin	**<to>**
An input pin is a pin that holds input values to be consumed by an action. They are object nodes that receive values from other actions	It specifies the target variable name to be used for an assignment expression.
CallOperationAction	**<invoke>**
An action that transmits an operation call request to the target object. If the action is marked asynchronous, the execution of the call operation waits until the execution completes, otherwise it is completed when the invocation is established.	An activity that is used by a process to make invocation to Web Services provided by partners. Can be either synchronous (request/reply) or asynchronous (one way).
ReplyAction	**<reply>**
An action that accepts a set of return values and a token containing return information produced by a previous accept call action.	An action to send a response to a request previously accepted through a receive activity.

Table 1. Mapping elements of the activity diagram to BPEL4WS

Transformation Rules: The transformation method adopted is based on [10] and makes use of the OCL with the following conventions: The *UML* refers to the source language (Activity diagram), the *BPEL4WS* refers to the target language, the *params* refer to any parameters used during the transformation, the *source* and *target* refer to various named source or target meta-language model elements, the source and target conditions refer to source or target language conditions that must hold in order to apply the rule, the *mapping process* performs the mapping among the model elements, and -- for various comments.

The transformation process is initiated by performing a mapping between the overall UML Activity diagram and a business process. The transformation *UMLActivity2BusinessProcess* (see Table 2), firstly, maps the *activity name* into a *process* name. Then it performs three nested sub-transformations named as *UML Datastore2BPVariable*, *ActivityPartition2BPPartnerLink* and *UMLProcessActivity2 BPActivity*. The first sub-transformation maps the *variables* or *datastores* used in the UML activity diagram into the corresponding entities in the business processes. The UML Activity diagram uses *partitions* and *swimlanes* to represent *partners*. The sub-transformation *ActivityPartition2BPPartnerLink* maps such *partitions* into to BPEL4WS *partner links*. Finally, the sub-transformation *UMLProcessActivityBP Activity* (see Table 3) maps the UML *Process Activities* into *Business Process Activities*.

Transformation ***UMLActivity2BusinessProcess*** (UML, BPEL4WS)	
`params`	`srcActivity: UML::Activity -- the UML source Model`
`source`	`srcActNodes: OCL::Set(ActivityNode)`

`target`	`trgProcess: BPEL4WS::Process`
	`trgVariable: BPEL4WS::Variable`
	`trgPartnerLink: BPEL4WS::PartnerLink`
	`trgActivity: BPEL4WS::Activity`

`source cond`	`srcActNodes = srcActivity.nodes->asSet()->union(`
	`srcActivity.group->collectNested(ActivityNode))`

`--mapping process`
`srcActivity.name <~> trgProcess.name`
`try` **`UMLDatastore2BPVariable`** `on srcActNodes->collect(DataStore)`
`<~> trgVariable.type`
`try` **`ActivityPartition2BPPartnerLink`** `on srcActNodes->`
`collect(ActivityPartition) <~> trgPartnerLink.type`
`try` **`UMLProcessActivity2BPActivity`** `on srcActNodes->`
`collect(Action) <~> trgActivity.type`	

Table 2. The *UMLActivity2BusinessProcess* Transformation

When mapping the *ProcessActivity* to a *BusinessProcess activity* we need to check the *activity* type, which is stereotyped in our model of Fig. 5 in order to trigger the appropriate mapping. In our example, the process activity is of the type *sequence,* therefore the *UMLSequence2BPSequence* (see Table 4) rule is called. If the activity was of type *flow,* then the mapping rule *UMLFlow2BPFlow* would have been applied.

Transformation ***UMLProcessActivity2BPActivity*** (UML, BPEL4WS)	
`params`	`srcActions: OCL::Set(UML::Action)`
`source`	`srcActivity: UML::Activity`
`target`	`trgSequence: BPEL4WS::Sequence`

	trgFlow: BPEL4WS::Flow

```
--mapping process
if srcActivity.oclIsTypeof(Sequence) then
try UMLSequence2BPSequence on scrActions <~>   trgSequence.type
elseif srcActivity.oclIsTypeof(Flow) then
try UMLFlow2BPFlow on scrActions <~> trgFlow.type
endif
```

Table 3. The *UMLActivity2BusinessProcess* Transformation

Following, we specify the mapping between *sequential actions* in the UML and BPEL4WS. First the *actions* within a *sequence* activity are passed as parameters. The method adopted involves inductive transformation of the model elements of the UML sequence. Starting from the *starting action* (the one that initiates the sequence of actions), each model element is interpreted and transformed into its corresponding model element at the destination. The transformation procedure continues until, the final action is reached. For example, if the type of *outgoing.target,* see the table below, is an *AcceptCallAction*, we have to apply the *AcceptCallAction* to BPEL4WS *ReceiveActivity* mapping rule. Similarly, it is possible to deal with other target types.

Transformation ***UMLSequence2BPSequence*** (UML, BPEL4WS)	
params	`srcActions: OCL::Set(UML::Action) )`
source	`srcInitActions: OCL::Set(UML::Action) –-` *`possible set of starting actions`*
target	`trgActReceive: BPEL4WS::Receive` `trgActInvoke: BPEL4WS::Invoke` `trgActReply: BPEL4WS::Reply` `trgActAssign: BPEL4WS::Assign`
source cond	`srcInitActions  = srcActions.iterate(a:Action acc:Set(ActivityNode))=Set{} |   if a.incoming.isEmpty() then     acc->including(a)) && srcInitActions.count=1`

```
--mapping process
action = srcInitActions
Do while action <> null
If initAction.outgoing.target.oclIsTypeOf(AcceptCallAction)
  try UMLAcceptCallAction2BPReceiveActivity on
action.outgoing.target.type <~> trgActReceive.type
elseif
initActin.outgoing.target.oclIsTypeOf(VariableAction)
  try UMLAssign2BPAssignActivity on action.outgoing.target.type
<~> trgActAssign.type
elseif
initActin.outgoing.target.oclIsTypeOf(CallOperationAction)
  try UMLCallOperationAction2BPInvokeActivity on
action.outgoing.target.type <~> trgActInvoke.type
elseif initActin.outgoing.target.oclIsTypeOf(ReplyAction)
  try UMLReplyAction2BPReplyActivity on
action.outgoing.target.type <~> trgActReply.type
endif
action = action.outgoing.target
loop
```

Table 4. The *UMLSequence2BPSequence* Transformation

The above transformations are samples of the developed rules. Even though, the above sets of rules are sufficient to illustrate the model transformation of our running example. If we apply the transformation rules to the Activity diagram of Fig. 3 or Fig. 5, an equivalent BPEL4WS activity model of Fig. 7 is produced.

4 Discussions and Related Work

The UML Activity diagram of Fig. 5 and Fig. 7 seem very similar, as they represent the same case scenario. From the conceptual point of view, they belong to totally different metamodels and have different properties. In addition, we found that when more complex behaviours are applied (such as representing *scopes* and *repetitions*) the models become more dissimilar.

We have a view of model transformation, which includes the transformation of both static and dynamic aspects of the system. Following the notation of [6][7], suppose that $m_1(s)/f_1$ ($m_2(s)/f_2$) represent static aspects of the system s as models m_1(m_2) in formalisms f_1 (f_2), respectively. Assume that $m_1(s)/f_1 \rightarrow m_2(s)/f_2$ is a transformation that maps static aspects of the system from the formalism f_1, the source, to the formalism f_2, the destination. Suppose that g_1 and g_2 are two formalisms for expressing dynamic aspects of the system s, i.e. they can be used to specify how various components cooperate to manage various tasks and provide functions of the system. As a result, the transformation $m_1(s)/g_1 \rightarrow m_2(s)/g_2$ are used to map the dynamic aspects of the system. Since the static and dynamic aspects of a system are closely interrelated, it is naïve to assume that the transformation of the two aspects can be carried out independent of one another. For example, in our case study, the metamodel of BPEL4WS shown in Fig. 6 contains references to WSDL elements; there are references to *port type*, *message* and *operation*, which are static model elements. In other words, since the dynamic aspects represent how entities (defined in the structural models) work together to accomplish a task, it is essential to find a systematic way of integrating the two views together. In our opinion, presenting a general method of integrating static and dynamic aspects of arbitrary systems is a highly non-trivial and challenging task. As a result, our current research focuses only on the transformation of behavioural models of Business Processes, from a UML model to a specific implementation technology (BPEL4WS). In conducting the above case study, we have followed the following trivial rule of thumb:

"If a business process task t1 at the source is transformed to a task t2 at the destination, then t2 and t1 must have the same effect on the corresponding collaborating services.''

Would it be possible to formalise the above rule of thumb? Caplat and Sourroulle [6] study a similar question for the static aspects of systems. For example, they show that, similarity of metamodels is not a good criterion for judging the similarity of formalisms. However, answering the above question require further research.

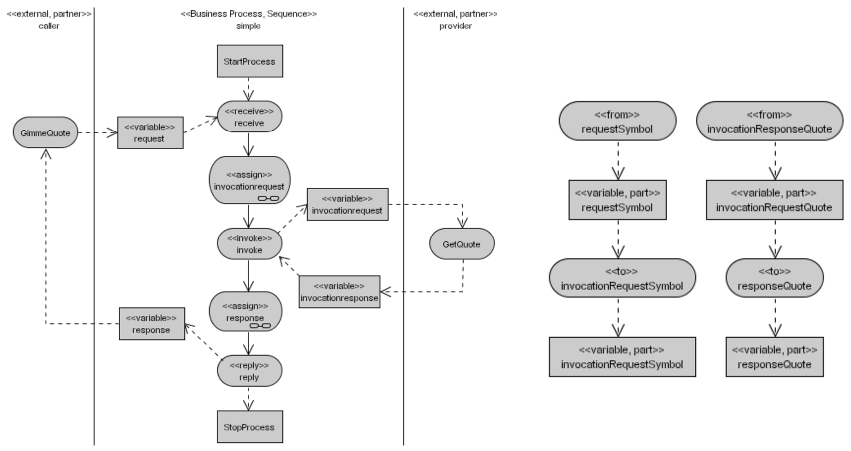


Fig. 7. Activity Diagram for BPEL

The Activity diagram of Fig. 3 is a high level conceptual model, which is too abstract to be translated to an implementation. As a result, we incorporated further information, which required making further design decisions. For example, refine the diagram of Fig. 3 to the Activity diagram of Fig. 5. The metamodel can help us in identifying various possible ways of refining a model. For instance, the metamodel of Fig. 4 specifies that an *ActivityNode* is either a *ControlNode*, *ObjectNode* or *ExecutableNode*, see page 268 in [19].

5 Conclusion

This paper deals with the modelling of the behavioural aspects of composite Web Services. The paper studies a model transformation of business processes from a PIM, created as a UML Activity diagram, into a PSM, modelled via BPEL4WS. We have presented a metamodel for the UML Activity diagram, which can be used for the refinement of the conceptual models of business processes. We have also introduced a MOF compliant metamodel for the BPEL4WS. A mapping between the corresponding model elements in the UML Activity diagram and BPEL4WS is introduced. To translate Activity diagram to BPEL4WS, we have introduced a set of transformation rules, which are specified in the OCL. Finally, we have applied our approach to the model transformation of a Stock Quote Web Service.

References

1. Appukutan, B., Clark, T., Reddy, S., Tratt, T., Venkatesh, R.: A model driven approach to model transformations, Kings College, (2003)
2. BEA, IBM, Microsoft, SAP AG and Siebel Systems: Business Process Execution Language for Web Services, Version 1.1, (May 2003)

3. Bezavin, J., Gerard, S.: A preliminary identification of MDA components, University of Nantes, CEA, (2002)
4. Bezivin, J., Hammoudi, S., Lopes, D., Jouault, F.: An Experiment in Mapping Web Services to Implementation Platforms, Atlas Group, INRIA and LINA University of Nantes, Research Report, (March 2004)
5. BPMI: Business Process Modelling Language (BPML), Business Process Management Initiative, (November 2002)
6. Caplat, G., Sourrouille, J. L.: Considerations about Model Mapping, Workshop on MDA, INSA, (2003)
7. Caplat, G., Sourrouille, J. L.: Model Mapping in MDA, INSA, France, (2002)
8. Frankel, D.S.: Model Driven Architecture, Model Driven Architecture: Applying MDA to Enterprise Computing, OMG Press, ISBN: 0471319201,(January 2003)
9. Ganesarajah, D., Lupu, E.: Workflow-based composition of Web Services, Department of Computer Science, Imperial College, (2002)
10. Kleppe, A., Warmer, J., Bast, W.: MDA Explained. The Model Driven Architecture: Practice and Promise, Addison-Wesley, (April 2003)
11. Kreger, H.: Fulfilling The Web Services Promise, Communications of the ACM, Vol. 46, No. 6, (June 2003)
12. Lopes, D., Hammoudi, S.: Web Services in the Context of MDA, University of Nantes, France, (2003)
13. O' Riordan, D.: Business Process Standards For Web Services, Published by Tect, Chicago, USA
14. OMG: Enterprise Collaboration Architecture (ECA) Specification, Object Management Group, Version 1.0, (February 2004)
15. OMG: Meta Object Facility (MOF) Specification, Object Management Group, Version 1.4, (2002)
16. OMG: Object Constraint Language Specification (OCL), Part of UML 1.4.1 Specification, (2003)
17. OMG: Object Management Group, Available from http://www.omg.com
18. OMG: Request for Proposal: MOF 2.0 Query / Views / Transformations RFP, Object Management Group, (October 2003)
19. OMG: UML 2.0 Superstructure Specification, Object Management Group, Adopted Specification, Version 2, (August 2003)
20. Papazoglou, M.P.,Georgakopoulos, D.: Service Oriented Computing, Communications of the ACM, Vol. 46, No. 10, (October 2003)
21. Siegel, J.: Using OMG's Model Driven Architecture (MDA) to integrate Web Services, Object Management Group White Paper, (November 2001)
22. Tratt, L., Clark, T.: Model Transformation in Converge, Kings College, (2003)
23. W3C: Extensible Markup Language (XML) 1.0, Third Edition, W3C Recommendation, (February 2004)
24. W3C: Simple Object Access Protocol (SOAP), Version 1.2, W3C Recommendation, Available from http://www.w3.org/TR/soap12-part1, (2003)
25. W3C: Web Service Choreography Interface (WSCI) 1.0, W3C Note, Available from http://www.w3.org/TR/wsci, (August 2002)
26. W3C: Web Services Architecture, Working Group Note, (February 2004)
27. W3C: Web Services Description Language (WSDL) Version 2.0, W3C Working Draft, (2003)