

Structuring Business Nested Processes Using UML 2.0 Activity Diagrams and Translating into XPDL

Barbara Gallina and Nicolas Guelfi and Amel Mammar
Software Engineering Competence Center
Faculty of Sciences, Technology and Communication
University of Luxembourg, L-1359 Luxembourg-Kirchberg, Luxembourg
{barbara.gallina, nicolas.guelfi, amel.mammar}@uni.lu

Abstract: Engineering complex distributed business processes necessitates an integrated use of modeling, verification and validation techniques. This paper presents a pragmatic approach for such purpose. In particular we present complex business processes modeled using transactions described with UML 2.0 activity diagrams, in which we introduce hierarchy as a structuring primitive and exception handling as a fault tolerance mechanism. The structuring capabilities of our approach allow designers to tackle the complexity of large business transactions, the exception handling mechanism introduces a novel way to design transactions and to cope with exceptional behaviors inherent to BPM. Since our transaction models need to be validated during the development process, we have chosen to integrate the modeling tool together with a workflow management system that will allow animation of the business transactions. This is done by model transformation from activity diagrams to XPDL descriptions. One of the other interesting aspects of our approach is that the UML 2.0 activity diagrams models are given in such a way that different execution environments can be targeted for deployment. These environments (MTS, CORBA, BTP,) differ mainly on the way they implement the ACID properties and on their underlying exception handling mechanism. A first prototype supporting this approach, but limited to XPDL 1, has been developed and we also present its possible extension to the new XPDL 2 standard.

Keywords UML 2.0 Activity Diagrams, XPDL, sub-flows, exception handling, ACID properties.

1 Introduction

This work is part of the Efficient research project of Henri Tudor research center (Luxembourg) in cooperation with our university. The project presents a three layered approach: a business, a specification and technical layer. At the business layer, transactions, modeling business processes, are identified; at the specification layer, these transactions are specified using UML; at the technical layer, the execution of the transactions, through an adequate tool called Animator, takes place. Animator is a toolset for designing, animating, validating and verifying trusted business transactions [DG01, Tud05]. At the specification layer, UML 2.0 Activity Diagrams (AD) have been chosen since we think they could represent a reasonable communication trade off between business logic experts and IT experts in

order to describe trusted business processes. Business logic experts simply need to express functional requirements of the system which they are thinking of. For IT experts, however, every requirement has to be verified and consequently the use of formal methods is important.

XPDL is a standardized language allowing process definitions interchange between a variety of tools ranging from workflow management systems to modeling and simulation tools. In the Efficient context, XPDL is the language used at workflow engine level in order to execute and animate processes to be able to validate them.

Several dimensions have to be considered in order to think about a complete notation for describing business processes. As deeply argued in [RtHEvdA04, vdAtHKB00], several typical design patterns characterise business processes. In this work however we are going to consider basic elements and patterns necessary in order to reach our goal: structuring complex distributed business processes, modeling them through UML 2.0 activity diagrams and translating them into XPDL in order to be able to validate their design through animation.

The rest of this paper is structured as follows. In section 2, some relevant workflow process peculiarities are presented; in section 3, a typical business workflow scenario is described using UML 2.0 activity diagrams. XPDL is briefly introduced in section 4 while in section 5, the translation rules are informally presented. Finally, in section 6, conclusions and future work are presented.

2 Basic workflow elements

A workflow process is defined as the automatic routing of documents to the users responsible for working on them. Workflows are concerned with providing the information required to support each step of the business cycle. These documents may be physically moved over the network or maintained in a single database with the appropriate users given access to the data at the required times. Triggers can be implemented in the system to alert managers when operations are overdue. Similarly, in Efficient project context, a business transaction is defined as the description of an exchange of a set of information to achieve a business goal, mostly the delivery of an output to a particular customer or market. The basic elements that constitute our business transactions are described in the following list.

- Tasks/activities may be elementary or composed of a set of activities. In case of composed activities hierarchical levels among them may be recognised. Processes are composed of activities that have to be executed in order to reach a goal.
- Documents/objects represent administrative documents, credit cards and so on. Process tasks are based on operations that are executed on objects.
- Decisions and parallelisms are quite crucial points in business processes since they allow business expert to express alternative (but still normal) paths and parallel paths.
- Exceptions are nowadays deeply investigated at both levels: modelling and workflow engines. Since it is hard to foresee in an exhaustive way the entire spectrum of

situations that could happen during the process execution, it is better to avoid freezing a system by trying to define all possible situations. Better is to define particular circumstances that diverge from the normal ones and try to define a behaviour that is useful to follow in those cases.

- Roles/participants represent the entity that is going to execute the process. Generally a role may be a person, a machine or a particular software. Sometimes the process execution may need the participation of more roles of different typologies at the same time. Roles (business partners) may cooperate in order to reach the transaction goal in normal and exceptional cases.

As far as we know, not all these elements have been already taken into consideration. In particular until now, among possible tasks, only elementary ones have been considered; moreover only normal process behaviors have been modeled. The contributions of this work are the consideration of composed activities and a first theoretical step to investigate how to represent transactions and exception handling mechanism in UML 2.0 activity diagrams and also in both XPDL 1.0 and XPDL 2.0.

3 UML 2.0 activity diagrams for nested business processes: a typical case-study

Activity diagrams, as means to describe business processes, have been chosen also in other previous works [GCR04]. We still choose activity diagrams since we think that they can be an efficient communication trade-off between business experts and IT experts but we focus our attention on UML 2.0 activity diagrams because of their more powerful expressiveness compared with those of UML 1.5.

In UML 2.0 substantial changes have been made [Obj03] if compared with UML 1.5, in particular the activity diagram metamodel subset has been redesigned from scratch. The main concept underlying activity diagrams is now called *Activity* and replaces *Activity-Graph* in UML 1.5. *Activity* is not a subclass of *StateMachine* any more. The metamodel defines six levels of increasing expressiveness. Since our aim is to express nesting business transactions in a distributed context and exception handling and since business transactions imply the exchange of messages, *IntermediateActivities* level is involved. In fact this level supports modeling of activity diagrams that include concurrent control and data flow. Moreover in order to use partitions we also use *CompleteActivities*.

SubactivityStates have vanished, and nesting is now accomplished by calling subordinate (enclosed) Activities from Actions defined in the enclosing context. These enclosed activities in UML 2.0 terminology are called *CallBehaviorAction*. An AD is a directed graph, consisting of nodes that are connected via directed edges. Nodes comprise action nodes, object nodes and control nodes.

Action nodes, as already mentioned, may invoke other behaviours and the related behaviour is still an activity diagram. All actions may receive parameters as input and return parameters as output that are subclasses of object nodes.

In this work, instead of extending UML 2.0 activity diagrams by adding a stereotype for

each XPDL activity type, as done in [JMN03], the effort is that to exploit as much as possible the standard AD elements and adding stereotypes only where they are necessary. Currently, however, UML doesn't support transactional concept. We mean that if we want to underline ACID properties at the moment we can not. In order to do that we have to stereotype an existing symbol by changing its semantics.

We are going to describe our extension by briefly re-introducing a typical business case-study (already illustrated in [GCR04]) involving a customer, a wholesaler and a manufacturer, detailing how ACID properties and exception handling mechanism (both forward and backward error recovery) could be designed. A business process in order to be modeled as a transaction should ensure ACID properties. As already mentioned, in UML 2.0 there is no way to express the transaction concept and in this work we are interested in doing a first step towards it because in order to talk about efficient capture of transactions, first of all, we have to ensure transactions quality. We then propose to model a nested process through a CallBehaviorAction Metaclass with stereotype Transaction in cases in which the underlying protocol assures (relaxed or not) ACID properties. In all other cases a nested process will be modeled through a CallBehaviorAction without any stereotype. A transaction may terminate normally (according to the expectations), exceptionally or aborting the effects. In cases in which an hazard happens we propose a way to also model this circumstance.

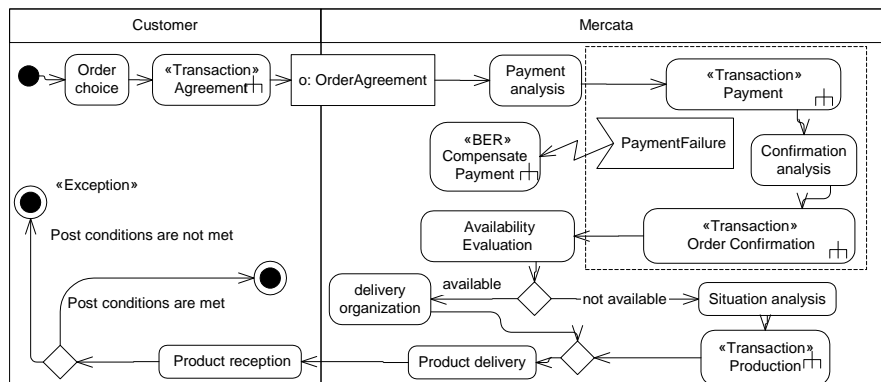


Figure 1: Highest level diagram of Mercata Case-Study.

In Figure 1 a business process, depicted through a UML 2.0 activity diagram, is presented. There are two roles Customer and Mercata. A customer after having chosen the order and agreed (successfully executed the sub-process Agreement) about the order details, sends the order to Mercata. The Payment process, a new sub-process, takes place. During this sub-process an exception may occur. This possibility is depicted by drawing an interruptible region around the part in which the exception is supposed to happen. If during Payment sub-process a Failure Exception occurs and we must roll-back the situation, apply Backward Error Recovery (BER), because we have no means to go on in another way, a special sub-process, a UML 2.0 CallBehaviorAction with stereotype BER, has to be called.

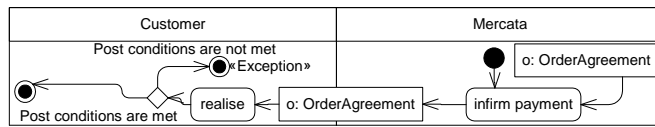


Figure 2: Compensation sub-process

In figure 2 the roll-back is shown: the payment is infirmed and the customer receives the notification. If Payment terminates successfully and the order is also successfully

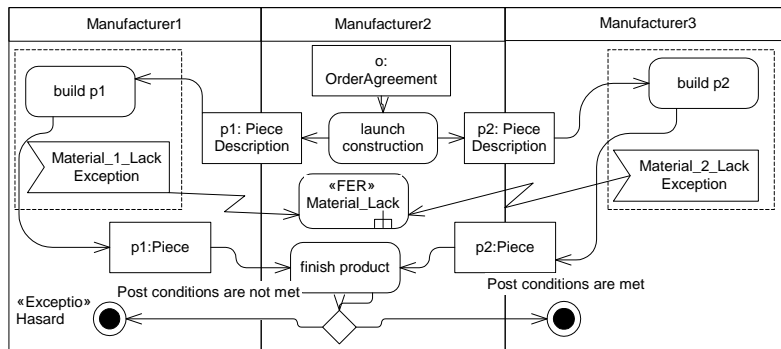


Figure 3: Production sub-process

confirmed, Mercata will deliver the product to the customer in case of availability or will ask for production. In case of production call, figure 3 shows how the manufacturers are going to work in order to satisfy Mercata request. Manufacturer2 receives the order and establishes who has to do what. Manufacturer1 and Manufacturer3 have to produce a part of the final product. In case of lack of material exception for both parts, the two exceptions are handled by consulting the exception tree, depicted in figure 5. Exceptions are represented trough a class diagram. Among exceptions a hierarchical relationship is underlined. Following this relationship it can be established which exception has to be handled in case of concurrency. Class diagrams may be mapped into XMLSchema and passed as parameters to processes. A general Material Lack exception has then to

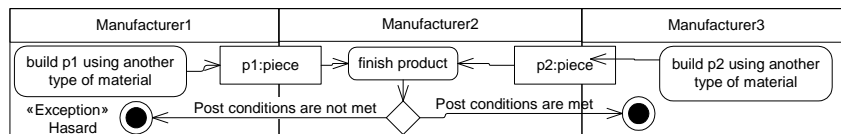


Figure 4: FER-Lack of Material.

be handled and a CallBehaviorAction stereotyped FER (Forward Exception Handling) is called. In Figure 4 the same roles of Production process cooperate in order to face the exception. If at the end of FER conditions are met, a consistent state will be reached.

This UML 2.0 extension seeks to be a first answer to the lack of ACID properties specification already mentioned in [SD05]. Its purpose is to provide basic building blocks that may be used in order to model advanced transactional protocols.



Figure 5: Class Diagram describing Exception Tree.

4 XPDL: a standard workflow process definition language

The Workflow Management Coalition (WfMC) [WfM05] was founded in August 1993 as an international non-profit organization. The goal of the WfMC is to promote and develop the use of workflow through the establishment of standards to define workflow terminology, to satisfy interoperability and connectivity between workflow products. One of the main activities since 1993 has been the development of standards for these interfaces. The WfMC’s reference model identifies five interfaces. Interface 1 is the link between the so-called “ProcessDefinition Tools“ and the “Enactment Service“. The Process Definition Tools are used to design workflows while the Enactment Service can execute workflows. The primary goal of Interface 1 is the import and export of process definitions. To support the interchange of workflow process definitions, XPDL has been proposed. XPDL [XPD02] uses an XML-based syntax, specified by an XML schema. The main elements of the language are: *Package*, *Application*, *WorkflowProcess*, *Activity*, *Transition*, *Participant*, *DataField*, and *DataType*. *Package* element is the container holding the other elements. *Application* is used to specify the applications/tools invoked by the workflow processes defined in a package. *WorkflowProcess* is used to define workflow processes or parts of workflow processes and it also consists of an oriented graph composed of nodes and edges. Nodes are activities while edges are transitions.

Activity is the basic building block of a workflow process definition. There are three types of activities: *Route*, *Implementation*, and *BlockActivity*. Activities of type *Route* are dummy activities just used for routing purposes. Activities of type *BlockActivity* are used to execute sets of smaller activities. Element *ActivitySet* refers to a self contained set of activities and transitions. A *BlockActivity* executes such an *ActivitySet*. Activities of type *Implementation* are steps in the process which are implemented by manual procedures (No implementation), implemented by one of more applications (called *Tool*), or implemented by another workflow process (*Subflow*). Activities may have *restrictions* on the incoming and outgoing transitions. A restriction on the incoming transitions is called “join“, while a restriction on the outgoing transitions is called “split“. Restrictions may have a type specification: AND or XOR. If an activity node has a join restriction type AND, then all incoming edges need to be present in order to start the activity. If an activity node has a join restriction type XOR, then the activity node is initiated when at least one of the incoming edges is taken.

Participant is used to specify the participants in the workflow, i.e., the entities that can execute work. There are 6 types of participant: *ResourceSet*, *Resource*, *Role*, *OrganizationalUnit*, *Human*, and *System*.

Elements of type *DataField* and *DataType* are used to specify workflow relevant data.

Data is used to make decisions or to refer to data outside of the workflow, and is passed between activities and subflows.

The XPDL 2.0 version has just been released [XPD05]. Relevant changes have been introduced in the new version. For example the possibility to represent messages exchanged among participants directly using *MessageFlow* entity instead of an indirect representation. Moreover the concept of transaction is finally introduced that means that an activity which is specified as transactional may terminate in three ways: normal, compensation or hazard. This is a step further towards the introduction of transaction and exception handling concepts and encourages our work in the same direction. In XPDL 2.0, in fact, a sub-process activity (implemented by Subflow) can be set as being a Transaction, which will have a special behavior that is controlled through a transaction protocol (such as Business Transaction Protocol or Microsoft Transaction Protocol). In BTP protocol, for example, participants may use recorded (before or after) images, or compensation operations to provide the roll-forward, roll-back capacity which enables their subordination to the overall outcome of an atomic business transaction. It is possible that one of the participants may end up with a problem that causes a Cancel or a Hazard. In this case, the flow will then move to the appropriate Intermediate Event, even though it had apparently finished successfully. Since our research began before the final release of XPDL 2.0 and transaction model is still an open issue (see 7.6.13 in [XPD05]), we focused on XPDL 1.0.

5 From UML 2.0 AD to XPDL

A transformation is the generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language [MG05]. In order to provide transformation rules between UML 2.0 AD and XPDL a deep knowledge on their expressiveness seems to be necessary. Their expressiveness may be evaluated by using the workflow patterns proposed in [vdAtHKB00, WvdAD⁺04, vdA03, vdA04]. In those papers a more intuitive notion of expressiveness is considered. The modelling effort is the criterion in order to evaluate the expressiveness: a great effort in modeling implies a less powerful expressiveness. We use some of the workflow patterns that result to be relevant in our project context in order to cover the basic elements presented in section 2. In cases in which there is no direct support for the workflow pattern and a workaround solution can not be sketched easily, the language under discussion results to be not powerful enough in terms of easy-modelling. In XPDL specification we can read: “using the basic transition entity plus dummy activities, routing structures of arbitrary complexity can be specified.” In order to supply to the lack of direct support to some important patterns, this suggestion has to be followed. We present in the following, in a declarative way, the translation rules. Some of these rules are already in use in the context of Efficient project (see [EBD⁺03] and [Esh03] for more details). In this work, however, a deep study about the possibility to nest processes has been done. Moreover a first step towards the inclusion of exception handling mechanism and transaction concept has been done.


AD Construct	XPDL Construct
Swimlane. Each role is modeled through a swimlane. 	a default-value inside the Data Fields in XPDL or further investigation about the use of Participant type ROLE + participant element specification inside the activity. <pre></DataType><InitialValue>Swimlane </InitialValue></DataField></pre>

Table 1: Role representation and translation.

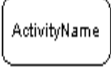
AD Construct	XPDL Construct
B2B Action node. 	Activity with No Implementation. <pre><Activity Id="ActivityID" Name="ActivityName"><Implementation> <No/></Implementation>...</Activity></pre>

Table 2: Simple task representation and translation.

Using the toolset, called animator, provided by the Efficient project, users model processes through activity diagrams and data through class diagrams. There are some constraints that have to be followed. A project has obviously to be defined and inside it a package for each activity diagram has also to be defined. The activity diagram is translated into an XPDL *WorkflowProcess* element in [GGM05] a complete example is presented, in particular the translation of the highest level activity diagram of Mercata case-study. The package in which an AD is defined has to be translated into the XPDL package. The business domain described through a class diagram could be part of the XPDL repository. In fact in the repository it is possible to introduce all the relevant data that the workflow processes may need.

5.1 Roles

Roles may represent human beings, a piece of software, a machine or something else. A role is the entity that is responsible of the execution of a task. Playing the different roles involved in the process, designers may, exploiting Efficient toolset, validate the process model. In table 1 role concept translation is shown. In particular we see that in UML 2.0 AD roles are represented through swimlanes while in XPDL we propose to use a default-value inside Data Fields.

5.2 Simple and complex activities

Simple activities represent executable activity nodes that constitute fundamental units of work in both XPDL and UML 2.0. In the Efficient project context, simple activities have no implementation and they serve only to show the documents processing, done manually in most cases. In Table 2 the translation rule is represented.


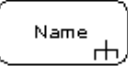
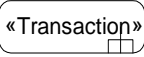
AD Construct	XPDL Construct
CallOperationAction 	Activity implementation type Tool. <pre><Activity Id="ID"Name="Operation" <Implementation> <Tool Id="IDTool" Type="APPLICATION"> <ActualParameters>...</ActualParameters> </Tool></Implementation>...</Activity></pre>
Activity node (CallBehaviourAction). 	Activity implementation type SubFlow (synch). <pre><Activity Id="ID" Name="Name"> <Implementation> <SubFlow Execution="SYNCHR" Id="RelatedWPID">...</SubFlow> </Implementation></Activity></pre>
Activity node (CallBehaviourAction) stereotyped <i>transaction</i> . 	Activity implementation type SubFlow (synch). The description element could be used in order to indicate the stereotype nature. <pre><Description>Transaction</Description></pre> In XPDL 2 the isTransaction element could be used.

Table 3: Complex tasks representation and translation.

Complex activities, represented in XPDL by Tool activities, have also operations associated. In this work we provide a UML representation for them because designers could also be interested in representing Web Services and Web Service invocation. We propose to represent Tool Activities with UML 2.0 CallOperationAction model element. Composed activities are very important because they allow designers to encapsulate specific functionalities together. These activities may accept/return parameters. Moreover by splitting a process into sub-processes we increase the possibility to reuse some of them somewhere. When a subprocess represent a real transaction concept, we propose to explicit it at design time by adding a stereotype *Transaction* in UML and exploiting the description element in XPDL. Workflow engines at the moment are not aware about transactions concepts, however, since XPDL 2.0 will rapidly replace the previous specification, the transaction concept will be supported and our work will find a concrete application. Table 3 summarizes how complex tasks may be represented and translated.

5.3 Pseudo state nodes (or Control nodes) into Route activities

Initial and final pseudo nodes could also be omitted in XPDL since they result deducible. In fact an initial node has usually no ingoing arrows, as well as a final node has no outgoing arrows. But, since in some cases, in which, for example, we are modelling loops, we could have ingoing arrows on the entry point of the initial node, we prefer translating it

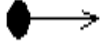
AD Construct	XPDL Construct
-initial. 	Route Activity without transition restrictions or nothing (when optimization applicable). <pre><Activity Id="initialID" Name="initial"><Route/>...</pre>

Table 4: Initial and final nodes representation and translation.


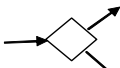
AD Construct	XPDL Construct
- fork. 	Route Activity with transition restriction AND split. <pre><Activity Id="ID" Name="fork"><Route/> ...<TransitionRestrictions> <TransitionRestriction> <Split Type="AND"/></TransitionRestriction> </TransitionRestrictions></Activity></pre>
- decision. 	Route Activity with transition restriction XOR split. WfMC defines a decision as an OR split. <pre><TransitionRestriction> <Split Type="XOR"/></TransitionRestriction></pre>

Table 5: Fork and decision representation and translation.

into XPDL by using a route activity. Since final node is dealt with similarly, the reader is referred to [GGM05].

Remaining control nodes, in some cases, may also be removed, allowing an optimization in the XPDL specification generation. A *fork* may be removed in all the cases in which it is not preceded by a decision node. The removal is possible because its semantics may be synthesized in the antecedent node. Obviously this synthesis has sense only when the semantics of the two nodes do not come into conflict. Dually a *decision* may be removed in all cases in which its antecedent is not a fork node. In cases in which no optimization rule in order to remove fork/decision nodes is applicable, we propose to represent and translate them as illustrated in Table 5. Concerning *join* and *merge* nodes, the reader is referred to [GGM05].

Deferred choice differs from the “normal“ choice, in that the choice is not made explicitly (based on existing data) but several alternatives are offered to the environment, and the choice between them is delayed until an external signal is received. Using the WFMS terminology, this means that the alternative activities are placed in the worklist, but as soon as one of them starts its execution, the others are withdrawn. This pattern is called implicit XOR-split. In UML 2.0 AD in order to represent this pattern we use the same solution proposed in [WvdAD⁺04, Whi04].

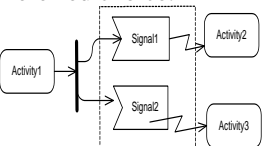
AD Construct	XPDL Construct
Deferred choice. 	not directly supported'a workaround solution has to be used. The solution already in use inside the project context has been kept.

Table 6: Deferred choice representation and translation.

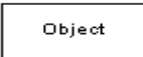
AD Construct	XPDL Construct
Object node. The details of the object and the relationships among all the relevant objects inside the process are presented into a class diagram. 	Data Field (represented by an XMLSchema)+formal and actual parameter definition in case an object constitutes input/output parameter of a sub-flow or of an application. The actual parameter must be the identifier of the corresponding workflow relevant data. <pre> <DataField Id="o_schema" IsArray="FALSE" Name="object"> <DataType> </pre>

Table 7: Documents/Objects representation and translation.

5.4 Documents and Objects

Documents and objects constitute the principal resource that has to be processed. This resource appears as input to activities, it is transformed and reappears as output. Workflow processes orchestrate these documents processing in the best way. In our proposition UML 2.0 Object nodes are translated into XPDL data fields, as Table 7 shows, which structure may be detailed in a class diagram and translated into an XML Schema and finally referenced into the final XPDL generation.

5.5 Transitions or Arcs

Activity diagrams and also the corresponding XPDL Workflow processes are oriented graphs made of nodes and arcs. After having described the nodes representation, we now show in table 8 how to represent transitions.

5.6 Exception Representation and Handling

Exceptions represent undesirable events that happen in an unpredictable instant. In order to face these events, at design time, we should think about a possible behavior to call in those circumstances. Designing exceptional behavior beside the normal one is one way in order to improve reliability. In the Efficient project context, the goal is that of efficiently capturing business transactions. By efficiently capturing, the quality is really important. Reliability is a non functional requirement that belongs to QoS requirements.

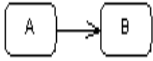
AD Construct	XPDL Construct
Edge. 	Transition. <pre><Transition From="A-ID" Id=transID" To="B-ID">... </Transition></pre>

Table 8: Transition representation and translation.

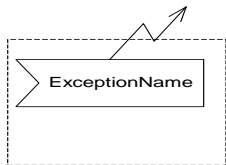
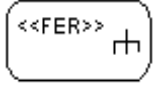
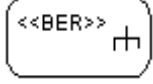

AD Construct	XPDL Construct
Accept Event Action Object Node with an outgoing Interrupting Edge inside an Interruptible region. 	Deadline specification inside each activity belonging to a specific context. <pre><Deadline Execution="ASYNCHR"> <DeadlineCondition>date </DeadlineCondition> <ExceptionName>ExName </ExceptionName></Deadline></pre> Transition condition of type Exception <pre><Transition From="Ax" Id="Trans" Name="XFailure" To="Ay"> <Condition Type="EXCEPTION"> </Condition>...</Transition></pre>
FER (call behaviour action) 	Sub-Process Definition (synchronous). The description element could be used in order to indicate the stereotype nature. <pre><Description>FER</Description></pre> In XPDL 2.0, however, an <i>Intermediate Event Activity</i> of type <i>Target</i> could be used.
BER (call behaviour action) 	Sub-Process Definition (synchronous). The description element could be used in order to indicate the stereotype nature. <pre><Description>BER</Description></pre> In XPDL 2.0, however, an <i>Intermediate Event Activity</i> of type <i>TriggerCancel</i> could be used.
Exceptional Outcome <pre><<Exception>></pre> 	Route Activity with description element Exception. <pre><Description>Exception</Description></pre> In XPDL 2.0, <i>End Event Activity</i> are available in order to underline the different possible outcomes of a transaction.

Table 9: Exceptions and exceptions handling representation.

As Table 9 summarizes, we propose to use an *InterruptibleRegion* in UML 2.0 around the part of the process that may be subject to exceptions. In XPDL, this concept may be represented by declaring a deadline related to each activity that could be subject to exceptions. Following an *InterruptibleEdge* (a transition condition of type *exception* in XPDL) we land to the exception handling mechanism that is represented by a stereotyped *CallBehaviorAction* called *FER*, in case in which we are able to bring the process to a consistent state different from the initial one, or *BER* when we only are able to roll back the situation. To describe an exceptional termination of a process we propose to stereotype the final node with *Exception*. The process will terminate with an exception in all cases in which the postconditions are not verified. This proposition has been integrated in a research project, called CORRECT [CGGP05], funded by the Luxembourg Ministry of Higher Education and Research under the project number MEN/IST/04/04. In CORRECT the transaction protocol is an advanced one, particularly adapt for facing exceptions in distributed and concurrent complex transactions.

In this work, since the protocol can not be known, because of the recent introduction of real transaction concepts inside workflow definition languages, we only propose a first step towards the inclusion of exceptions description. In XPDL 2.0 not only transaction concept will be supported but also the *Event* concept. In particular Intermediate and End Event Activities seem to be useful in order to catch exceptions and in order to express the different possible outcomes of a transaction. These improvements of expressiveness inside the language make us to keep on thinking that we are moving in the right direction by working on them.

6 Conclusion and future work

In this work we have presented a way to describe nested business processes and also exception and exception handling mechanisms in UML 2.0 activity diagrams and the corresponding transformation rules in order to generate the XPDL specification from them. In fact in order to automatically generate an XPDL specification from activity diagrams, currently, we have improved the plugin developed in the project context. This plugin has been implemented inside the commercial tool MagicDraw [NM05] and is part of the toolset Animator. Since however, at the time of writing, the stable release of this tool does not support UML 2.0 metamodel. We have used UML 1.5 metamodel (Subactivities instead of CallBehaviorAction, for example) and we have generated automatically the specification of nested processes without covering the exception handling part because of the lack of tools. Exploiting MagicDraw open API, the algorithm has been implemented in the Java language.

In the future, we intend to update to the tenth MagicDraw version in order to exploit the UML 2.0 metamodel support and to be able to complete implement the proposed transformation rules. We also intend to take into deep consideration XPDL 2.0 and in particular the support for ACID properties and exception handling. Using IsATransaction specification inside an Activity definition, a subflow, for example, can be set as being a Transaction, which will have a special behavior that is controlled through a transaction protocol.

Moreover while the translation of Backward and Forward Error Recovery into XPDL 1.0 necessitates an extension of the available workflow engines; in XPDL 2.0, these concepts are part of the language and all workflow engine compatible with the new specification will understand them. Moreover the formalisation research work started in [GM05] will be kept on by formalizing nested processes and exception handling mechanism. Our goal in fact is that of verifying not only structural properties of workflow models described in activity diagrams but also non functional ones in order to ensure step by step more trusted systems.

References

- [CGGP05] A. Capozucca, B. Gallina, N. Guelfi, and P. Pelliccione. Modeling Exception Handling: a UML2.0 Platform Independent Profile for CAA. In *Proceedings of ECOOP 2005 Workshop on Exception Handling in Object Oriented Systems, Glasgow (Scotland), Department of Computer Science, LIRMM; Montpellier-II University*, http://se2c.uni.lu/tiki/se2c-bib_download.php?id=1928, 2005.
- [DG01] E. Dubois and N. Guelfi. Demande de Contribution Financière pour la Réalisation d'un Projet de Recherche: EFFICIENT, Fonds National de la Recherche, Luxembourg-Kirchberg, Luxembourg., 2001.
- [EBD⁺03] R. Eshuis, P. Brimont, E. Dubois, B. Grégoire, and S. Ramel. EFFICIENT: a Tool Set for Supporting the Modelling and Validation of ebXML. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 359–362, 2003.
- [Esh03] R. Eshuis. Activity Diagrams as XPDL Specification, Draft version May 16,2003.
- [GCR04] N. Guelfi, G. Le Cousin, and B. Ries. Engineering of Dependable Complex Business Processes Using UML and Coordinated Atomic Actions. In Springer, editor, *International Workshop on Modeling Inter-Organizational Systems (MIOS'04), Larnaca, Cyprus*, pages 468–482, 2004.
- [GGM05] B. Gallina, N. Guelfi, and A. Mammar. Structuring Business Nested Processes Using UML 2.0 Activity Diagrams and Translating into XPDL, Technical Report TR-SE2C-05-07 University of Luxembourg 2005.
- [GM05] N. Guelfi and A. Mammar. A Formal Framework to Generate an XPDL Specification from a UML Activity Diagram, Technical Report available on <http://se2c.uni.lu/users/AM> University of Luxembourg 2005.
- [JMN03] P. Jiang, Q. Mair, and J. Newman. Using uml to design distributed collaborative workflows: from uml to xpdl. In *12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Linz, Austria, IEEE Press*, pages 71–76, 2003.
- [MG05] T. Mens and P. Van Gorp. A Taxonomy of Model Transformation (s). In *International Workshop on Graph and Model Transformation (GraMoT) Tallinn, Estonia September 28,* 2005.
- [NM05] Inc No Magic. MagicDraw 9.0 Version, Commercial tool <http://www.magicdraw.com/>, 2005.

- [Obj03] Object Management Group (OMG). Unified Modeling Language (UML): Superstructure version 2.0, final adopted specification (02/08/2003). <http://www.omg.org/cgi-bin/doc?ptc/2003-08-02>, 2003.
- [RtHEvdA04] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns., QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.
- [SD05] B. A. Schmit and S. Dustdar. Model-driven development of web service transactions. In M. Nüttgens and J. Mendling, editors, *XML4BPM 2005, Proceedings of the 2nd GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 11th GI Conference BTW 2005, Karlsruhe Germany, March 2005*, pages 39–54, <http://wi.wu-wien.ac.at/~mendling/XML4BPM2005/xml4bpm-2005-proceedings-schmit.pdf>, March 2005.
- [Tud05] Tudor Research center. Animator. <http://efficient.citi.tudor.lu/releases/>, 2005.
- [vdA03] W.M.P. van der Aalst. Patterns and XPD L: A Critical Evaluation of the XML Process Definition Language., QUT Technical report, FIT-TR-2003-06, Queensland University of Technology, Brisbane, 2003.
- [vdA04] W.M.P. van der Aalst. Business process management demystified: A tutorial on models, systems and standards for workflow management. In Springer, editor, *In J. Desel, W. Reisig, and G. Rozenberg, editors, Lectures on Concurrency and Petri Nets, volume 3098 of Lecture Notes in Computer Science, Berlin*, pages 1–65, 2004.
- [vdAtHKB00] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced Workflow Patterns. In Springer, editor, *In O. Etzion and P. Scheuermann, editors, 7th International Conference on Cooperative Information Systems (CoopIS 2000), volume 1901 of Lecture Notes in Computer Science, Berlin*, 2000.
- [WfM05] WfMC. <http://www.wfmc.org/>, 2005.
- [Whi04] IBM Corp. United States S. A. White. Process Modeling Notations and Workflow Patterns. In *The Workflow Handbook 2004. WfMC, Future Strategies Inc., Lighthouse Point, FL, USA*, 2004.
- [WvdAD⁺04] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. Pattern-based Analysis of UML Activity Diagrams, BETA Working Paper Series, WP 129, Eindhoven University of Technology, Eindhoven, 2004.
- [XPD02] XPD L1.0. http://www.wfmc.org/standards/docs/tc-1025_10_xpdl_102502.pdf, 2002.
- [XPD05] XPD L2.0. http://www.wfmc.org/standards/docs/tc-1025_xpdl_2_2005-10-03.pdf, 2005.