# XPDL Enabled Cross-Product Exception Handling for WfMSs

**Carlo Combi, University of Verona, Italy,**
**Florian Daniel, University of Trento, Italy, and**
**Giuseppe Pozzi, Politecnico di Milano, Italy**

## Abstract

The effort invested by the Workflow Management Coalition in the interchangeability of process definitions has led to the definition of the XPDL language, a commonly acknowledged XML format for process definition. While XPDL effectively enables the cross-product portability of process definitions, the language has not been designed to also capture undesired behaviors that may arise during process execution, i.e. exceptions. Nonetheless, exceptions—especially those that are predictable at process definition time—do have semantics that are not negligible.

Our investigation of exception handling mechanisms in workflow management products has shown that a commonly accepted approach does not exist, and that, hence, a proposal for an exception-specific XPDL extension would probably not succeed. In this chapter, we describe our resulting idea of leveraging the products' very own extension mechanisms to enable cross-product exception definitions. The proposed approach operates at the conceptual level, formalizes exceptions in a fully XPDL-compliant fashion, and abstracts from product-specific details until process execution. Along with the description of our approach, we also discuss our experience with its effective portability across a few XPDL-compliant commercial and open-source workflow management products.

## 1 Introduction

Workflow management systems (WfMSs) enable the automated management of work that is typically executed by multiple roles (comprising both human actors and software applications) and whose flow is modeled according to one of the explicit process definition formalisms that have been developed over the last years. "Explicit" means that a process model expresses what is allowed or admissible during the execution of the actual work; other activities can simply not be enacted by the involved actors. All the semantics of the workflow are captured and suitably expressed in the process definition by means of basic modeling constructs such as tasks, roles, splits, joins, conditions, and so on.

Unfortunately, in general it is not easy—if not impossible—to explicitly capture all possible situations that may happen during the execution of a given process. Specifically, Eder and Liebhart [EL 1995] distinguish between exceptions and failures, which they further specialize into: *expected exceptions* (e.g., the impossibility to complete a payment activity), *unexpected exceptions* (e.g., the need to change the process definition during runtime), *basic failures* (e.g., a system crash), and *application failures* (e.g., a null pointer error). In addition to the model of the expected behavior of a WfMS, it is thus also necessary to specify how the system should react to the previous problems.

As failures are however out of the control of the WfMS, and unexpected exceptions cannot be predicted at process definition time, in this chapter we shall focus on expected exceptions[1], which can be considered part of the semantics of the overall process definition. Expected exceptions indeed deal with events that depend on the actual process in execution such as constraint violations over data managed by the process, or the start/end of a task or of a case, or a temporal deadline (e.g., for the completion of a task within a given timestamp), or an external event (e.g., a phone call from a customer to cancel the car reservation in a car rental company).

In order to successfully capture, manage, and execute exceptions, several efforts have been performed. We mention here WIDE [CCPP 1999] and other exception handling units [HA 2000], which are however tightly bundled inside the WfMSs and the process definition formalisms they have been developed for. As a consequence, there is no portability of exception specifications among different workflow management (WfM) products. The only way to obtain a portable and cross-product implementation of exception handling features is to map them onto the activity graphs.

In this chapter, we describe our experience with mapping exception handling features onto process or activity graphs. As our goal is to provide a portable definition of such exception handling features, we have opted for XPDL as process definition formalism [MNN 2005], also to assess its real support by current WfM products. The enriched process models we obtain from this mapping process still comply with the recommendations of XPDL and should successfully execute in XPDL-compliant WfMSs, provided that they enable a suitable customization of the system—a feature that is supported at different levels by all of today's WfMSs. We checked the portability of a normal process (one without embedded exception handling constructs) as well as that of the respective enriched process on some WfMSs declared to be XPDL-compliant. So far, we have considered Enhydra Shark, Bonita, Ascentn Agile Point, OBE, and WfMOpen, and results are interesting.

## 2 A REFERENCE EXAMPLE

Throughout this chapter, we shall make use of a reference example to better explain our approach. Consider Figure 1, which describes the *OrderManagement* process of a company trading in books called *RareBook* with very limited editions and accepting orders by its customers via telephone; payments can be done by credit card only.

After receiving an order, the *Sales Office* immediately checks the credit status of the customer's credit card. In case of overdraft, the order is declined. Otherwise, the *Sales Office* proceeds and checks the stock for the requested books. If all the products are available, the customer is notified of the immediate shipment, and the *Production* department provides for shipment. If, instead, not all the requested products are available, the *Sales Office* informs the user and asks for the approval of a delayed shipment. To accelerate the overall process, the *Production* department plans the production of the missing items in parallel, regardless of the user's decision, produces them and, if no cancellation is received, ships the complete order.[2]

[1] For simplicity, in the following we shall use the term *exception* to refer to the specific case of *expected exceptions*.

[2] Due to space restrictions, in this chapter we cannot exemplify the whole exception mapping procedure. For a more detailed discussion, the interested reader is referred to [CDP 2006].

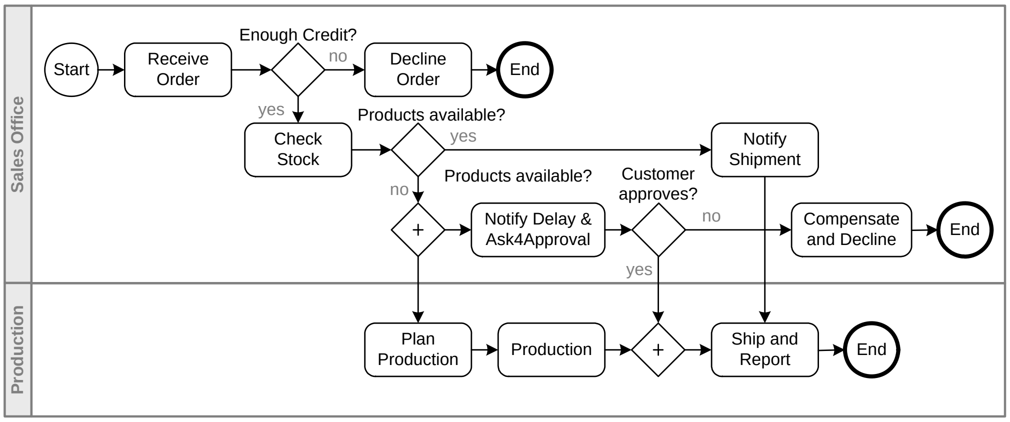

***Figure 1 The OrderManagement process of the RareBook online book shop [CDP 2006].***

For a better readability, in this chapter we shall use BPMN modeling constructs to express XPDL processes and activities, instead of providing unexpressive snippets of XPDL code. *Tasks* are represented as boxes, whose names describe the performed activity. *Transitions* are represented as directed arcs to connect process nodes, which, for routing purposes, may have associated a *condition* over workflow-relevant, application, or system data. Each process definition has one *start* and one *end* task, depicted by thin or thick circles, respectively. *Gateways* are represented as diamonds; condition gateways also have an associated textual condition, while *AND* gateways are diamonds that contain a + symbol.

## 3 DEFINING EXCEPTIONS

Expected exceptions in WfMSs deal with events that may occur during the enactment of a case. In order to successfully complete the execution of a case that has been affected by an exception, optional conditions may be tested and, possibly, some actions can be performed. Typical performed actions include the activation of a specific task or of a new case, the roll-back or compensation of the tasks already completed for that case, the re-assignment of a task to a new actor, etc.

Expected exceptions are generally classified according to their triggering event, i.e. to the event that raises the exception. As in [CCPP 1999], triggering events are classified as:

- *Workflow events*: events such as the start/end of a task/case may trigger an exception;
- *Data events*: the insertion, deletion, or update of a value of a workflow variable may trigger an exception, e.g., to monitor whether constraints defined over that workflow variable are violated or not;
- *Temporal events*: the occurrence of a specified timestamp, as well as the expiration of a deadline (e.g., for the completion of a task/case) or the periodic occurrence of timestamps (e.g., to perform a complete backup of the archives every Monday morning at 3:30 a.m.) may trigger an exception [CP 2004];
- *External events*: the occurrence of events raised by external applications (e.g., if the water level inside a basin raises above a specified maximum level) may trigger an exception which requires an intervention by the WfMS.

Properly dealing with such kinds of exceptions typically requires that the process execution continues with a deviation from the normal flow of execution, allowing the WfMS to respond to the exception. The semantics of expected exceptions are not negligible when specifying a new process definition.

While very few WfMSs come with a proper exception management unit, several efforts have been performed in order to define tools and languages for exception handling. As a reference, in our approach we consider the Chimera-Exception language [CCPP 1999], which, although quite complex, enables one to define exception handling triggers that are able to manage several types of events and of actions. The language reflects the above classification of events and provides an adequate set of process-specific actions to deal with typical exceptions in WfMSs. Also, the language is characterized by an adequate level of abstraction, i.e., it is completely product-independent and remains at a conceptual level, and is thus particularly suited as basis for our cross-product exception handling approach.

Referring to the example of the *RareBook* agency of Section 2, we consider here the following Chimera-Exception trigger (*CustomerCancel*), which is activated whenever a customer calls in to cancel his/her book order, causing the abnormal interruption of a respective case of the process *OrderManagement*:

```
define trigger CustomerCancel for OrderManagement
  events    modify(Cancel)
  condition OrderManagement(O),
            occurred(modify(Cancel),O),
            O.Cancel="Yes"
  actions   startTask("CompensateAndDecline")
end
```

The meaning of the trigger *CustomerCancel* is as follows. The trigger is defined for the *OrderManagement* process only, which means that the trigger is a so-called process-specific trigger whose validity is confined to that specific process. The trigger reacts to a data event over the workflow-relevant data field *Cancel*, which is a process variable that is set to “No” as long as no cancellation has occurred; we assume that in the moment a users calls in to cancel an order, *Cancel* gets set to “Yes”. The condition first defines an object *O* of type *OrderManagement*, which is used to identify the specific instance of *OrderManagement* in which the event has occurred. Then, the condition checks whether the new value of the *Cancel* variable equals “Yes”, in which case the trigger enacts its action; otherwise, no action is performed. The action consists in the enactment of the task *CompensateAndDecline*, which can be seen in Figure 1 and causes the cancellation of the running case.[3] In this example, a task already existing in the normal process specification is enacted; it could be the case that an ad-hoc task has to be enacted, to be performed either by the exception handler or by another agent.

## 4 MAPPING EXCEPTIONS TO XPDL

The mapping of triggers like the previous *CustomerCancel* trigger from Chimera-Exception to XPDL occurs along two orthogonal dimensions, one that is *process-independent* and one that is *process-dependent*:

- The support for the basic expression evaluation and expression composition functions, as well as the support for the enactment of Chimera-

[3] The full syntax of the Chimera-Exception language is out of the scope of this chapter, but the interested reader is invited to read [CCPP 1999].

Exception's set of workflow actions, requires the provisioning of a set of basic sub-processes that can be reused across multiple process definitions. These sub-processes form a library of basic exception handling features, whose internal implementation typically varies from product to product. Sub-processes are process-independent.

- The actual mapping of the exception handling logic, instead, is process-dependent, as it requires intimate knowledge of the process structure in order to be able to correctly expand the original process graph with exception handling constructs according to the nature of the trigger to be mapped. The expansion of the process graph uses and combines basic sub-processes into macros and patterns in order to achieve an expressive power that is equivalent to the one of Chimera-Exception.

Due to the very tight interactions between the basic exception management constructs (the sub-processes) and the workflow engine of the host WfMS in order to suitably capture events and to enact actions, the development of the library of basic sub-processes typically requires intimate knowledge of the host WfMS. The knowledge of a process designer who is skilled in using the host WfMS however largely suffices to perform this task.

The mapping of the exception handling constructs onto the actual process graph enables one to define and manage expected exceptions even in WfMSs which do not feature an own exception management unit. According to the characteristics of the considered exception (i.e., its triggering event, its synchronicity or asynchronicity with respect to the normal flow of work, the number of process models affected by the exception—to mention few of them), several mapping techniques are required [CDP 2006].

In general, mapping events and actions into the process definition leads to less efficient and sometimes hardly readable schemata. To alleviate process designers from the inherent complexity of enriching existing process definitions with exception handling constructs by hand, we have developed a suitable compiler, which enables the designer to model the *plain* process as usual, to specify the exceptions to be handled in form of Chimera-Exception *triggers*, and to automatically compile the trigger definition into the *enriched* process definition. Process designers, thus, do not have to deal with the complexity of the enriched process graph. The enriched process definition produced by the compiler is still runnable by the host WfMS, provided that the necessary sub-processes have been customized.

Figure 2 depicts the logic of how to obtain the enriched process definition. The compiler requires in input the plain process definition in XPDL and a text file containing the Chimera-Exception rules to be compiled. Starting from the developed XPDL macros, patterns, and sub-processes, the compiler then enriches the plain process definition with exception handling constructs by adding a proper exception handling swim lane into the process definition. After the compilation, an optimizer prunes auxiliary nodes from the enriched process, which are required during the compilation process. Finally, the whole mapping process produces in output the enriched XPDL process definition and a report of the compilation/optimization process.

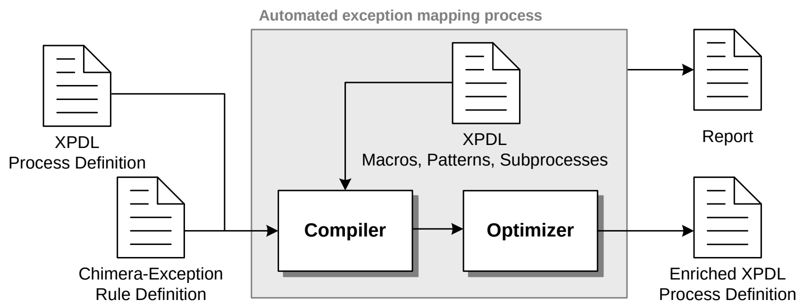

***Figure 2 Compiling Chimera-Exception rules into enriched XPDL process definitions.***

Let us now consider the example exception defined in Section 3. The exception is asynchronous with respect to the actual process flow since its time of occurrence cannot be known a priori: in fact, there is no relationship between the state of the running case and the timestamp at which the customer calls in to cancel the order. The exception is also process-specific, meaning that it relates to all the instances of the schema *OrderManagement*, only.[4] The only way of enriching a process model, enabling it to manage asynchronous exceptions, is that of starting immediately at case start time an exception handler that periodically checks for the raising of the exception. If the exception occurs, it can be captured at periodical time instants, which enables the WfMS to properly react. The mapping of such an asynchronous exception is performed by means of an additional swim lane, i.e., *Exc. Handler*, which contains the necessary exception handling logic. The result of the compilation of the trigger from Section 3 into the process definition of Figure 1 is shown in Figure 3.

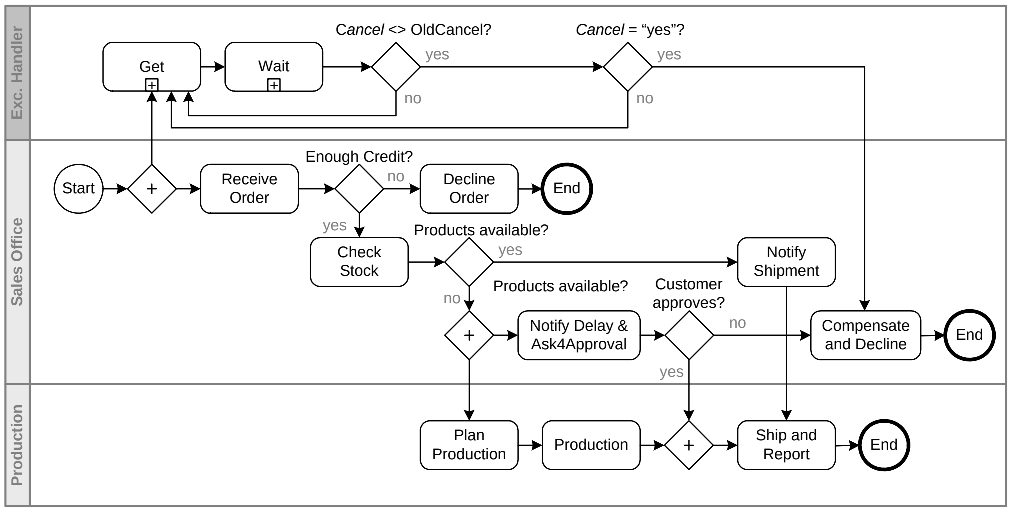

***Figure 3 The original process model (see Figure 1) enriched with exception handling constructs. The compilation process automatically added the Exc. Handler swim lane.***

[4] Other exceptions may also be cross-process, meaning that they may affect instances of different schemata (e.g., when an agent leaves, he/she may be involved in the executions of several cases from different process models, and all these cases will have no executing agent).

Let us analyze in detail the contents of the *Exc. Handler* lane. In order to check whether the data event of the exception has occurred, the *Get* sub-process reads the workflow variable named *Cancel* and stores its value in the temporary workflow variable *OldCancel*. The sub-process *Wait* then inserts a temporal delay *D*, i.e., the temporal interval that must expire between two subsequent readings of the same workflow variable to check for modifications. The actual comparison of the two values of the *Cancel* variable is performed by the *OR* split "Cancel <> OldCancel?", which compares the current value of *Cancel* with its previous value stored in *OldCancel*. The outgoing arc that goes back to the *Get* sub-process is executed if the two values are the same, meaning that no change has been detected; instead, the other outgoing arc that goes to the next *OR* split is executed if a change in the workflow variable has been detected. This concludes the event detection logic for the mapped trigger. The final *OR* split in the *Exc. Handler* lane represents the mapping of the condition part of the trigger. It checks whether the new value of *Cancel* is "Yes", in which case the *Compensate and Decline* task is executed (the action of the trigger); otherwise, the event detection logic continues polling the *Cancel* variable.

The so enriched process definition (from Figure 1 to Figure 3) is now capable of monitoring the occurrence of the data event *modify(Cancel)*, of checking the respective condition, and of enacting the *Compensate and Decline* task as required by the trigger in Section 3. It is worth noting that choosing an adequate polling interval to monitor the occurrence of data events is a critical task: in fact, too high a value of *D* introduces an average detection delay of $D/2$, which can be critical for some real-time applications; too small a value of *D*, even if reducing the detection delay, introduces an additional workload into the host WfMS. Fixing a suitable polling interval is thus a trade-off between the two considerations; a meaningful value can be configured before starting the compilation process.

## 5 EXECUTING PROCESSES ON XPDL-COMPLIANT WFMSS

In order to validate our exception mapping methodology, we tried to execute both the plain and the enriched process definitions on some XPDL-compliant WfMSs. We considered only systems that declare to be capable of directly executing XPDL process definitions or, however, of importing XPDL process definitions.[5] More precisely, the website of the Workflow Management Coalition[6] lists several WfMSs with the required characteristics. Some of these WfMSs are open source and available for free, while some others are not. Among the first ones, we chose and performed our tests on Enhydra Shark (version 2.0), WfMOpen (version 2.1), Bonita (version 3.0), and OBE - Open Business Engine (version 1.0); among the latter ones, we chose Ascentn Agile Point Server (version 4.01), which has been kindly provided to us by the vendor.

In order to perform the tests over the five different WfMSs, we needed five separate installations that could not interfere with each other. This was achieved by installing each WfMS on a separate virtual machine: we used several VMware virtual machines, one running Windows 2000 for Enhydra Shark, for OBE, and for WfMOpen, one running Windows XP for Bonita, and one running Windows 2003 Server for Ascentn Agile Point (these were  strong installation requirements by the considered WfMSs). As our goal was to test the overall approach and the portabil-

[5] In this chapter, we always refer to XPDL version 1.0. For an overview of XPDL versions, the interested reader could refer to [Shapiro 2006].

[6] http://www.wfmc.org/standards/xpdl.htm

ity of plain and enriched process definitions, we did not take into account the performance of the different systems; this also justifies the use of virtual machines to run the systems.

The *OrderManagement* process model of the *RareBook* agency described in Section 2 has been modeled with the open source workflow editor Enhydra JAWE, which allows the graphical editing of process definitions and supports XPDL as native file format. According to its developers, JAWE does not make use of any proprietary XPDL extensions and, hence, provides a fully compliant implementation of XPDL.

Very few changes (ideally no changes) were expected when porting the plain process from one XPDL-compliant WfMS to another XPDL-compliant WfMS. On the other hand, the porting of the enriched process was expected to require more efforts, also due to the additional product-specific modules (macros, sub-processes) that are required. The following considerations do not represent a thorough scientific investigation, but rather represent a straight-forward experience report that highlights the problems and difficulties that we have encountered during our first test phase.

### *5.1 Implementation of Basic Sub-processes*

One could expect that sub-processes (like *Get* and *Wait* in the considered example) are defined once for all as XPDL library. Unfortunately, as their internal execution logic typically requires a tight integration with the WfMS, i.e. they are WfMS-dependent, it is not possible to provide a universal library of exception handling sub-processes. Therefore, for each of the systems considered, we implemented a minimum set of sub-processes, in order to support a few tests and, in particular, the example described in this chapter. We were able to provide a small library of exception handling sub-processes with consistent interfaces across the libraries for each of the systems.

### *5.2 Execution Tests*

As a first proof of validity of the proposed approach, we only used Enhydra Shark. We loaded the plain process model into Enhydra Shark and successfully run it. Next we specified the exception of Section 3 in a text file, compiled it into the enriched process model of Figure 3, and successfully run it again, leveraging the Shark-specific library of sub-processes. To our satisfaction, executing both versions of the process did not require any noteworthy interventions in the respective process definitions.

The next tests dealt with porting the plain process model from Enhydra Shark to the other WfMSs. We successfully imported the process model into WfMOpen and back. When we tried to import the respective process model into OBE and Bonita, we experienced some difficulties with both the WfMSs in reading the XPDL file designed with Enhydra JAWE. Thus, we had to adjust the XPDL file and to slightly redesign the plain process in both OBE and Bonita: main changes concerned the translation between basic data types (enumerative, string etc.), date formats (month/day/year, day/month/year), roles of agents. Subsequently, we also experienced difficulties when importing the plain process into Ascentn Agile Point, and we redesigned the process for Ascentn Agile Point: these importing difficulties relate to mismatching tags in the file format. To complete the test of porting the plain process model across the chosen WfMSs, we tried to port each product-specific process model to all the other WfMSs, but again in some cases readjusting and partial redesigning were still needed.

At this point, we had four XPDL files (Enhydra Shark and WfMOpen shared the same file), describing the same process model of Figure 3 but in four different XPDL implementations. According to the mapping methodology described in Section 4 and starting from the four different definitions of the plain process model, we were however able to obtain four enriched process definitions that could be executed on the five systems. However, in this case, we had to adjust the output of the compiler according to the specific XPDL dialect, just as we had to adjust the plain process definition in the first place. Nonetheless, the actual mapping of the exception definitions from Chimera-Exception to the single XPDL dialects of the chosen products works, and the resulting enriched processes execute correctly.

Due to the described difficulties, we are now considering to provide the rule compiler with suitable drivers (i.e., language customizers) for each of the encountered XPDL dialects.

## 6 DISCUSSION AND OUTLOOK

In this chapter we described our experience with the automatic enrichment of existing process definitions with inline exception handling constructs. Exceptions are specified as triggers in Chimera-Exception, an event-condition-action language developed in the context of the WIDE project [CCPP 1999]; process definitions are formalized in XPDL. A suitable compiler enables the automated translation of exception definitions from Chimera-Exception into XPDL, and WfMSs are equipped with a common library of supporting sub-processes, which provide for the necessary execution support for enriched process definitions and enable the actual portability of the proposed approach.

The results achieved so far with this cross-product exception handling approach can be summarized as follows:

- The mapping technique is robust when moving from an XPDL process definition to its enriched version;
- It is possible to provide product-specific implementations of the basic sub-processes that are at the basis of the execution of enriched processes (at least for the five WfMSs considered in this chapter);
- Enriched processes execute correctly and still have a good performance in terms of usage of system resources and execution (till now, this is only a subjective evaluation).

As the previous section has shown, we could however not achieve the initially expected level of portability among different XPDL-compliant WfMSs. While XPDL, on the one hand, was the enabling factor that led us to conceive the described approach, on the other hand, it is also its main limiting factor. Indeed, our "naive" portability tests (just save/export and load/import tests) for both the plain and the enriched version of a process definition revealed the existence of different dialects of XPDL in the tested WfMSs, which prevent even generic process definitions from really being portable among the systems.

In order for XPDL to succeed on the market of WfMSs, it seems thus of utmost importance that vendors fully comply with the language specification proposed by the Workflow Management Coalition, and that they avoid as much as possible the use of proprietary XPDL extensions. It is not necessary that products use XPDL as native file format, but—although this might prevent the optimal use of a given system's features—we believe the provided import and export functions should concentrate on XPDL-compliant constructs, only. If a process designer wants to use XPDL, he/she will be aware of and accept the possible limitations, but he/she will also be sure that process definitions effectively are portable.

In this chapter, we informally described the difficulties we encountered in testing our exception handling approach. In our future work, instead, we shall try to better understand the nature of these difficulties so as to formalize them in a more detailed manner and to be able to propose concrete solutions. Also, we intend to systematically evaluate the execution performance of plain and enriched processes in terms of workload, robustness, and speed.

## 7 ACKNOWLEDGEMENTS

This work has been partially funded by the Department of Computer Science, University of Verona and by the Department of Electronics and Information, Politecnico di Milano.

## 8. REFERENCES

[CCPP 1999] Casati, F., Ceri, S., Paraboschi, S., & Pozzi, G. (1999). Specification and Implementation of Exceptions in Workflow Management Systems. ACM Transactions on Database Systems, 24(3), 405-451.

[CDP 2006] Combi, C., Daniel, F., & Pozzi, G. (2006). A Portable Approach to Exception Handling in Workflow Management Systems. In R. Meersman & Z. Tari (Eds.), OTM Conferences (1), LNCS 4275 (pp. 201-218). Montpellier, France: Springer Verlag.

[CP 2004] Combi, C., & Pozzi, G. (2004). Architectures for a Temporal Workflow Management System. In H. Haddad, A. Omicini, R. L. Wainwright, & L. M. Liebrock (Eds.), Proceedings of SAC'04 (pp. 659-666). New York: ACM Press.

[EL 1995] Eder, J., & Liebhart, W. (1995). The Workflow Activity Model WAMO. In Proceedings of CoopIS'95 (pp. 87-98).

[HA 2000] Hagen, C., & Alonso, G. (2000). Exception Handling in Workflow Management Systems, IEEE Transactions on Software Engineering, 26(10), 943-958.

[MNN 2005] Mendling, J., Nuemann, G., & Nuttgens, M. (2005), A Comparison of XML Interchange Formats for Business process Modelling, Workflow Handbook 2005, Edited by Layna Fischer, Future Strategies Inc., Lighthouse Point, Florida.

[Shapiro 2006] Shapiro, R. M. (2006) XPDL 2.0: Integrating Process Interchange and BPMN, Workflow Handbook 2006, Edited by Layna Fischer, Future Strategies Inc., Lighthouse Point, Florida.