A Portable Approach to Exception Handling in Workflow Management Systems

Carlo Combi[1], Florian Daniel[2], and Giuseppe Pozzi[2]

[1] Università degli Studi di Verona, Strada le Grazie 15, 37134 Verona, Italy
carlo.combi@univr.it
[2] Politecnico di Milano, P.za Leonardo da Vinci 32, 20133 Milano, Italy
{florian.daniel, giuseppe.pozzi}@polimi.it

Abstract. Although the efforts from the Workflow Management Coalition (WfMC) led to the definition of a standard process definition language (XPDL), there is still no standard for the definition of expected exceptions in workflows. Yet, the very few Workflow Management Systems (WfMS) capable of managing exceptions, provide a proprietary exception handling unit, preventing workflow exception definitions from being portable from one system to another one.
In this paper, we show how generic process definitions based on XPDL can be seamlessly enriched with standard-conform exception handling constructs, starting from a high-level event-condition-action language. We further introduce a suitable rule compiler, enabling to yield portable process and exception definitions in a fully automated way.

1 Introduction

Workflows are activities involving the coordinated execution of atomic activities (*tasks*) performed by different processing entities (*agents*). The process model (*schema*) of a workflow includes the description of its component tasks, of the flow of execution, of *routing nodes* (*connectors*) to activate parallel (*AND split*) and conditional executions (*OR split*), and to resynchronize executions (*AND join*, *OR join*). Instances of schemas are known as *cases*. Workflow management systems (WfMS) support the automatic execution of workflows [1, 2].

The schema of a workflow considers the *normal* flow of execution, where cases evolve through predefined execution paths according to the variables of every case. Occasionally, during the execution of a case, some exceptional situations may occur, possibly deviating the execution path from the set of normal execution paths. The semantics of these exceptions is not negligible and must be foreseen at workflow design time: we call these exceptions *expected exceptions* [3]. Other exceptions (e.g. power fail, network disconnection...) are not modelled at workflow design time and will no longer be considered throughout the paper.

The *Workflow Management Coalition* (WfMC [1]), whose recommendations and standards influence WfMS vendors, issues the XPDL language, which is an XML-based process definition language, aimed at obtaining portability of the process definitions among different WfMSs. While several efforts have been

done with respect to the normal process flow definition, and several WfMSs may share the same schema based on XPDL, very few WfMSs come with an expected exception handling unit and no standard has been achieved about exceptions.

In the following, we propose a technique to map expected exceptions inside the XPDL schema, and: i) provide WfMSs with a primordial exception management unit; ii) maintain the portability of the process definition among WfMSs, capable of processing XPDL descriptions. As reference language to define, capture, and manage exceptions we consider the Chimera-Exception language [4].

In this paper, Section 2 considers related work; Section 3 and Section 4 consider process and exception specification, introducing XPDL and Chimera-Exception. Section 5 shows how to model ECA rules in XPDL. Section 6 describes how to support expression evaluation in XPDL, used in Section 7 to map ECA rules onto process specifications. Section 8 describes some practical experiences gained. Finally, Section 9 draws some conclusions and anticipates future work.

2 Related Work

The literature covers several facets of exception handling in WfMS. Mourão et al. [5] define a suitable environment to orchestrate ad-hoc human interventions with a minimum impact on system integrity. Golani et al. [6, 7] consider how to increase the flexibility of a WfMS to deal with unexpected exceptions, and propose a dynamic mechanism that allows backtracking and forward stepping at the instance level. Luo et al. [8] focus on exception-handling schemes for conflict resolution in the delivery of e-business transactions, with particular attention to support conflict resolution in cross-organizational settings. Schuschel et al. [9] exploit the functionalities of triggers defined within a WfMS to automate the creation of process definitions by planning algorithms: by replanning and dynamically adapting the process, the newly defined process is capable of reacting to unexpected events. Finally, van der Aalst et al. [10] propose a mixed approach between workgroup management systems and WfMSs,where the normal evolution of the process is managed by predefined process control structures, while exceptions are manually managed by a human agent and the case handling system assists rather than guiding him/her in doing so.

Our particular attention is focused on expected exceptions as defined by Eder et al. in [3]. To this regard, very few WfMSs include a fully fledged exception handling unit to deal with expected exceptions. An interesting approach to exception handling is provided by the OPERA prototype [11], where exceptions can be triggered by data events or by notifications from external applications: tasks and control flows are used as reactions to captured exceptions. In OPERA, as soon as an exception is detected, the process is suspended and the control is transferred to the exception handler. COSA [12] comes with the notion of trigger, defined as an event-action rule capturing events or deadline expirations, and reacting by activating a task or a (sub)process instance. InConcert [13] includes event-action triggers in its workflow model: triggering events can be process state changes (e.g., a task becomes ready for execution), external (user

defined) events, or temporal events; actions include notification of messages to agents, activation of a new process, or invocation of a user-supplied procedure. Staffware [14] and HP's Chang*e*ngine allow the definition of a special kind of task called event node (or event step), which can suspend the case execution on a given path until a defined (exceptional) event occurs, and can then activate an event-handling path in the workflow.

The WIDE project has a powerful exception handling unit [4], where exceptions are specified by means of Event-Condition-Action (ECA) rules: workflow, temporal, external, and data events may trigger actions as suspending a case or a task, starting a new case, changing database instances, or notifying messages to agents. In this paper, we adopt the formal Chimera-Exception language developed within the WIDE project.

3 Business Process Modelling and XPDL

The XPDL (*XML Process Definition Language*) is an XML file format from the WfMC: it has the same expressive power as the visual process specification language Business Process Management Notation [15] (BPMN). XPDL can be used as file format of BPMN. In this paper we shall use BPMN modelling constructs for expressing 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.

An XPDL document is structured hierarchically, represents a *package* made of a set of process definitions, and includes package-wide workflow *participants*, *applications* and *relevant data fields*. *Process definitions* may contain references to separately defined *sub-flows*, making up part of the overall process definition, and inherit globally defined entities. Process definitions are articulated in *pools* and *swim lanes*. For instance, when modelling interactions in B2B scenarios, pools represent private business processes, while swim lanes represent the different participants performing each of the private processes. Process *activities* represent tasks and are associated to pools and lanes. Activities are connected by means of *transitions* (intra-pool flows) or *message flows* (inter-pool flows).

An Example As process model example, we consider (Figure 1) a company trading in books with very limited editions and accepting orders by its customers via telephone; payments are 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.

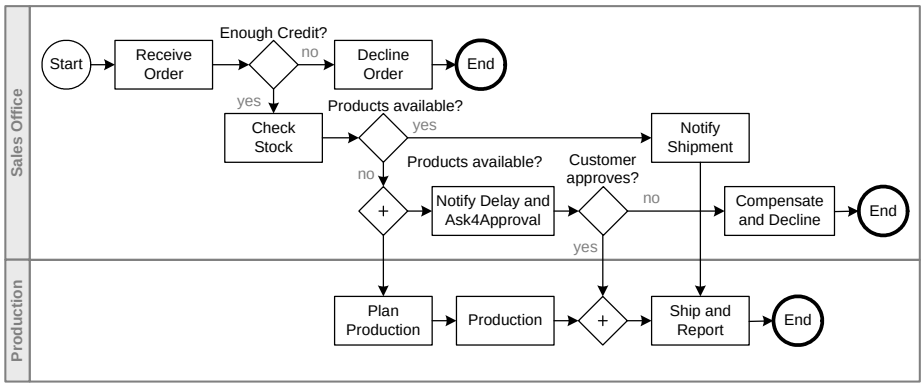


Fig. 1. Example `OrderManagement` process definition

4 Handling Workflow Exceptions

Although "anomalous" with respect to the "normal" process flow, expected exceptions are part of the semantics of the overall process and can be classified [4, 16] according to *synchronicity*, *scope*, and *triggering event*.
Synchronicity: synchronous exceptions are bound to specific workflow events (i.e., task start, task completion), while asynchronous ones may occur at any time during process execution. Synchronous exceptions may be further subdivided into *localized* exceptions, caused only by the execution of one or few tasks, and *sparse* exceptions, caused at several stages during process execution.
Scope: *process-specific* exceptions may occur during the execution of one unique process; *cross-process* exceptions may occur during the execution of more processes; *global* exceptions may occur during the execution of any process.
Triggering event: events can be *data*, *temporal*, *external* or *workflow* events.

These characteristics strongly influence the way expected exceptions can be mapped onto XPDL, and thus onto process definitions. Mapping exceptions moves from Chimera-Exception definitions, as specified in the following.

4.1 The Chimera-Exception Language

Chimera-Exception builds on an object-oriented formalism. Classes are *workflow-independent* (e.g., role, agent, task) or *workflow-dependent* (e.g., workflow-relevant data fields). Rules are specified by event-condition-action constructs.

Events. Rules can be triggered by the following events: *data events* (changes of the content of workflow-relevant data or within the underlying data source, by `create`, `modify` or `delete` statements); *temporal events*, which can be (i) *instant events* expressed through the `@`-symbol (e.g. `@timestamp ‘‘December 15th, 2005’’`), (ii) *periodic events* defined by the `during` keyword (`1/days during weeks` denotes the first day of every week [17]), (iii) *interval events* expressed as `elapsed` *duration* `since` *instant*, (e.g. `elapsed (interval 1 day) since caseStart`); *external events* (recognized by means of the `raise` primitive, which – when an external event occurs – provides the name of the triggering event and suitable parameters); or *workflow events* (start and completion of tasks and cases by the primitives `caseStart`, `caseEnd`, `taskStart`, and `taskEnd`).

Conditions. A condition consists of predicates that inspect the content of the database. The predicate `occurred` enables to refer to objects or tasks that were affected by the triggering event. The predicate `old` enables to access the database state at the time of the triggering.

Actions. The actions of Chimera-Exception focus on exception management within the workflow environment and can assign a task or a case to an agent, cancel a task or a case, start a task or a new case, suspend a task...

The trigger `ProductionNotification` valid for the schema `OrderManagement` is an example of a Chimera-Exception trigger. It monitors – as event – the completion of the task `Production`: after the completion of the task, the trigger notifies the `Sales Office` about the terminated production.

```
define trigger ProductionNotification for OrderManagement
  events    taskEnd("Production")
  condition Agent(A), A.Name = "Sales Office"
  actions   notify(A,"Production done.")
end
```

The `CustomerCancel` trigger reacts to cancellations of the whole process. The process is informed of the cancellation by changes to the workflow-relevant variable `CustomerCalledToCancel`. During case execution, should the parameter change to "Yes", the task `CompensateAndDecline` is started.

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

5 Modelling ECA Rules in XPDL

In the next sections, we show how the textual specification of high-level Chimera-Exception rules can be translated into XPDL by a mapping mechanism based on

XPDL *macroes*. Macro modules represent fragments of the process graph and provide flexible interconnections of workflow fragments and macro modules.

Macroes consider *events*, *conditions*, and *actions*. High-level macroes are made of lower-level macroes, such as connector macroes, and of a set of predefined sub-processes, which serve for evaluating expressions.

5.1 The Rationale for Macroes

A generic macro element has at least one input node, prefixed by "`In-`", and one output node prefixed by "`Out-`". The suffix specifies the kind of graph element that can be connected to the interface. Table 1 shows the possible suffixes for interface nodes. Triggers may have a void condition primitive (if set to `none`), while the event and the action part can not be null; for simplicity, conditions will be considered as part of the action. Therefore, the `Ac` suffix can connect action macroes as well as condition macroes.

Interface	Suffix	Graph element
In	Wf	Primary workflow
	Ev	Event macro
	Ac	Action or condition macro
Out	Wf	Primary workflow
	Ac	Action or condition macro
	True/False	Action macro

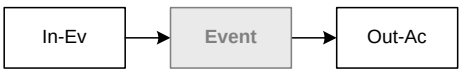
Table 1. Possible interface suffixes for specifying the kind of graph element to connect to a given macro element

Connecting two macroes requires connecting the output interface of the first macro to the input interface of the second macro. In order to be connectable, the two interfaces must present the same suffix. Interfaces are modelled by means of *task* nodes, because task nodes enable more flexible connection configurations than *transitions*. Transitions, in fact, would allow only one connection through an interface, while the auxiliary tasks allow more than one incoming transition to be connected to a specific interface.

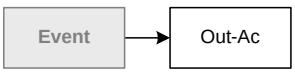
5.2 ECA Macro Elements

We now consider the generic, minimal structure of the three high-level macroes representing events, conditions and actions.

Events. Figure 2(a) depicts a generic macro for events with one input interface and one output interface. Actually, events should respect the macro structure of Figure 2(b), as generally events are not activated by incoming transitions. However, since we are tackling the problem of mapping events to XPDL process



(a) Macro with `In-Ev` and `Out-Ac` interfaces.



(b) Macro with `Out-Ac` interface.

Fig. 2. Generic macro for Chimera-Exception events

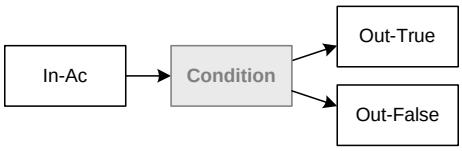


Fig. 3. Generic condition macro

definitions/graphs, events require an input interface for connecting them to the primary process flow, too. Indeed, as shown in Section 7.1, event macroes could also contain the event-generating logic, rather than representing the event itself.

Conditions. Figure 3 shows a generic condition macro with one input and two output interfaces. This macro is the only one using the suffixes `True` and `False` to denote the interface activated if the condition evaluates *true* or *false*.

The use of condition macroes requires the introduction of an exception to the general rule for connecting interfaces (see Section 5.1), as there is no input interface with `True` or `False` suffixes. Therefore, if condition macroes are used, their output interfaces can only be connected to interfaces with `Ac` suffixes.

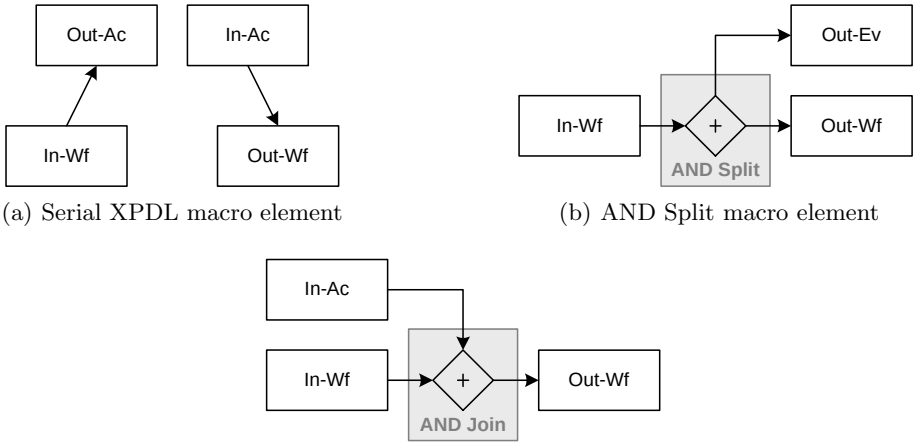
Fig. 4. Generic description of Chimera-Exception actions

Actions. Figure 4 describes the generic macro for actions. It has one input interface and one output interface, both to be connected to other `Ac` interfaces.

5.3 Basic Connector Macroes

In order to switch between the "process environment" and the "exception environment" within an enriched process definition, proper connector macroes are introduced. Figure 5 depicts the three basic connectors adopted in this paper.

Serial macro This macro (Figure 5(a)) allows splitting the normal process flow and inserting an exception handling sub-flow. The normal process is connected to the two interfaces with suffix `Wf`, while the two other interfaces connect to



(a) Serial XPDL macro element

(b) AND Split macro element

(c) AND Join macro element

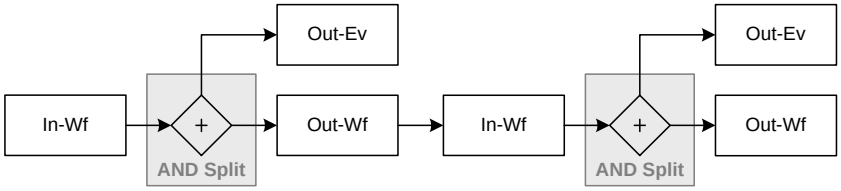
Fig. 5. Basic connector macroes

conditions or actions. The serial macro is used only to map exceptions triggered by workflow events, which are *synchronous* and *localized* (see Section 4). The serial connector may therefore act as both connector and triggering event.

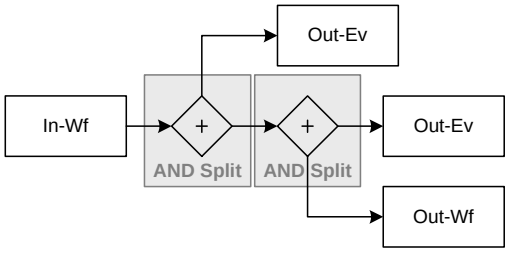
AND split The AND split (Figure 5(b)) starts parallel flows, one for the normal process execution, one for the exception handling. The output interface with suffix `Ev` is arranged for attaching proper event macroes; hence, the AND split does not represent an event. The AND split is particularly suited for mapping *asynchronous* or *synchronous, sparse* exceptions (see Section 4), when used in combination with event macroes that monitor the actual triggering event (i.e., data events, temporal events, external events).

AND join The AND join (Figure 5(c)) aims at joining a process flow and a parallel exception handling flow. The join represents the termination of a triggered rule and allows connecting in input an action or a condition macro.

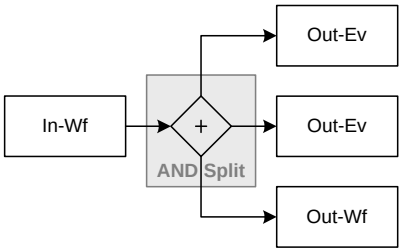
Process Graph Reduction The use of task nodes for interfaces leads to a high number of auxiliary tasks within the process graph. Input and output interfaces as well as suffixes are particularly useful for the consistent enrichment of the process graph with exception handling constructs. After the mapping process is terminated, the additionally introduced interface nodes are no longer needed and the enriched process graph can be optimized. Figure 6 exemplifies the graph reduction or optimization process relative to two connected AND splits: whenever the process graph contains an output interface connected to an input interface having the same suffix, the two nodes can be removed.



(a) Two connected AND splits



(b) Elimination of redundant In/Out tasks



(c) Reduced graph after optimization

Fig. 6. Process graph reduction

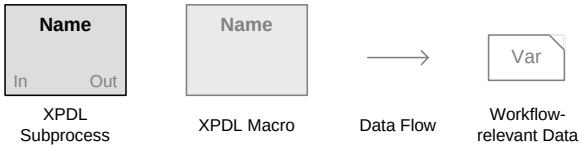


Fig. 7. Notation for modelling sub-processes and workflow-relevant data

6 Supporting Chimera-Exception Expressions

Chimera-Exception rules may contain complex expressions: we need some XPDL constructs capable of interpreting object-oriented primitives. We propose a solution based on sub-processes to provide basic and atomic functionalities for expression evaluation and data access (*basic sub-processes*). We also define *expression patterns* referring to particular configurations of workflow primitives and sub-processes, implementing higher-level functionalities with respect to those supported by sub-processes. Figure 7 depicts the notation for *sub-processes*, *data flows* and *workflow-relevant data fields*. Sub-processes exhibit the names of their input data, positioned at the left hand side, and of output data, at the right hand side. A workflow-relevant data field connected by an arrow to an input variable represents *consumed* values, while a field connected to an output variable gathers *produced* values. An appropriate use of workflow-relevant data fields therefore allows combining sub-processes and propagating parameter values.

6.1 Basic Sub-Processes for Expression Evaluation

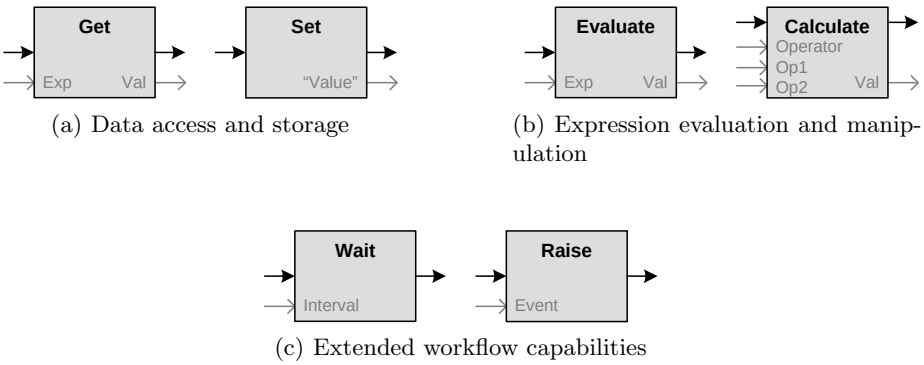
Figure 8 depicts some basic sub-processes. *Get(ExpRef)* accesses persistent data by object-oriented expressions (i.e., `Task1.Status`): *ExpRef* is a string variable referencing the variable to be read. *Set(Value,DataRef)* assigns values to workflow-relevant data fields: `Value` is the constant value to be assigned to variable referenced by *DataRef*. *Evaluate(PredRef)* evaluates Chimera-Exception condition predicates and returns the evaluation result that can be stored within a workflow-relevant data field: *PredRef* references the predicate to be evaluated. *Calculate(OperatorRef,Op1Ref,Op2Ref)* computes the 4 basic arithmetic operations, referenced by *OperatorRef*, over the operands referenced by *Op1Ref*, *Op2Ref*. *Wait(IntRef)* implements a temporal delay, where the termination of the sub-process signals the expiration of the delay referenced by *IntRef*. *Raise(Event)* implements the WfMS-specific logic required to detect external events, described by a unique name; the termination of the sub-process implies the occurrence of the specified external event.

Also proper sub-processes for Chimera-Exception *actions* are required. Actions strongly depend on the adopted WfMS: each action requires a customized sub-process, implementing the respective functionality on top of the chosen WfMS.

6.2 Expression Patterns

Complex *expression patterns* can now be defined. Expression patterns and basic sub-processes are the starting point for the automatic translation of Chimera-Exception triggers into XPDL.

Figure 9 shows an example of a workflow pattern `GetDataObject` mapping to XPDL Chimera-Exception expressions like `Task1.Status`. As shown, the pattern is composed of one `Set` operation and one `Get` operation. The former assigns the constant value ‘‘`Task1.Status`’’ to a workflow-relevant data field (`Exp`); the



(a) Data access and storage

(b) Expression evaluation and manipulation

(c) Extended workflow capabilities

Fig. 8. Basic subprocesses for evaluating Chimera-Exception expressions

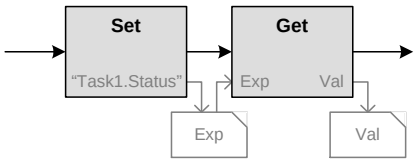


Fig. 9. Expression pattern for accessing persistent workflow data and status information

latter evaluates the expression and accesses the respective data. The retrieved value is then stored within the data field `Val`.

Besides data access patterns, other patterns manipulate strings, correctly instantiate `Wait` sub-processes, evaluate Chimera-Exception predicates...

By combining and concatenating basic sub-processes, according to such expression patterns, we build the specification and translation of complex expressions, as exemplified by Figure 10, where the expression `‘‘Status of Process = ’’ + C.Task1.Status` is expressed by proper sub-processes. For this purpose, two expression patterns are used, one for accessing persistent data within the underlying data source, and one for the concatenation of generic strings.

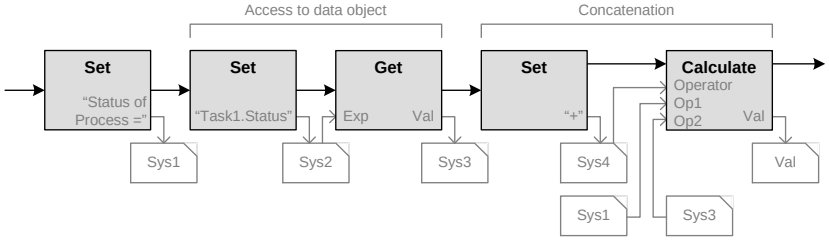


Fig. 10. Mapping of the expression `‘‘Status of Process = ’’ + C.Task1.Status`

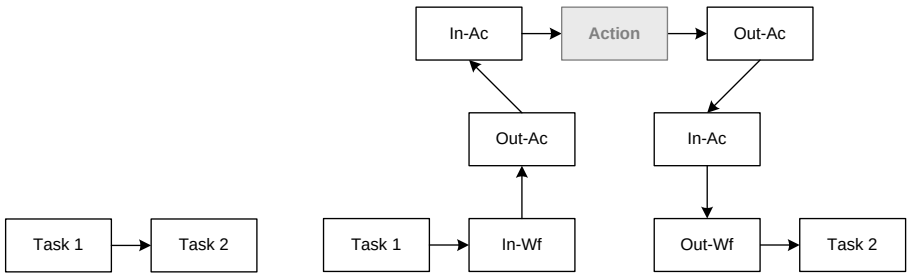
The example in Figure 10 further shows how sub-processes are configured and connected automatically. The constant values used by the `Set` operations within the depicted workflow fragment can be directly taken from the expression to be translated, while the passing of parameters from one sub-process to another can be achieved by means of system-generated data fields (i.e., `Sys1` or `Sys2`). The result of the described chain of sub-processes is stored within the data field `Val`. System-generated data fields can be derived at compilation time in a completely automated way, starting from the expression to be translated and the set of known expression patterns.

7 Mapping Exceptions to XPDL

The mapping of expressions can be achieved in a completely *process-independent* way, while the mapping of the high-level ECA constructs can only be accomplished in a *process-dependent* way. Sub-processes can directly be translated into proper XPDL constructs for building up *event*, *condition* and *action* macroes, which then are connected to the process graph by means of the already mentioned interface nodes. In this section, we describe in more detail how ECA macroes can be built, starting from basic sub-processes, and how they can be connected to the process graph.

7.1 Mapping Events

Events are the starting point for the evaluation of ECA rules and directly depend on the kind of exception. We have workflow, data, temporal, and external events.



(a) Start configuration: two subsequent tasks

(b) After insertion of the `Serial` macro

Fig. 11. Mapping of the `taskEnd(Task1)` workflow event

Workflow Events. Workflow events are *synchronous*, *localized*, and are mapped by the *serial* connector macro described in Section 5.3. Figure 11(b) shows the fragment of Figure 11(a) after the mapping of the event `taskEnd(Task1)`. The

serial connector macro plays the twofold role of connector and event. The two interface nodes available after the mapping process directly allow the connection of action or condition macroes. Events `taskEnd()`, `caseStart()` and `caseEnd()` map analogously.

Data Events. Data events are *asynchronous*, not localized, and mapped by an *AND split* connector, which splits the process flow immediately after the *start node* into two parallel flows, one for the primary process, one for the handling of the asynchronous exception.

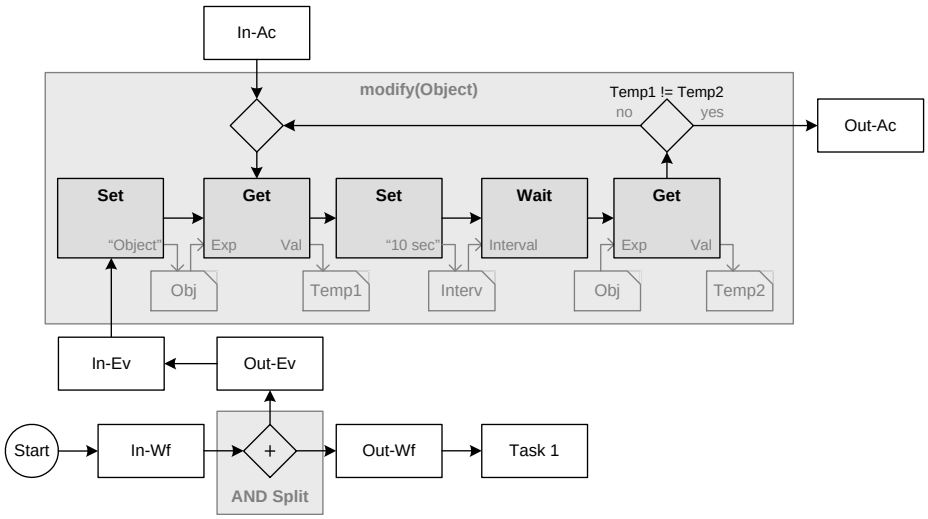


Fig. 12. Mapping of the `modify(Object)` data event

Figure 12 exemplifies the mapping of the data event `modify(Object)` and its connection to the process graph by means of the AND split connector (inside the grey-shaded box at the bottom of the figure). The actual event macro is connected to the `Out-Ev` interface of the connector by means of an `In-Ev` interface. The `modify(Object)` event further presents two other interfaces towards condition or action macroes (`Out-Ac` and `In-Ac`). The former enables the execution of the rule's actions in case of modifications of the data object `Object`, the latter re-connects the output interface of the action macro to the event macro.

The event macro in itself, rather than representing a real event, contains the runtime logic required for "generating" actual data events. This is achieved by cyclic monitoring (polling) of the respective data attribute within the underlying data source and transferring control to the condition and action macroes, in case a modification is detected. In such a case, re-connecting the output interface of the action macro to the `In-Ac` interface of the event macro closes the polling cycle, which is suspended for the whole time taken to execute the rule's actions.

Internally, the `modify(Object)` event consists in the following steps: the current value of the data object to be monitored is retrieved and stored in a workflow-relevant data field (`Temp1`). After the expiration of a predefined time interval, the (possibly changed) value of the monitored data object is stored in a second data field (`Temp2`). The two stored values are compared and, if they differ, the `Out-Ac` interface is enabled and the rule's condition and action parts are evaluated. Otherwise, the macro continues polling the monitored data object. The `delete` and `create` events map analogously.

Temporal Events. Temporal events require different mapping techniques, mainly based on their synchronicity features.

- *Instant events* are asynchronous with respect to the normal process flow. Their implementation is based on the use of suitable `Wait` operations;
- *Periodic events* are asynchronous, thus mapping similarly to instant events;
- *Interval events* are synchronous events as their evaluation depends on workflow events. The event `elapsed(interval 1 day) since taskStart(Production)`, for instance, is bound to the *anchor* event start of task `Production`. These events are mapped by branching the normal process flow according to the anchor event and by connecting the event handling constructs to the parallel control flow. After the expiration of the time interval, the actual event occurs, and the respective ECA rule is evaluated.

External Events. External events are asynchronous. The implementation of the appropriate detection mechanism strongly depends on the particular used WfMS. The detection of external events requires a proper sub-process (Section 6.1), which is in charge of waiting for the incoming event. After receiving the notification of the occurrence of the external event, the sub-process terminates, thus signalling the occurrence of the event. The respective macro for raising asynchronous, external events is based on cyclic activations of this sub-process, and is attached in parallel to the process graph after the start task. A more detailed treatment of external events exceeds the scope of this paper.

7.2 Mapping Conditions

The mapping of a condition involves as many instances of the `Evaluate` subprocess (introduced in Section 6.1) as predicates in the condition expression. The single intermediate evaluations are stored as workflow-relevant data, while a final condition primitive determines the overall evaluation.

Figure 13 exemplifies the described mapping of condition expressions: the `Set` operation stores the predicate to be evaluated (a string constant) in the data field `Expr`, which is then consumed as input by the `Evaluate` operation. This operation parses the predicate and stores the result in a system-generated data field (`Sys`). The actual result of the condition macro is computed by the final condition primitive.

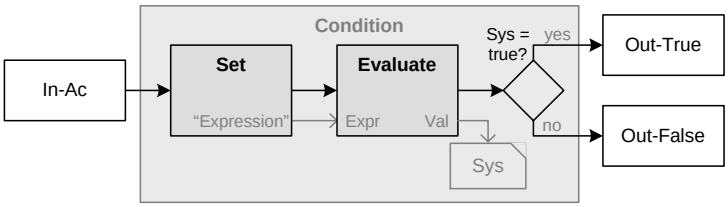


Fig. 13. XPDL macro element for expressing conditions

7.3 Mapping Actions

Action implementation must be provided by basic sub-processes prior to the mapping process. In order to form proper action macroes and to connect such sub-processes to the process graph, action sub-processes are wrapped by means of an `In-Ac` interface and an `Out-Ac` interface. Possible parameters are then specified by means of complex Chimera-Exception expressions, interpreted, and made available to the execution environment by workflow-relevant data fields.

In this way, the actions described in Section 4.1 can be seamlessly integrated into the enriched process definition. While the translation of exception triggers from Chimera-Exception to XPDL only requires the definition of the *interface* of the actions to be mapped, their execution on top of a particular WfMS requires their *implementation* according to the WfMS's extension mechanisms.

8 Compiling Chimera-Exception into XPDL

As a proof of concept, we developed a Java prototype of a *Compiler* for the automatic mapping into XPDL of exceptions defined in the Chimera-Exception language. An additional *Optimizer* module is in charge of reducing the enriched process graph by eliminating useless interface nodes.

The prototype demonstrates both the feasibility of the envisioned automatic translation of Chimera-Exception triggers into XPDL at the process definition level, and that the enriched process definitions are effectively executable by real workflow engines, if proper WfMS-specific sub-processes are provided.

8.1 XPDL Process Modelling and Rule Compilation

To validate the compilation process, the open-source, graphical workflow process editor JaWE [18] has been adopted. JaWE enables the visual specification of workflow processes and their storage in XPDL as native file format. The process definition example of Figure 1 is now modelled in JaWE by the swim lanes *Sales Office* and *Production* of Figure 14: white nodes represent *routing nodes* (automatically performed), while the gray shaded nodes represent the actual tasks to be executed by the resource/role associated to the respective swim lane.

Several trigger definitions have been compiled to test the compilation process: as an example, we consider here the `CustomerCancel` trigger described in Section

4.1 (Figure 14). The triggering event is an asynchronous data event, occurring after changes to the workflow-relevant data field `CustomerCalledToCancel`. If a modification to this parameter occurs, the case is aborted by activating the `CompensateAndDecline` task.

The enabling event `modify(CustomerCalledToCancel)` has been connected in parallel to the primary process flow by branching the primary flow immediately after the start node. According to Figure 12, the data event is implemented by a polling mechanism, monitoring the value of the workflow-relevant data field `CustomerCalledToCancel` and represented in figure by the tasks `Get`, `Wait` and the first `Cond` task. The second `Cond` task maps the conditional predicate `OrderManagement(O)`, `occurred(modify(CustomerCalledToCancel),O)`, and `O.CustomerCalledToCancel = ''Yes''`, while the specified action `startTask(''CompensateAndDecline'')` is achieved by connecting one of the outgoing transitions of the `Route` node to the respective task.

The evaluation of the execution of enriched process definitions has been tested by means of the open-source Java workflow engine Shark [19], which is completely based on XPDL as native process definition format. In order to execute enriched process definitions, the basic sub-processes introduced throughout this paper have been implemented as "Java applications" to be associated to the so-called `application tasks` (performed by means of predefined applications). The performed tests allow us to cover successfully both compilation and execution of enriched process definitions.

9 Conclusions and Future Work

In this paper we considered a relevant aspect of modern workflow management systems, i.e., exception handling. We leveraged the use of a high-level specification language for the definition of workflow exceptions, such as Chimera-Exception, and proposed a fully automated translation technique for rule definitions. According to our approach, rules are translated into XPDL, the standard process definition language proposed by the Workflow Management Coalition, which hence fosters portable process and exception handling specifications. As a proof-of-concept, the implemented rule compiler prototype allowed us to test all the stages of the compilation process. The main contribution of this paper is that it keeps the definition of exceptions at a conceptual level, by assuring – at the same time – portability.

As future research directions, we plan a formal analysis of the *complexity* of the outlined mapping approach, which however we expect being *linear* since there are no interdependencies between the mappings of two triggers. We are also confident about the *termination* of enriched processes, assuming a correct termination of the input processes and of the Chimera-Exception triggers to be mapped.

References

1. The Workflow Management Coalition. http://www.wfmc.org (2005)

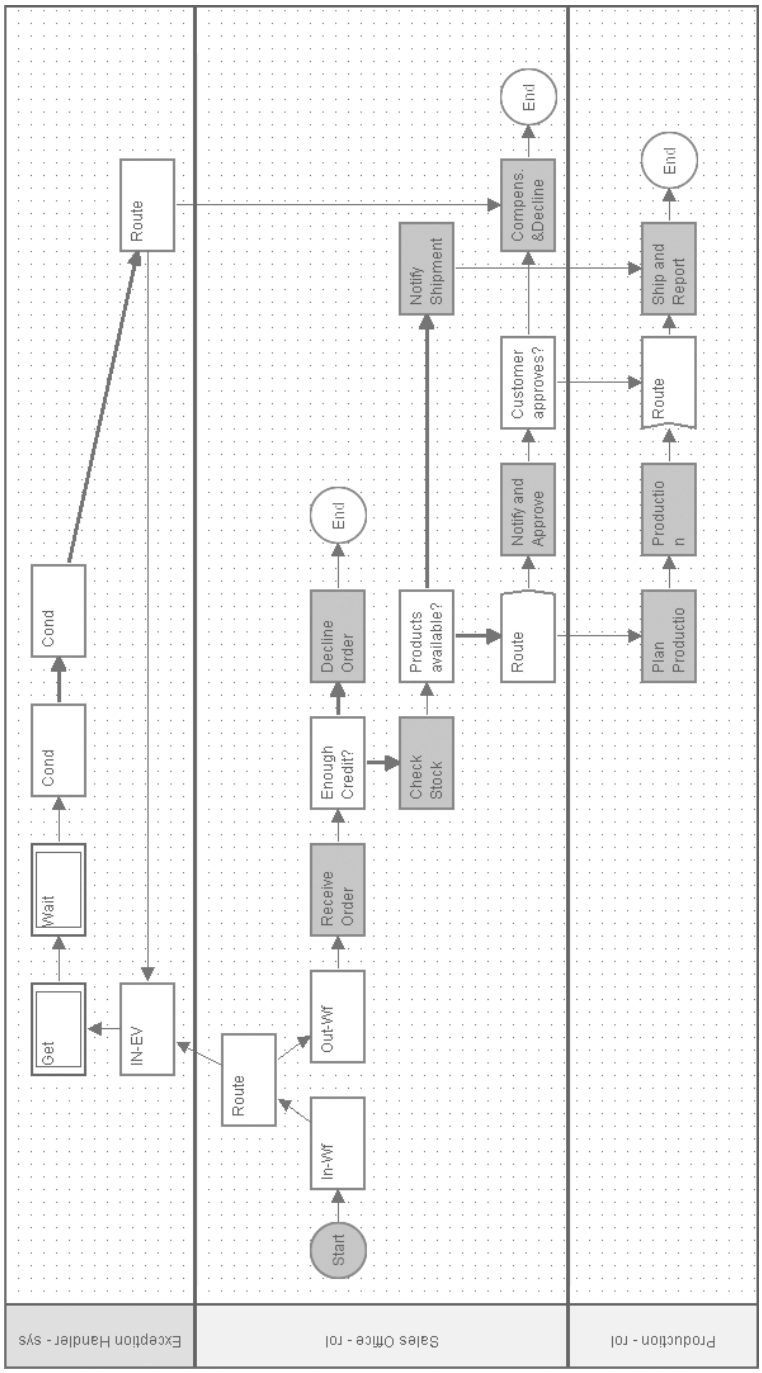


Fig. 14. Process model after compilation of the `CustomerCancel` trigger

2. Combi, C., Pozzi, G.: Architectures for a Temporal Workflow Management System. In: SAC '04: Proceedings of the 2004 ACM symposium on Applied computing, New York, NY, USA, ACM Press (2004) 659–666
3. Eder, J., Liebhart, W.: The Workflow Activity Model WAMO. In: CoopIS. (1995) 87–98
4. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and Implementation of Exceptions in Workflow Management Systems. ACM Transactions on Database Systems **24** (1999) 405–451
5. Mourão, H., Antunes, P.: Exception handling through a workflow. In Meersman, R., Tari, Z., eds.: CoopIS/DOA/ODBASE (1). Volume 3290 of Lecture Notes in Computer Science., Springer (2004) 37–54
6. Golani, M., Gal, A.: Flexible business process management using forward stepping and alternative paths. In van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F., eds.: Business Process Management. Volume 3649. (2005) 48–63
7. Reichert, M., Dadam, P.: $\text{Adept}_{\text{flex}}$-supporting dynamic changes of workflows without losing control. J. Intell. Inf. Syst. **10** (1998) 93–129
8. Luo, Z., Sheth, A.P., Kochut, K., Arpinar, I.B.: Exception handling for conflict resolution in cross-organizational workflows. Distributed and Parallel Databases **13** (2003) 271–306
9. Schuschel, H., Weske, M.: Triggering replanning in an integrated workflow planning and enactment system. In Gottlob, G., Benczúr, A.A., Demetrovics, J., eds.: ADBIS. Volume 3255 of Lecture Notes in Computer Science., Springer (2004) 322–335
10. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data Knowl. Eng. **53** (2005) 129–162
11. Hagen, C., Alonso, G.: Flexible Exception Handling in the OPERA Process Support System. In: ICDCS. (1998) 526–533
12. Baan Company N.V. - COSA Soultions: COSA Reference Manual (1998)
13. McCarthy, D., Sarin, S.: Workflow and transactions in In-Concert. IEEE Data Engineering, 16(2):5356 (1993)
14. Staffware Corporation: Staffware for Intranet based Workflow Automation. http://www.staffware.com/home/whitepapers/data/globalwp.htm (1997)
15. (BPMI.org), B.P.M.I.: Business Process Modeling Notation - Version 1.0. www.bpmi.org (2004)
16. Casati, F., Pozzi, G.: Modeling Exceptional Behaviors in Commercial Workflow Management Systems. In: CoopIS. (1999) 127–138
17. Leban, B., McDonald, D.D., Forster, D.R.: A Representation for Collections of Temporal Intervals. In: Proceedings of the Conference on AAA-I, (AAAI86, Philadelphia, PA) (1986) 367–371
18. ObjectWeb Consortium: Enhydra JaWE (Java Workflow Editor). http://jawe.objectweb.org/ (2005)
19. ObjectWeb Consortium: Enhydra Shark. http://shark.objectweb.org/ (2005)