Workflow Exception Patterns*

Nick Russell[1], Wil van der Aalst[2,1], and Arthur ter Hofstede[1]

[1] School of Information Systems, Queensland University of Technology,
GPO Box 2434, Brisbane QLD 4001, Australia
{n.russell, a.terhofstede}@qut.edu.au
[2] Department of Technology Management, Eindhoven University of Technology,
PO Box 513, NL-5600 MB, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

Abstract. This paper presents a classification framework for workflow exception handling in the form of patterns. This framework is independent of specific modelling approaches or technologies and as such provides an objective means of delineating the exception-handling capabilities of specific workflow systems. It is subsequently used to assess the level of exceptions support provided by eight commercial workflow systems and business process modelling and execution languages. On the basis of these investigations, we propose a graphical, tool-independent language for defining exception handling strategies in workflows.

1 Introduction

Business process management continues to receive widespread focus by technology-enabled organisations offering them a means of optimising their current organisational business processes in a way that aligns with top-level business objectives. In many cases, workflow systems serve as the enabling technology for mission-critical business processes. They offer a means of streamlining such processes by mapping out the key activities, decision points and work distribution directives and then automating much of the overhead that is often associated with managing the various activities which form part of a business process.

Workflow systems are generally based on a comprehensive process model (often depicted in graphical form) that maps out all of the possible execution paths associated with a business process. This ensures that the work activities which comprise each of the likely execution scenarios are fully described. Whilst this approach to specifying business process works well for *well-behaved* cases of a process i.e. those that conform to one of the expected execution paths, it is less successful in dealing with unexpected events encountered during execution.

Deviations from normal execution arising during a business process are often termed *exceptions* in line with the notion of exceptions which is widely used in the

* This work was partially supported by the Dutch research school BETA as part of the *PATINT* program and the Australian Research Council under the Discovery Grant *Expressiveness Comparison and Interchange Facilitation between Business Process Execution Languages.*

E. Dubois and K. Pohl (Eds.): CAiSE 2006, LNCS 4001, pp. 288–302, 2006.

software engineering community. Because it is difficult to characterise all of the unanticipated situations that may arise during the execution of a program, the notion of exceptions was developed where unexpected events are grouped into classes which are related by similarities that they possess in terms of the conditions under which they might arise. Exception handlers can then be defined in the form of programmatic procedures to resolve the effects of specific events as they are detected. At the lowest level, exceptions can be defined for events such as divide by zero errors and appropriate handling routines can be defined. For business processes, this level of detail is too fine-grained and it is more effective to define exceptions at a higher level, typically in terms of the business process to which they relate.

In this paper, we investigate the range of issues that may lead to exceptions during workflow execution and the various ways in which they can be addressed. This provides the basis for a classification framework for workflow exception handling which we subsequently define in the form of patterns. The patterns-based approach to exception classification is a continuation of previous research conducted as part of the *Workflow Patterns Initiative* which has identified "generic, recurring constructs" in the control-flow [22], data [18] and resource [19] perspectives of workflow systems. These patterns have proven to be extremely intuitive to both practitioners and researchers alike and have been widely utilised for a variety of purposes including tool evaluation and selection, business process modelling, workflow design and education[1]. They also provide the conceptual foundations for the YAWL system [21], an open-source reference implementation of a workflow system.

In line with the broader *Workflow Patterns Initiative*, the motivation for this paper is to provide a conceptual framework for classifying the exception handling capabilities of workflow systems and process-aware information systems more generally in a manner that is independent of specific modelling approaches or technologies. This approach is distinguished from other research activities in this area which seek to extend specific process modelling formalisms and workflow enactment technologies to provide support for expected and unexpected events by incorporating exception detection and handling capabilities. Instead of directly proposing a concrete implementation, we first provide an overview of relevant exception patterns, then we evaluate existing products and languages on the basis of these, and finally we propose a graphical, tool-independent language for exception handling.

2 Related Work

The need for reliable, resilient and consistent workflow operation has long been recognised. Early work in the area [8, 24] was essentially a logical continuation of database transaction theory and focussed on developing extensions to the classic ACID transaction model that would be applicable in application areas requiring the use of long duration and more flexible transactions. As the field

[1] Further details are available at `www.workflowpatterns.com`.

of workflow technology matured, the applicability of exceptions to this problem was also recognised [20] and [7] presented the first significant discussion on workflow recovery which incorporated exceptions. It classified them into four types: basic failures, application failures, expected exceptions and unexpected exceptions. Subsequent research efforts have mainly concentrated on the last two of these classes. Investigations into expected exceptions have focussed previous work on transactional workflow into mechanisms for introducing exception handling frameworks into workflow systems. Research into unexpected exceptions has established the areas of adaptive workflow and workflow evolution [17].

Although it is not possible to comprehensively survey these research areas in the confines of this paper, it is worthwhile identifying some of the major contributions in these areas that have influenced subsequent research efforts and have a bearing on this research initiative. Significant attempts to include advanced transactional concepts and exception handling capabilities in workflow systems include WAMO [6] which provided the ability to specify transactional properties for tasks which identified how failures should be dealt with, ConTracts [16] which proposed a coordinated, nested transaction model for workflow execution allowing for forward, backward and partial recovery in the event of failure and Exotica [2] which provided a mechanism for incorporating Sagas and Flexible transactions in the commercial FlowMark workflow product. OPERA [10] was one of the first initiatives to incorporate language primitives for exception handling into a workflow system and it also allowed exception handling strategies to be modelled in the same notation as that used for representing workflow processes. TREX [23] proposed a transaction model that involves treating all types of workflow failures as exceptions. A series of exception types were delineated and the exception handler utilised in a given situation was determined by a combination of the task and the exception experienced. WIDE [4] developed a comprehensive language – Chimera-Exc – for specifying exception handling strategies in the form of Event-Condition-Action (ECA) rules.

Other important contributions include [13] which identified the concepts of compensation spheres and atomicity spheres and their applicability to workflow systems, [3] which proposed modelling workflow systems as a set of reified objects with associated constraints and conceptualising exceptions as violations of those constraints which are capable of being detected and managed and [15] which first identified the pivot, retriable and compensation transaction concepts widely used in subsequent research.

Identifying potential exceptions and suitable handling strategies is a significant problem for large, complex workflows. Recent attempts [9, 11] to address this have centred on mining execution logs to gain an understanding of previous exceptions and using this knowledge to establish suitable handling strategies. [12] proposes a knowledge-based solution based on the establishment of a shared, generic and reusable taxonomy of exceptions. [14] uses a case-based reasoning approach to match exception occurrences with suitable handling strategies.

Until recently the area of unexpected exceptions has mainly been investigated in the context of adaptive or evolutionary workflow [17] which centre on dynamic

change of the process model. A detailed review of this area is beyond the scope of this paper, however two recent initiatives which offer the potential to address both expected and unexpected exceptions simultaneously are ADOME-WFMS [5] which provides an adaptive workflow execution model in which exception handlers are specified generically using ECA rules providing the opportunity for reuse in multiple scenarios and user-guided adaptation where they need refinement, and [1] which describes a combination of "worklets" and "ripple-down rules" as a means of dynamic workflow evolution and exception handling.

3 A Framework for Workflow Exception Handling

In this section we consider the notion of a workflow exception in a general sense and the various ways in which they can be triggered and handled. The assumption is that an exception is a distinct, identifiable event which occurs at a specific point in time during the execution of a workflow and relates to a unique work item[2]. The occurrence of the exception is assumed to be immediately detectable as is the type of the exception. The manner in which the exception is handled will depend on the type of exception that has been detected. There are a range of possible ways in which an exception may be dealt with but in general, the specific handling strategy centres on three main considerations:

- how the work item will be handled;
- how the other work items in the case will be handled; and
- what recovery action will be taken to resolve the effects of the exception.

We discuss the range of possible exception types and the options for handling them in the following sections.

3.1 Exception Types

It is only possible to specify handlers for *expected* types of exception. With this constraint in mind, we undertook a comprehensive review of the workflow literature and current commercial workflow systems and business process modelling and execution languages in order to determine the range of exception events that are capable of being detected and provide a useful basis for recovery handling. These events can be classified into five distinct groups.

Work Item Failure: Work item failure during the execution of a workflow process is generally characterised by the inability of the work item to progress any further. This may manifest itself in a number of possible forms including a user-initiated abort of the executing program which implements the work item, the failure of a hardware, software or network component associated with the work item or the user to whom the work item is assigned signalling failure to

[2] We recognise that exceptions may also be bound to groups of tasks, blocks or even entire cases, and in these situations we assume that the same handling considerations apply to all of the encompassed tasks.

the workflow engine. Where the reason for this failure is not captured and dealt within the process model, it needs to be handled elsewhere in order to ensure that both later work items and the process as a whole continue to behave correctly.

Deadline Expiry: It is common to specify a deadline for a work item in a workflow process model. Usually the deadline indicates when the work item should be completed, although deadlines for commencement are also possible. In general with a deadline, it is also useful to specify at design time what should be done if the deadline is reached and the work item has not been completed.

Resource Unavailability: It is often the case that a work item requires access to one or more data resources during its execution. If these are not available to the work item at initiation, then it is usually not possible for the work item to proceed. Similarly, workflow systems are premised on the fact that work items are usually allocated to resources (typically human) who execute them. Problems with work item allocation can arise if: (1) at distribution time, no resource can be found which meets the specified allocation criteria for the work item or (2) at some time after allocation, the resource is no longer able to undertake or complete the work item. Although the occurrence of these issues can be automatically detected, they often cannot be resolved within the context of the executing process and may involve some form of escalation or manual intervention. For this reason, they are ideally suited to resolution via exception handling.

External Trigger: Triggers from sources external to a work item are often used as a means of signalling the occurrence of an event that impacts on the work item and requires some form of handling. These triggers are typically initiated by non-linked work items (i.e. work items that are not directly linked to the work item in question by a control edge) elsewhere within the process model or even in other process models or alternatively from processes in the operational environment in which the workflow system resides. Although a work item can anticipate events such as triggers and provision for dealing with them can be included at design-time, it is not predictable if or when such events will occur. For this reason, the issue of dealing with them is not suited to normal processing within the work item implementation and is better dealt with via exception handling. Generally signals or some other form of processing interrupt indicate that an *out-of-bound* condition has arisen and needs to be dealt with. A general consequence of this is that the current work item needs to be halted, possibly undone and some alternative action taken.

Constraint Violation: Constraints in the context of a workflow system are invariants over elements in the control-flow, data or resource perspectives that need to be maintained to ensure the integrity and operational consistency of the workflow process is preserved. Ongoing monitoring is generally required to ensure that they are enforced. The implementation of routines to identify and handle constraint violations detected within the context of a workflow is similar to the issue of dealing with external triggers. Typically the construct that will detect and need to deal with the violation is a work item although there is no reason why the constraint could not be specified and handled at block or process

level. As constraints may be specified over data, resources or other work items within a process model, the approach chosen for handling them needs to be as generic as possible to ensure that it has broadest applicability.

3.2 Exception Handling at Work Item Level

In general an exception will relate to a specific work item in a case. There are a multitude of ways in which the exception can be handled although the specific details will depend on the current state of execution of the work item. Before looking at these options, we first review the execution lifecycle for a work item. Figure 1 illustrates as solid arrows the states through which a work item progresses during normal execution. It is initially *offered* to one or more resources for execution. A resource issues an *allocate* command to indicate that it wishes to execute the work item at some future time, the work item is then *allocated* to that resource. Typically this involves adding the work item to the resource's work queue and removing any references to the work item that other resources may have received, either on their work queues or via other means. When the resource wishes to commence the work item, it issues a *start* command and the state of the work item changes to *started*. Finally, once the work item is finished, the resource issues a *complete* command and the state of the work item is changed to *completed*. Note that there are two possible variations to this course of events shown as dotted arcs in Figure 1: (1) where a work item offered to a resource is *selected* by another resource, it is *withdrawn* from the first resource's worklist and (2) where an executing work item is detected as having *failed*, its state is changed accordingly. This lifecycle map also provides the basis for determining what options exist for handling a work item in a given state when an exception is detected.

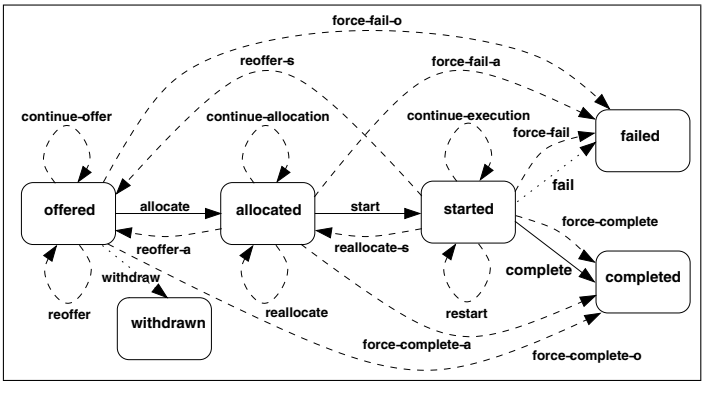

Fig. 1. Work item lifecycle

Figure 1 illustrates fifteen strategies as dashed arcs from one work item state to another. There are subtle differences between each of these transitions, and in order to distinguish between them, we briefly describe each of them:

1. **continue-offer (OCO)** – the work item has been offered to one or more resources and there is no change in its state as a consequence of the exception;
2. **reoffer (ORO)** – the work item has been offered to one or more resources and as a consequence of the exception, these offers are withdrawn and the work item is once again offered to one or more resources (these resources may not necessarily be the same as those to which it was offered previously);
3. **force-fail-o (OFF)** – the work item has been offered to one or more resources, these offers are withdrawn and the state of the work item is changed to failed. No subsequent work items on this path are triggered;
4. **force-complete-o (OFC)** – the work item has been offered to one or more resources, these offers are withdrawn and the state of the work item is changed to completed. All subsequent work items are triggered;
5. **continue-allocation (ACA)** – the work item has been allocated to a specific resource that will execute it at some future time and there is no change in its state as a consequence of the exception;
6. **reallocate (ARA)** – the work item has been allocated to a resource, this allocation is withdrawn and the work item is allocated to a different resource;
7. **reoffer-a (ARO)** – the work item has been allocated to a resource, this allocation is withdrawn and the work item is offered to one or more resources (this group may not necessarily include the resource to which it was previously allocated);
8. **force-fail-a (AFF)** – the work item has been allocated to a resource, this allocation is withdrawn and the state of the work item is changed to failed. No subsequent work items are triggered;
9. **force-complete-a (AFC)** – the work item has been allocated to a resource, this allocation is withdrawn and the state of the work item is changed to completed. All subsequent work items are triggered;
10. **continue-execution (SCE)** – the work item has been started and there is no change in its state as a consequence of the exception;
11. **restart (SRS)** – the work item has been started, progress on the current execution instance is halted and the work item is restarted from the beginning by the same resource that was executing it previously;
12. **reallocate-s (SRA)** – the work item has been started, progress on the current execution instance is halted and the work item is reallocated to a different resource for later execution;
13. **reoffer-s (SRO)** – the work item has been started, progress on the current execution instance is halted and it is offered to one or more resources (this group may not necessarily include the resource that was executing it);
14. **force-fail (SFF)** – the work item is being executed, any further progress on it is halted and its state is changed to failed. No subsequent work items are triggered; and
15. **force-complete (SFC)** – the work item is being executed, and further progress on it is halted and its state is changed to completed. All subsequent work items are triggered.

3.3 Exception Handling at Case Level

Exceptions always occur in the context of one or more cases that are in the process of being executed. In addition to dealing with the specific work item to which the exception relates, there is also the issue of how the case should be dealt with in an overall sense, particularly in regard to other work items that may currently be executing or will run at some future time. There are three alternatives for handling workflow cases:

1. **continue workflow case (CWC)** – the workflow case can be continued, with no intervention occurring in the execution of any other work items;
2. **remove current case (RCC)** – selected or all remaining work items in the case can be removed (including those currently executing); or
3. **remove all cases (RAC)** – selected or all remaining work items in all cases which correspond to the same process model can be removed.

In the latter two scenarios, a selection of work items to be removed can be specified using both static design time information relating to the corresponding task definition (e.g. original role allocation) as well as relevant runtime information (e.g. actual resource allocated to, start time).

3.4 Recovery Action

The final consideration in regard to exception handling is what action will be taken to remedy the effects of the situation that has been detected. There are three alternate courses of action:

1. **no action (NIL)** – do nothing;
2. **rollback (RBK)** – rollback the effects of the exception; or
3. **compensate (COM)** – compensate for the effects of the exception.

Rollback and compensation are analogous to their usual definitions (e.g. [15]). When specifying a rollback action, the point in the process (i.e. the task) to which the process should be undone can also be stated. By default this is just the current work item. Similarly with compensation actions, the corresponding compensation task(s) must also be identified.

3.5 Characterising Exception Handling Strategies

The actual recovery response to any given class of exception can be specified as a pattern which succinctly describes the form of recovery that will be attempted. Specific exception patterns may apply in multiple situations in a given process model (i.e. for several distinct constructs), possibly for different types of exception. Exception patterns take the form of tuples comprising the following elements:

- how the task on which the exception is based should be handled;
- how the case and other related cases in the process model in which the exception is raised should be handled; and
- what recovery action (if any) is to be undertaken.

Table 1. Exceptions patterns support by exception type

Work Item Failure	Work Item Deadline	Resource Unavailable	External Trigger	Constraint Violation
OFF-CWC-NIL	OCO-CWC-NIL	ORO-CWC-NIL	OCO-CWC-NIL	SCE-CWC-NIL
OFF-CWC-COM	ORO-CWC-NIL	OFF-CWC-NIL	OFF-CWC-NIL	SRS-CWC-NIL
OFC-CWC-NIL	OFF-CWC-NIL	OFF-RCC-NIL	OFF-RCC-NIL	SRS-CWC-COM
OFC-CWC-COM	OFF-RCC-NIL	OFC-CWC-NIL	OFC-CWC-NIL	SRS-CWC-RBK
AFF-CWC-NIL	OFC-CWC-NIL	ARO-CWC-NIL	ACA-CWC-NIL	SFF-CWC-NIL
AFF-CWC-COM	ACA-CWC-NIL	ARA-CWC-NIL	AFF-CWC-NIL	SFF-CWC-COM
AFC-CWC-NIL	ARA-CWC-NIL	AFF-CWC-NIL	AFF-RCC-NIL	SFF-CWC-RBK
AFC-CWC-COM	ARO-CWC-NIL	AFF-RCC-NIL	AFC-CWC-NIL	SFF-RCC-NIL
SRS-CWC-NIL	AFF-CWC-NIL	AFC-CWC-NIL	SCE-CWC-NIL	SFF-RCC-COM
SRS-CWC-COM	AFF-RCC-NIL	SRA-CWC-NIL	SRS-CWC-NIL	SFF-RCC-RBK
SRS-CWC-RBK	AFC-CWC-NIL	SRA-CWC-COM	SRS-CWC-COM	SFF-RAC-NIL
SFF-CWC-NIL	SCE-CWC-NIL	SRA-CWC-RBK	SRS-CWC-RBK	SFC-CWC-NIL
SFF-CWC-COM	SCE-CWC-COM	SRO-CWC-NIL	SFF-CWC-NIL	SFC-CWC-COM
SFF-CWC-RBK	SRS-CWC-NIL	SRO-CWC-COM	SFF-CWC-COM	
SFF-RCC-NIL	SRS-CWC-COM	SRO-CWC-RBK	SFF-CWC-RBK	
SFF-RCC-COM	SRS-CWC-RBK	SFF-CWC-NIL	SFF-RCC-NIL	
SFF-RCC-RBK	SRA-CWC-NIL	SFF-CWC-COM	SFF-RCC-COM	
SFC-CWC-NIL	SRA-CWC-COM	SFF-CWC-RBK	SFF-RCC-RBK	
SFC-CWC-COM	SRA-CWC-RBK	SFF-RCC-NIL	SFF-RAC-NIL	
SFC-CWC-RBK	SRO-CWC-NIL	SFF-RCC-COM	SFC-CWC-NIL	
	SRO-CWC-COM	SFF-RCC-RBK	SFC-CWC-COM	
	SRO-CWC-RBK	SFF-RAC-NIL		
	SFF-CWC-NIL	SFC-CWC-NIL		
	SFF-CWC-COM	SFC-CWC-COM		
	SFF-CWC-RBK			
	SFF-RCC-NIL			
	SFF-RCC-COM			
	SFF-RCC-RBK			
	SFC-CWC-NIL			
	SFC-CWC-COM			

For example, the pattern SFF-CWC-COM specified for a work item failure exception indicates that if a failure of a work item is detected after it has started, then the work item should be terminated, have its state changed to failed and the nominated compensation task should be invoked. No action should be taken with other work items in the same case. From the various alternatives identified for each of these elements in Sections 3.2 – 3.4, there are *135 possible patterns*. Not all patterns apply to a given exception type however, and Table 1 identifies those which apply to each of the exception types identified in Section 3.1.

4 Workflow Exception Handling in Practice

The exception patterns identified in Section 3 were used to assess the exception handling capabilities of eight workflow systems and business process modelling languages. The results of this survey[3,4] are captured in Table 2. They provide a salient insight into how little of the research into exception handling has been implemented in commercial offerings. Only deadline expiry enjoys widespread support although its overall flexibility is limited in many tools. Only two of the

[3] Full evaluation details are contained in report BPM-06-04 at `www.BPMcenter.org`

[4] Combinations of patterns are written as regular expressions e.g. (SFF|SFC)-CWC-COM represents the two patterns SFF-CWC-COM and SFC-CWC-COM.

Table 2. Support for exception patterns in commercial offerings

Offering	**Exceptions**			
	Work Item Failure	***Work Item Deadline***	***External Trigger***	***Constraint Violation***
Staffware Process Suite v9		OCO-CWC-COM ACA-CWC-COM OFF-CWC-COM AFF-CWC-COM SCE-CWC-COM	OCO-CWC-NIL ACA-CWC-NIL SCE-CWC-NIL SCE-CWC-COM	
WebSphere MQ 3.4 (IBM)		OCO-CWC-NIL ACA-CWC-NIL SCE-CWC-NIL		
FLOWer 3.1 (Pallas Athena)		AFC-CWC-NIL SFC-CWC-NIL		AFC-CWC-NIL SFC-CWC-NIL AFC-CWC-COM SFC-CWC-COM
COSA 5.1 (Transflow)	SFF-CWC-RBK	OCO-CWC-COM ACA-CWC-COM SCE-CWC-COM	OCO-CWC-COM ACA-CWC-COM SCE-CWC-COM	
iPlanet Integ. Server 3.1 (Sun)	(OFF|OFC|AFF|AFC|SRS|SFC|SFF)-(CWC|RCC)-(NIL|COM)			
XPDL 2.0 (WfMC)	SFF-CWC-COM SFF-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	SCE-CWC-COM SCE-CWC-NIL SFF-CWC-COM SFF-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	SFF-CWC-COM SFF-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	SFF-CWC-COM SFF-CWC-NIL SFF-RCC-COM SFF-RCC-NIL
BPEL 1.1	SFF-CWC-COM SFF-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	SCE-CWC-COM SCE-CWC-NIL SFF-CWC-COM SFF-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	SCE-CWC-COM SCE-CWC-NIL SFF-CWC-COM SFF-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	
BPMN 1.0 (BPMI)	SFF-CWC-COM SFF-CWC-NIL SFC-CWC-COM SFC-CWC-NIL SRS-CWC-COM SRS-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	SFF-CWC-COM SFF-CWC-NIL SFC-CWC-COM SFC-CWC-NIL SRS-CWC-COM SRS-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	SFF-CWC-COM SFF-CWC-NIL SFC-CWC-COM SFC-CWC-NIL SRS-CWC-COM SRS-CWC-NIL SFF-RCC-COM SFF-RCC-NIL	SFF-CWC-COM SFF-CWC-NIL SFC-CWC-COM SFC-CWC-NIL SRS-CWC-COM SRS-CWC-NIL SFF-RCC-COM SFF-RCC-NIL

workflow systems examined provide support for handling work items failures – generally via user-initiated aborts. There was also minimal support for external triggers and constraint violation management amongst the workflow tools with only Staffware and COSA, and FLOWer respectively supporting these exception classes. The business process languages (XPDL, BPEL and BPMN) provide better support across most areas although only for active work items. None of the offerings examined provided exception support for managing resource unavailability (and as a consequence this column has been omitted from Table 2 – this reflects other research findings [19] on the lack of support for the resource perspective in current commercial products.

5 Considerations for a Workflow Exception Language

The insights gained in the previous sections in relation to the identification and handling of workflow exceptions provide the basis for a general workflow

exception handling language. In this section, we propose a set of primitives for addressing exceptions that might arise during workflow execution and present a mechanism for integrating these primitives with the process model more generally. We then demonstrate the applicability of this approach to exception handling through a working example.

The conceptual model presented in Section 3 identified three key dimensions to handling an exception. These dimensions provide the basis for the primitives in the graphical exception language illustrated in Figure 2. Symbols 1–4, 8 and 12–14 are derived from the actions for dealing with the current work item from Figure 1, symbols 5–7 and 9–11 are derived from the options for dealing with other work items currently active in the same and other cases and symbols 15 and 16 correspond to the two forms of recovery action that can be undertaken. These primitives can be assembled into sequences of actions that define exception handling strategies. These sequences can also contain standard YAWL constructs [21] although we do not illustrate this capability here.

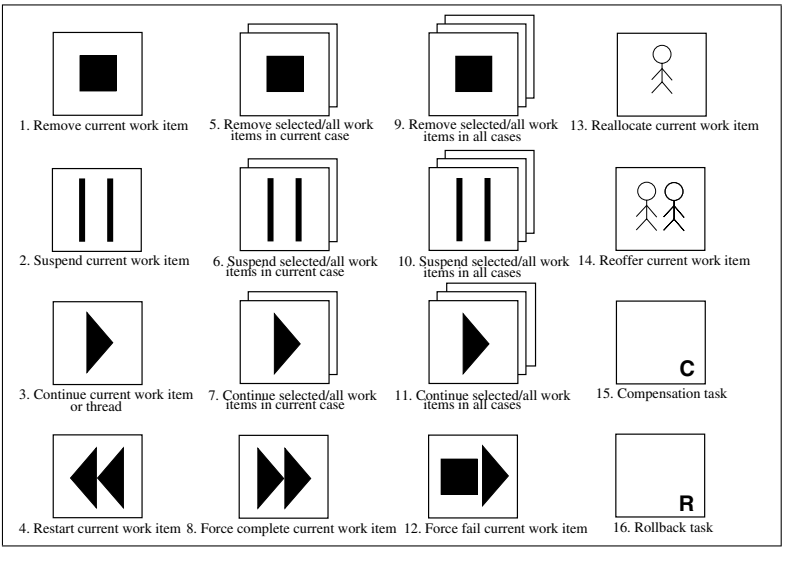

Fig. 2. Exception handling primitives

The interlinkage of exception handling strategies based on these primitives and the overall process model is illustrated in Figure 3. A clear distinction is drawn between the process model and the exception handling strategies. This is based on the premise that the process model should depict the normal sequence of activities associated with a business process and should aim to present these activities precisely without becoming overburdened by excessive consideration of unexpected events that might arise during execution. Exception handling strategies are able to be bound to one of five distinct workflow constructs: individual

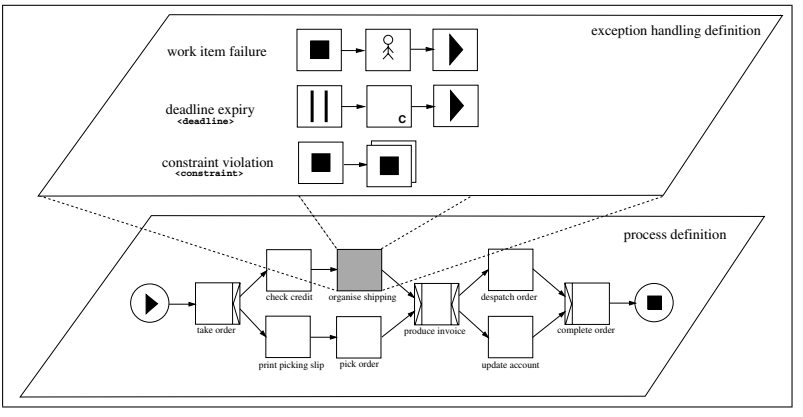

Fig. 3. Exception handling in relation to workflow processes

tasks, a scope (i.e. a group of tasks), a block, a process (i.e. all of the tasks in a process model) and a workflow (i.e. all of the process models in a given workflow environment). The binding is specific to one particular type of exception e.g. work item failure or constraint violation. It may also be further specialised using conditions based on elements from the data perspective e.g. there may be two exception handling strategies for a task, one for work items concerned with financial limits below $1000, the other with limits above that figure.

Exception handling strategies defined for more specific constructs take precedence over those defined at a higher level e.g. where a task has a work item failure exception strategy defined and there is also a strategy defined at the process-level for the same exception type, then the task-level definition is utilised should it experience such an exception. In order to illustrate the application of these concepts, we present an example based on the order fulfillment process illustrated in Figure 4 using the YAWL process modelling notation. In this process, orders are taken from customers, and a picking slip for the required items is prepared and subsequently used to select them from the warehouse. At the same time, the customer's credit is checked and shipping is organised for the order. When all of these tasks are complete an invoice is prepared for the customer and the goods are then packed and despatched whilst the customers account is updated with the outstanding amount. The order details are then finalised and filed.

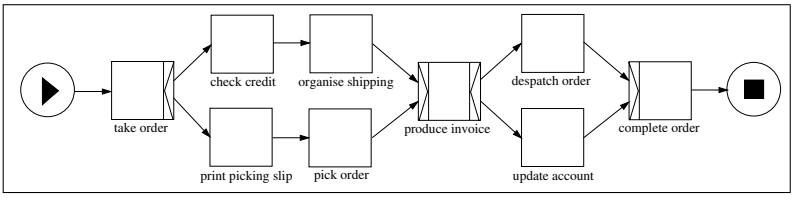

Fig. 4. Order despatch process

Figure 5(A) illustrates two alternate exception handling strategies for the *check credit* work item. If the credit required is less than $100, the current work item is suspended and the next work item is started. Where it is $100 or more, the current work item is suspended, the execution point is rewound to the beginning of the work item and it is recommenced. Figure 5(B) shows the exception handling strategy for the *pick order* work item where its completion deadline is not met. Recovery involves suspending the current work item, reassigning it to another resource, running a compensation task that determines if the order can be despatched within 48 hours (and if not applies a small credit to the account), then the *pick order* work item is restarted with the new resource. Figure 5(C) illustrates the resource unavailable handling strategy. Where the required resource is a data resource, this involves stopping the current work item and restarting it from the beginning. This strategy is bound to the process model i.e. by default, it applies to all work items. Where the unavailable resource is a human resource (i.e. the person undertaking the work item), the recovery action involves suspending the work item, reassigning it to another person and then restarting it from the beginning. Figure 5(D) indicates the approach to handling an *account frozen* trigger received by one of the tasks in the current process.

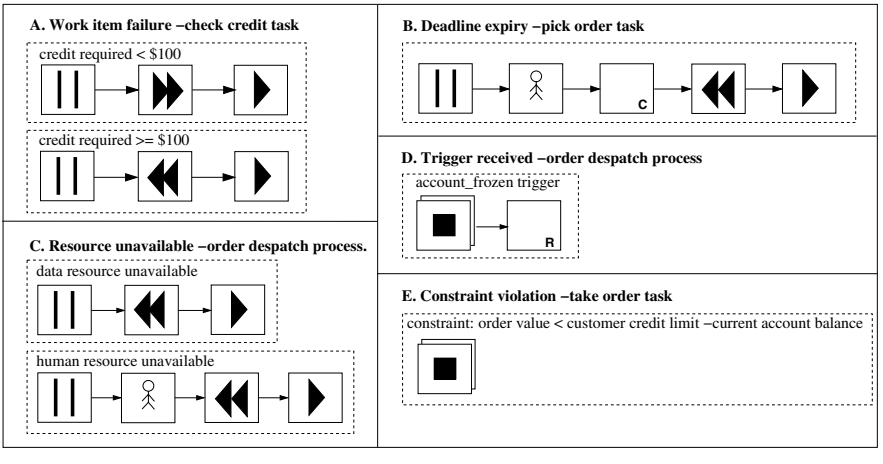

Fig. 5. Exception handling strategies – order despatch process

In this situation, the recovery action is to stop all work items in the case and to undertake a rollback action undoing all changes made since the case started. In other words, any work that has been undertaken on despatching goods to the customer is completely undone. Finally, Figure 5(E) illustrates the recovery action that is taken when the `order value` constraint is exceeded for the *take order* task. This involves stopping all work items associated with the process.

6 Conclusions

This paper has presented a patterns-based classification framework for characterising exception handling in workflow systems. The framework has been used to examine the capabilities of eight workflow systems and business process modelling and execution languages and has revealed the limited support for exception management in these offerings. As a consequence of the insights gained from these investigations, we have proposed a graphical, technology-independent language for defining exception handling strategies in workflows. This language offers the potential to assist in defining and managing deviations from normal process execution and will be the subject of further research in the context of exception handling in the YAWL reference implementation.

References

1. M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Facilitating flexibility and dynamic exception handling in workflows through worklets. In O. Belo, J. Eder, O. Pastor, and J. Falcao é Cunha, editors, *Proceedings of the CAiSE'05 Forum*, volume 161 of *CEUR Workshop Proceedings*, pages 45–50, Porto, Portugal, 2005. FEUP.
2. G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, G. Gunthor, and C. Mohan. Advanced transaction models in workflow contexts. In *Proceedings of the 12th International Conference on Data Engineering*, pages 574–581, New Orleans, USA, 1996.
3. A. Borgida and T. Murata. Tolerating exceptions in workflows: A unified framework for data and processes. In D. Georgakopoulos, W. Prinz, and A.L. Wolf, editors, *Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration (WACC'99)*, pages 59–68, San Francisco, USA, 1999.
4. F. Casati, S. Ceri, S. Paraboschi, and G. Pozzi. Specification and implementation of exceptions in workflow management systems. *ACM Transactions on Database Systems*, 24(3):405–451, 1999.
5. D.K.W. Chiu, Q. Li, and K. Karlapalem. ADOME-WFMS: Towards cooperative handling of workflow exceptions. In *Advances in Exception Handling Techniques*, pages 271–288. Springer-Verlag, New York, NY, USA, 2001.
6. J. Eder and W. Liebhart. The workflow activity model (WAMO). In S. Laufmann, S. Spaccapietra, and T. Yokoi, editors, *Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS-95)*, pages 87–98, Vienna, Austria, 1995.
7. J. Eder and W. Liebhart. Workflow recovery. In *Proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 124–134, Brussels, Belgium, 1996. IEEE Computer Society.
8. A. Elmagarmid, editor. *Database Transaction Models for Advanced Applications*. Morgan Kaufmann, San Mateo, CA, USA.
9. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving business process quality through exception understanding, prediction, and prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168, Rome, Italy, 2001. Morgan Kaufmann.

10. C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering*, 26(10):943–958, 2000.
11. S.Y. Hwang and J. Tang. Consulting past exceptions to facilitate workflow exception handling. *Decision Support Systems*, 37(1):49–69, 2004.
12. M. Klein and C. Dellarocas. A knowledge-based approach to handling exceptions in workflow systems. *Journal of Computer-Supported Collaborative Work*, 9(3-4): 399–412, 2000.
13. F. Leymann and D. Roller. Workflow-based applications. *IBM Systems Journal*, 36(1):102–123, 1997.
14. Z. Luo, A. Sheth, K. Kochut, and J. Miller. Exception handling in workflow systems. *Applied Intelligence*, 13(2):125–147, 2000.
15. S. Mehrotra, R. Rastogi, H.F. Korth, and A Silberschatz. A transaction model for multidatabase systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems (ICDCS'92)*, pages 56–63, Yokohama, Japan, 1992. IEEE Computer Society.
16. A. Reuter and F. Schwenkreis. ConTracts – a low-level mechanism for building general-purpose workflow management-systems. *Data Engineering Bulletin*, 18(1):4–10, 1995.
17. S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering*, 50:9–34, 2004.
18. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow data patterns: Identification, representation and tool support. In L. Delcambre, C. Kop, H.C. Mayr, J. Mylopoulos, and O. Pastor, editors, *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, volume 3716 of *LNCS*, pages 353–368, Klagenfurt, Austria, 2005. Springer.
19. N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In O. Pastor and J. Falcao é Cunha, editors, *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 216–232, Porto, Portugal, 2005. Springer.
20. D.M. Strong and S.M. Miller. Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems*, 13(2):206–233, 1995.
21. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.
22. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.
23. R. van Stiphout, T.D. Meijler, A. Aerts, D. Hammer, and R. Le Comte. TREX: Workflow transaction by means of exceptions. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Proceedings of the Sixth International Conference on Extending Database Technology (EDBT'98)*, pages 21–26, Valencia, Spain, 1998.
24. D. Worah and A.P. Sheth. Transactions in transactional workflows. In S. Jajodia and L. Kerschberg, editors, *Advanced Transaction Models and Architectures*, pages 3–34. Kluwer Academic Publishers, 1997.