Facilitating Flexibility and Dynamic Exception Handling in Workflows through Worklets

Michael Adams[1], Arthur H. M. ter Hofstede[1], David Edmond[1], and Wil M. P. van der Aalst[1,2]

[1] Centre for Information Technology Innovation
Queensland University of Technology, Brisbane, Australia
{m3.adams,a.terhofstede,d.edmond}@qut.edu.au
[2] Department of Technology Management
Eindhoven University of Technology, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

Abstract. Workflow management systems are implemented to support the modelling, analysis and enactment of rigidly structured business processes. However, they typically have difficulty supporting unexpected or developmental change occurring in the work practices they model, and providing adequate support for exceptions, or deviations from the process model, even though such deviations are a common occurrence for almost all processes. These limitations mean a large subset of business practices do not easily map to the inflexible modelling frameworks imposed, and so have inhibited wider acceptance.
This paper presents the basis of an approach for dynamic flexibility, evolution and exception handling in workflows through the support of flexible work practices, and based, not on proprietary frameworks, but on accepted ideas of how people actually work. A set of principles have been derived from a sound theoretical base and applied to the development of *worklets*, a repertoire of self-contained subprocesses that can be applied in a variety of situations depending on the context of the particular work instance.

1 Introduction

Organisations are constantly seeking to improve the efficiency and effectiveness of their business processes. One approach aimed at supporting these objectives has been the implementation of workflow management systems to configure and control business processes [1], by supporting their modelling, analysis and enactment [2]. Other key benefits workflow management systems seek to bring to an organisation include better process control, improved customer service, higher maintainability and business process evolution [3–5].

The use of workflow management systems has grown through their support for the modelling of rigidly structured business processes that in turn derive well-defined workflow instances [6, 7]. However, the frameworks imposed make it difficult to support (i) dynamic evolution (i.e. modifying process instances during execution) following unexpected or developmental change in the business processes being modelled [8]; and (ii)

deviations from the process model at runtime [2, 9]. These limitations mean a large subset of business processes do not easily map to the rigid modelling frameworks provided [10].

Without support for dynamic evolution, the occurrence of an unexpected deviation (or exception) requires either suspension of execution while the deviation is handled manually, or an entire process abort. However, since most processes are long and complex, neither intervention nor process termination are satisfactory solutions [11]. Manual exception handling incurs an added penalty: the corrective actions undertaken are not added to 'organisational memory' [12, 13], and so natural process evolution is not incorporated into future iterations of the process. Other evolution issues include problems of migration, synchronisation and version control [14, 15].

Thus the installation base of workflow management systems has been limited, due to the lack of flexibility inherent in a framework that, by definition, imposes rigidity. Process models are 'system-centric', or *straight-jacketed* [16] into the supplied framework, rather than truly reflecting the way work is actually performed. As a result, users are forced to work outside of the system, and/or constantly revise the static process model, in order to successfully support their activities, thereby negating the efficiency gains sought by implementing a workflow solution in the first place.

It is therefore desirable to extend the capabilities of WfMSs, so that the benefits offered to organisations employing rigidly defined, 'assembly-line' processes could also be enjoyed by those businesses which employ more flexible processes.

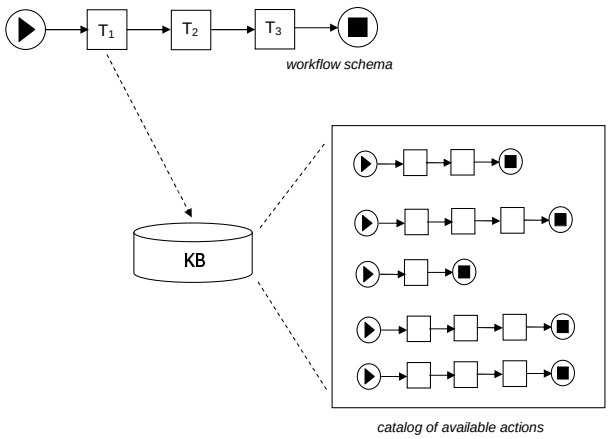

Fig. 1. Worklet Conceptual Diagram

This paper introduces the concept of *'worklets'*, a repertoire of self-contained subprocesses and associated selection and exception handling rules (fig. 1), to support the modelling, analysis and enactment of business processes, which is grounded in a formal set of principles of work practices called *Activity Theory*. We intend that this approach will directly provide for dynamic exception handling, ad-hoc change and process evolution, without having to resort to off-system intervention and/or system downtime.

This paper is organised as follows: Section 2 surveys issues of dynamic evolution and exception handling in workflows, provides a brief overview of Activity Theory and lists relevant principles derived from it, then introduces the worklet paradigm. Section 3 describes how the worklet approach utilises *Ripple Down Rules* to achieve contextual, dynamic selection of worklets at runtime. Section 4 presents the worklet dynamic selection process, while Section 5 describes the approach for dynamic exception handling. Section 6 discusses related work, and finally Section 7 outlines future directions and concludes the paper.

2 Achieving Flexibility through Worklets

Workflow management systems provide support for business processes that are generally predictable and repetitive. Process exceptions or deviations are largely uncatered for, even though dealing with them forms a substantial proportion of the everyday tasks carried out in an organisation [9, 17, 18]. Realistically, every executing instance of a work process will incorporate some deviation from the plan. Thus, formal representations of business processes may be said to provide merely a contingency around which tasks can be formulated dynamically [19]. In this sense, a work plan may be considered a resource which mediates activities towards their objective, rather than a prescriptive blueprint that must be strictly adhered to. Deviations from the plan should be considered as natural and valuable parts of the work activity, which provide the opportunity for learning and thus engender natural evolution of the plan. The prescriptive, assembly-line frameworks imposed by workflow management systems limit the ability to model and enact flexible work practices where exceptions to the rule are a normal part of every work activity [16]. Rather than continue to try to force business processes into inflexible frameworks (with limited success), a more flexible approach is needed that is based on accepted ideas of how people actually work.

A powerful set of descriptive and clarifying principles that describe how work is conceived, performed and reflected upon is *Activity Theory*, which focusses on understanding human activity and work practices, incorporating notions of intentionality, history, mediation, collaboration and development [20]. (An exploration of Activity Theory is beyond the scope of this paper; more details can be found in: [21, 22]). In [23], the authors undertook a detailed study of Activity Theory and derived from it a set of principles that describe the nature of participation in organisational work practices. Briefly, the relevant principles are:

1. Activities are *hierarchical* (consist of one or more actions), *communal* (involve a community of participants working towards a common objective), *contextual* (conditions and circumstances deeply affect the way the objective is achieved), *dynamic* (evolve asynchronously), and *mediated* (by tools, rules and divisions of labour).
2. Actions (i.e. tasks) are undertaken and understood contextually. A repertoire of actions is maintained and made available to any activity, which may be performed by making contextual choices from the repertoire.
3. A plan is not a prescription of work to be performed, but merely a guide which is modified during execution depending on context.

4. Exceptions are merely deviations from a plan, and will occur with every execution, giving rise to learning experiences which can then be incorporated into future instantiations of the plan.

Consideration of these derived principles have led to the conception of a flexible workflow support system that:

- regards the process model as a guide to an activity's objective, rather than a prescription for it;
- provides for a dynamic repertoire (or catalogue) of actions to be made available for each task at each execution of a process model;
- provides for choices to be made dynamically from the repertoire at runtime by considering the specific context of the executing instance; and
- allows those contextual choices to be made, not only for each task, but for appropriate exception handling techniques using the same selection and invocation mechanism, thus incorporating process exceptions, not only as part of the model, but as normal and valuable events that lead to system learning and therefore natural process evolution.

Each task of a process is linked to a repertoire of actions, one of which is contextually chosen at runtime to carry out the task. In this work, we present these repertoire-member actions as *"worklets"*. In effect, a worklet is a small, self-contained, complete workflow process which handles one specific task (action) in a larger, composite process (activity). A sequence of worklets are chained to form an entire workflow process. Note that in Activity Theory terms, a worklet may represent one action within an activity, or may represent an entire activity (for example, a top-level or manager worklet).

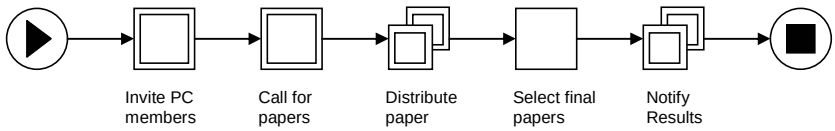

Fig. 2. Worklet: Coordinate Conference Publications (Top Level)

To denote the sequence of the composite process, a top-level or manager worklet is developed that captures the workflow at a macro level. From that manager process, worklets are contextually selected and invoked from the repertoire of each task (see *Section 4*). (Figure 2 shows a *Conference Proceedings* example of a top level worklet, using YAWL notation [24] – 'Select Final Papers' is an example of an atomic task, 'Call for Papers' a composite task and 'Distribute Paper' a multiple-instance composite task).

In addition, for each anticipated exception (an event that is not expected to occur in most instances), a complementary worklet for handling the event may be defined, to be dynamically incorporated into a running workflow instance on an as-needed basis (see *Section 5*). Further, worklets to handle these potential events are constructed in *exactly the same way* as those for standard processes. In the occurrence of an unanticipated

exception (i.e. an event for which a worklet has not yet been defined), then either an existing worklet can be selected from the repertoire, or one may be adapted on the fly to handle the immediate situation, allowing execution to continue. Most importantly, the method used to handle an exception is captured by the system, and so a history of the event and the method used to handle it is recorded for future instantiations. In this way, the process model undergoes a dynamic natural evolution. At the same time, a repertoire for each task is dynamically constructed as different approaches to completing a task are developed, derived from the context of each process instance.

3 Context and Worklet Selection

For any situation, there are multiple environmental, experiential and personal factors that may combine to influence a choice of action. Those factors are considered to be the *context* of the situation. The consideration of context plays a crucial role in many diverse domains, including philosophy, pragmatics, semantics, cognitive psychology and artificial intelligence [25]. In order to realise the worklet approach to workflow management, the situated contextual factors relevant to each case instance are required to be quantified and recorded [26] so that the appropriate worklet can be 'intelligently' chosen from the repertoire at runtime.

The types of contextual data that may be recorded and applied to a business case may be categorised as follows (examples are drawn from the *Conference Proceedings* process):

- **Generic (case independent):** data attributes that can be considered likely to occur within any process (of course, the data values change from case to case). Such data would include descriptors such as created when, created by, times invoked, last invoked, current status; and agent or worker descriptors such as experience, skills, rank, history with this worklet and so on.
- **Case dependent with *a-priori* knowledge:** that set of data that are known to be pertinent to a particular case or instantiation. Generally, this data set reflects the data objects of a particular process instance. Examples are: the dates invitations, papers and reviews sent and received; timeouts both approaching and expired; and actual committee member, reviewer and paper data.
- **Case dependent with no *a-priori* knowledge:** that set of data that only becomes known when the case is active and deviations from the process occur. Examples in this category may include data that describe a missing paper, a request to withdraw a paper or a conference cancellation.

Each worklet is a representation of a particular situated action that relies on the relevant context of each case instance, derived from case data, to determine whether it is invoked to fulfil a task in preference to another worklet in the repertoire. The actual worklet selection process is achieved through the use of modified *Ripple Down Rules* (RDR), which comprise a hierarchical set of rules with associated exceptions, first devised by Compton and Jansen [27].

The fundamental feature of RDR is that it avoids the difficulties inherent in attempting to compile a systematic understanding, organisation and assembly of all knowledge

in a particular domain. The RDR method is well established and fully formalised [28] and has been implemented as the basis for a variety of commercial applications, including systems for reporting DNA test results, environmental testing, intelligent document retrieval, fraud detection based on patterns of behaviour, personal information management and data mining of large and complex data sets [29].

An RDR Knowledge Base is a collection of simple rules of the form "if *condition* then *conclusion*", conceptually arranged in a binary tree structure (fig. 3). Each rule node may have a false ('or') branch and/or a true ('exception') branch to another rule node, except for the root node, which contains a default rule and can have a true branch only. If a rule is satisfied, the true branch is taken and the associated rule is evaluated; if it is not satisfied, the false branch is taken and its rule evaluated [30]. When a terminal node is reached, if its rule is satisfied, then its conclusion is taken; if its rule is not satisfied, then the conclusion of the last rule satisfied on the path to that node is taken. For terminal nodes on a true branch, if its rule is not satisfied then the last rule satisfied will always be that of its parent.

This tree traversal gives RDR implied *locality* - a rule on an exception branch is tested for applicability only if its parent (next-general) rule is also applicable - which allows for general rules to be defined first with refinements added later as the need arises [28].

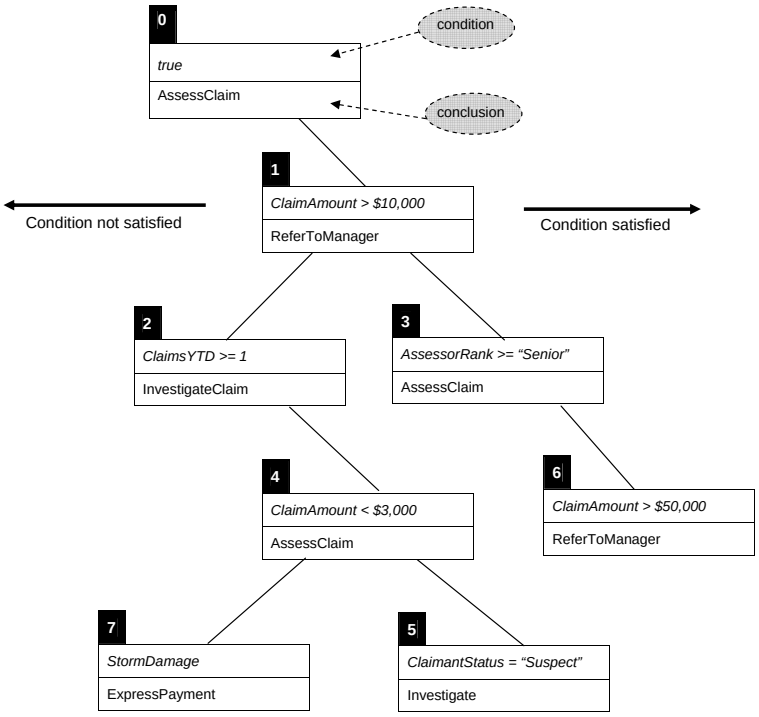

Fig. 3. Conceptual Structure of a Ripple Down Rule (*Assess Claim* Example)

If the conclusion returned is found to be unsuitable for a particular case instance, a new rule is formulated that defines the contextual circumstances of the instance and is added as a new leaf node using the following algorithm:

- If the conclusion returned was that of a satisfied terminal rule, then the new rule is added as a local exception to the exception 'chain' via a new true branch from the terminal node.
- If the conclusion returned was that of a non-terminal, ancestor node (that is, the condition of the terminal rule was not satisfied), then the new rule is added via a new false branch from the unsatisfied terminal node.

In essence, each added exception rule is a refinement of its parent rule. This method of defining new rules allows the construction and maintenance of the KB by "sub-domain" experts (i.e. those who understand and carry out the work they are responsible for) without regard to any engineering or programming assistance or skill [31].

Each node incorporates a set of case descriptors, called the 'cornerstone case', which describe the actual case that was the catalyst for the creation of the rule. The condition for the new rule is determined by comparing the descriptors of the current case to those of the cornerstone case and identifying a sub-set of differences. Not all differences will be relevant – it is only necessary to determine the factor or factors that make it necessary to handle the current case in a different fashion to the cornerstone case to define a new rule. The identified differences are expressed as attribute-value pairs, using the normal conditional operators. The current case descriptors become the cornerstone case for the newly formulated rule; its condition is formed by the identified attribute-values and represents the context of the case instance that caused the addition of the rule.

Rather than impose the need for a closed knowledge base that must be completely constructed *a-priori*, this method allows for the identification of that part of the universe of discourse that differentiates a particular case *as the need arises*. Indeed, the only context of interest is that needed for differentiation, so that processes evolve dynamically through experience gained as they are used.

RDR are well suited to the worklet selection process, since it:

- provides a method for capturing relevant, localised contextual data;
- provides a hierarchical structuring of contextual rules;
- explicitly provides for the definition of exceptions at a local level;
- does not require expert knowledge engineers for its maintenance; and
- allows a rule set to evolve and grow, thus providing support for a dynamic learning system.

A worker defines the contextual conditions as a natural part of the work they perform. This level of human involvement — at the 'coalface', as it occurs — greatly simplifies the capturing of contextual data. Thus RDR allows the construction of an evolving, highly tailored local knowledge base about a business process.

4 The Selection Process

The worklet approach allows for two related but distinct areas of dynamic and flexible workflow to be addressed: dynamic selection of tasks, and exception handling with corrective and compensatory action. The selection process is dealt with in this section; exception handling in the next.

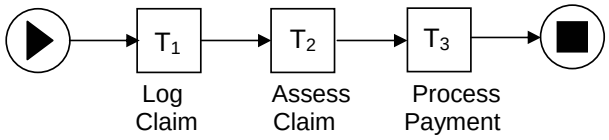

Fig. 4. Simple Insurance Claim Model

Consider the simple insurance claim example in figure 4. When this model is created by an analyst, it is stored as a template of 'placeholders', each linked to a repertoire of worklets from which one will be substituted into the placeholder at runtime. Along with the template, a corresponding set of RDRs is created which define the worklet selection process. That is, each placeholder corresponds to a particular chain of RDRs within which are referenced a repertoire of worklets, one of which will be selected and assigned to the placeholder dynamically.

Initially, the RDR chain for each placeholder will contain one rule: a default *true* condition, and a conclusion referencing the one worklet defined in the model constructed. (It is also possible for the modeller to define several worklets for each placeholder at design time, in which case the one deemed the most generally applicable would be inserted as the conclusion to the default rule). Note that whenever the model is viewed by a stakeholder, each placeholder in the template is filled with a reference to the conclusion of the default rule (i.e. the default worklet) for that placeholder, extracted from the RDRs that define it.

Suppose that, after a while, a new business rule is formulated which states that when the claim comes to be assessed, if the claim amount is more than $10,000 then it must be referred to a manager. In conventional workflow management systems, this would require a re-definition of the model. Using the worklet approach, it simply requires a new worklet to be added to the repertoire and a new rule added as a refinement to the appropriate RDR by the administrator. That is, the new business rule is added as a localised refinement of a more general rule (see fig. 3).

The modified RDR structure can be used to "backwards extract" a view or schematic representation of the model, with the modifed rule for the second placeholder represented as XOR choice (fig. 5 – in YAWL notation, T_1 represents an XOR-split task, T_3 and XOR-join task). Note that the worklet approach enables the model to be displayed as the derived view in figure 5, or as the original representation, thereby offering layers of granularity depending on factors such as frequency of the occurrence of the condition being satisfied. From this it can be seen that a chain of RDRs may be represented in the modelling notation as a composite set of XOR splits and joins. The advantage of using

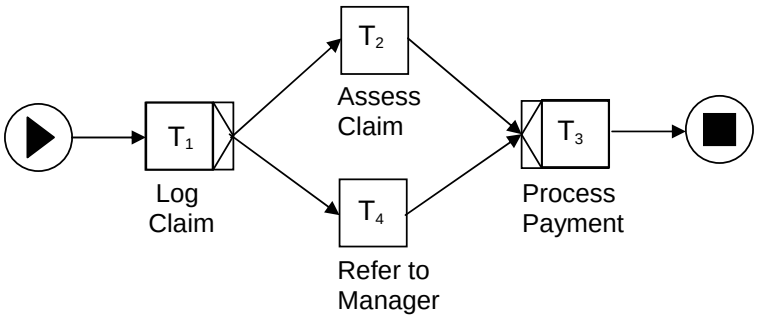

Fig. 5. A 'View' of fig. 4 extrapolated from the modified RDR for placeholder 2

RDRs is that the correct choice is made dynamically and the available choices grow and refine over time, negating the need to explicitly model the choices and repeatedly update the model (with each iteration increasingly camouflaging the original business logic).

It may also be the case that changes in the way activities are performed are identified, not by an administrator or manager, but by a worker who has been allocated a task. Following the example above, after *Log Claim* completes, *Assess Claim* is selected and assigned to a worker's inbox. The worker may decide that the generic *Assess Claim* is not appropriate for this particular case, because this claimant resides in an identified storm-damaged location. Thus, the worker *rejects* the *Assess Claim* worklet (fig. 6). On doing so, the worklet system presents the worker with the set of case data for *Assess Claim* (i.e. its cornerstone case), and the set of current case data.

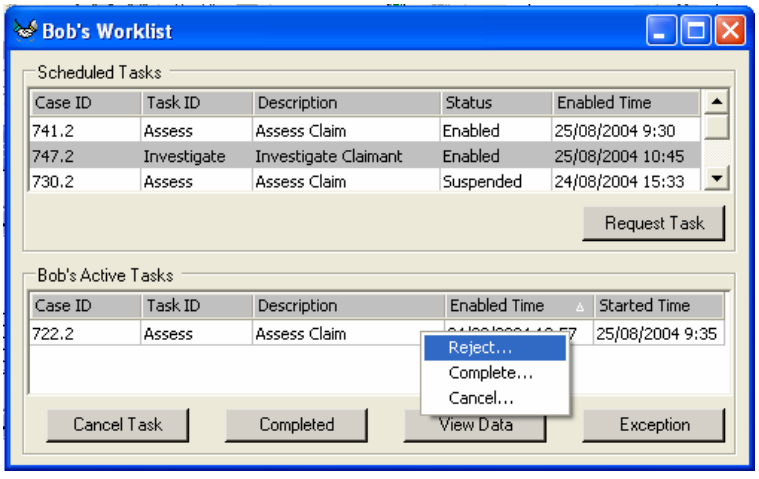

Fig. 6. Rejecting an Offered Worklet (illustration)

The worker then compares the two sets of case data to establish which relevant aspects of the current case differ from *Assess Claim's* cornerstone (fig. 7). Note that while many of the values of the two cases differ, only those that relate directly to the need to handle this case differently are selected. After identifying the differences, the worker is presented with a list of possible worklet choices that may suit this particular case, if available. The worker may choose an appropriate worklet to invoke in this case, or, if none suit the current case, refer to an analyst to define a new worklet for the current case. In either case, the identified differences form the conditional part of a new rule, which is added to the RDR for this placeholder using the rule addition algorithm above.

The principles derived from Activity Theory state that all work activities are mediated by rules, tools and division of labour. Translating that to an organisational work environment, rules refer to business rules, policies and practices; tools to resources and their limitations (physical and financial); and division of labour to organisational hierarchies, structures, roles, lines of supervision, etc. Of course, these constraints apply to the creation of a new worklet, just as they would in any workflow management system.

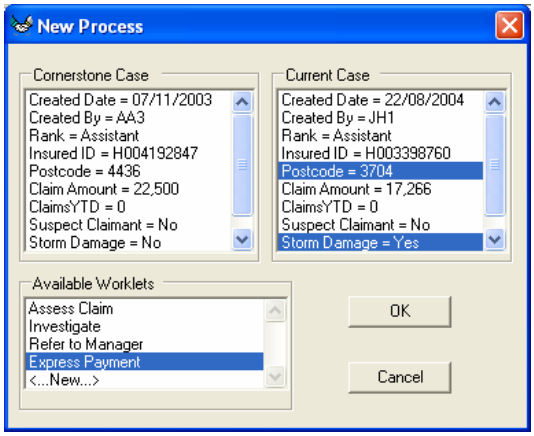

Fig. 7. Defining a New Rule (illustration)

In all future case instantiations, the new worklet defined above would be chosen for that placeholder if the condition defined by choosing the attribute differences occur in that instantiation's case data. Over time, the RDR chain for the placeholder grows as refinements are added to the rule base (fig. 3).

5 Exception Handling

Worklets may also be defined and used to provide exception handling capabilities for events that occur during the execution of a case instance. When such an event occurs, a corresponding global system event is triggered that passes an appropriate message to all pending, running or suspended (i.e. live) worklets. Each worklet has a second RDR

rule base for exceptions, separate from the normal worklet selection rule base, which is interrogated when a message is received. If the default condition in an RDR chain is an identifier for the message received, then the worklet will handle that exception by invoking an appropriately selected exception handling worklet. If there is no RDR for that exception, it is simply ignored.

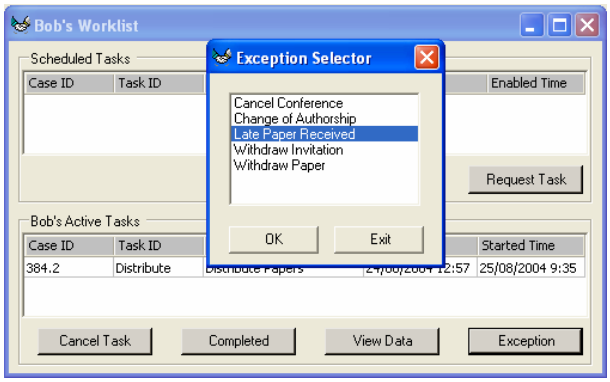

Fig. 8. Selecting an External Exception Trigger (illustration)

Exceptions may be triggered by a combination of (a) system generated messages (e.g. deadline reached, state-change of another worklet, etc); (b) domain dependent data (e.g. count thresholds, missing data etc.); and (c) external triggers (i.e. user interactions – see fig. 8). Exception handling begins at the 'lowest level child' — that is, those child worklets that are not also a parent — and then, when handling is complete, control is passed up hierarchically to each parent in turn until all 'interested' worklets have handled the exception. This method ensures that all exception handling tasks are defined and performed locally and in a distributed manner.

A worklet that has been invoked as an exception handler has the ability to modify the current process state of its invoker, or parent worklet, when it is activated and again when it completes. By default, the invoking worklet is suspended when the exception handler is activated, but there may be some occasions when the exception handler can (or needs to) operate in parallel to the invoker. For example, it is undesirable to suspend an entire *Conference Proceedings* process when a request to withdraw a single paper has been received – it is far easier to handle the request as a (local) exception and allow the parent process to continue. When an exception handling worklet completes, it may unsuspend the invoker (by default), leave the invoker's state unchanged (e.g. if it was not suspended to begin with), or cancel the invoker worklet (e.g. a 'withdraw paper' exception will cancel a parent 'review paper' worklet).

As an example of the exception handling process, consider the *Distribute Paper* worklet (fig. 9), which is invoked by the top level *Coordinate Conference* worklet, and in turn invokes a *Send Paper for Review* worklet, once for each reviewer selected to review a particular paper. If, after a deadline is reached, the number of reviews returned

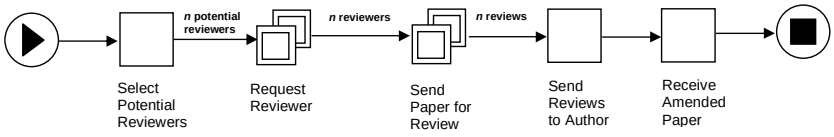

Fig. 9. Worklet: Distribute Paper. Invoked by *Coordinate Conference*

fails to meet a pre-defined threshold, an *Insufficient Reviews* exception is raised. The exception is handled by a rule chain in the Exception KB (fig. 10) that determines which worklet should be invoked to handle the situation.

By default, the worklet *Review Paper* is called which organises a new reviewer for the paper. However, if two reviews have already been received, then the paper is sent to a Committee member for review, unless the ratings from those two reviews are high, in which case a brief review is carried out. If all the above is true, and one of the reviewers is a Committee member, then no further reviewing is required (*NullWorklet*). If, however, the rating was not high and one of the reviewers is a Committee member, then a brief review is also carried out in this case.

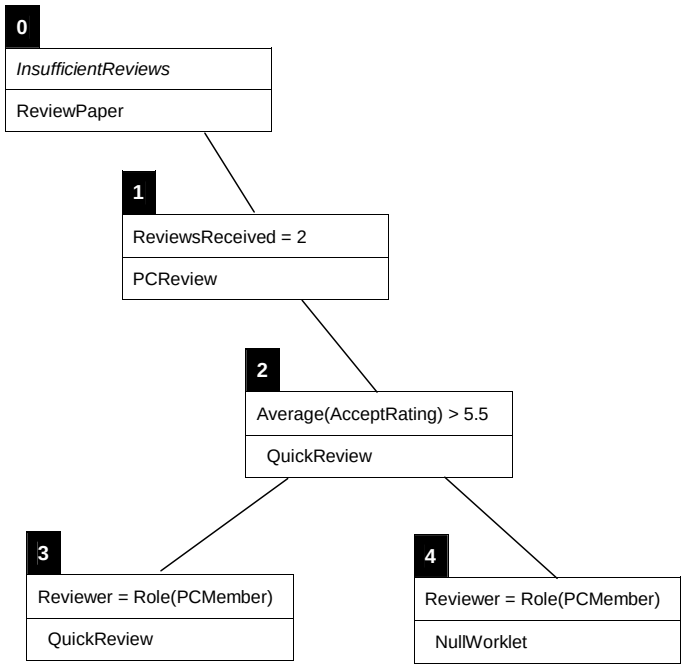

Fig. 10. RDR for exception 'InsufficientReviews'

Of course, this example is relatively simple; conditions may also be based on data such as: time left before deadline, difficulty of paper, areas of expertise, availability of extra reviewers, etc. Note that to build all of these conditions and related actions into a process model in the traditional fashion would be quite a complex procedure. To reiterate, there are two situations where selection of an appropriate worklet is to be made:

- Normal execution - when control flows from one task to the next. In this case, selection occurs immediately prior to the placeholder becoming enabled and so does not depend on the current process state for instantiation. Thus, in every case worklet selection for a particular placeholder will be based on domain specific data values (for example: how many, who by, when, etc.).
- Exception handling - choosing the most appropriate worklet to handle an exception will be achieved by defining RDR conditions that use a combination of current data attribute values (both domain dependent and independent) and the current state of each of the worklets that make up the case instance.

The current set of states for a worklet-enabled instantiated process may be deduced by mining the process log file [32]. A set of predicates can be formalised to enable the extraction of the current state set and any relations between active worklets. These predicates may then be used as conditionals in RDR nodes to enable selection of the appropriate exception handling worklet.

A full exploration of the formalisation is beyond the scope of this paper, however the kinds of information that may be extracted from the process log file using these predicates include the current status of a worklet, whether a worklet is a parent of child of another worklet, when a certain state was entered or exited for a particular worklet, the resource that triggered a state change, and so on.

An example of a user-invoked exception handler is *Cancel Conference*, which may occur at any time during a *Conference Proceedings* instantiation. To define an entire process cancellation in a non-worklet environment would be extremely complex, if not impossible. Using the worklet approach, such complexities are removed. The exception definition remains relatively simple, even though it halts the entire process and has myriad tasks to complete depending on the current state of the entire process. A *Cancel Conference* message is 'broadcast' to allow each currently active worklet to determine the means of its own termination by querying its current state and the states of related worklets, thereby localising the exception handling process and simplifying both the parent exception schema and the schematic of the overall process.

After all lower level worklets have completed handling the exception, the top-level manager worklet invokes its own exception handler. Since each worklet takes care of its own termination according to its circumstances, the manager worklet needs only to notify various stakeholders before finally terminating, using the process log predicates to determine the current process state.

6 Related Work

Generally, commercial workflow management systems provide only basic support for handling exceptions [11, 33, 34] (besides modelling them directly in the main 'business

logic'), and each deals with them in a proprietary manner. Staffware provides constructs called *event nodes*, from which a separate exception handling path or sequence can be activated when an exception occurs. It may also suspend a process either indefinitely or wait until a timeout occurs. If a work item cannot be processed it is forwarded to a 'default exception queue' where it may be manually purged or re-submitted. COSA provides for the definition of external 'triggers' or events that may be used to start a sub-process. All events and sub-processes must be defined at design time. MQ Workflow supports timeouts and, when they occur, will branch to a pre-defined exception path and/or send a message to an administrator. SAP R/3 provides for pre-defined branches which, when an exception occurs, allows an administrator to manually choose one of a set of possible branches.

All the commercial products reviewed provide modelling frameworks that are basically monolithic, but with various levels of support for the decomposition of tasks and sub-processing. Each of the products require the model to be fully defined before it can be instantiated, and changes must be incorporated by modifying the model statically. Staffware provides 're-usable process segments' that can be inserted into any process. SAP R/3 allows for the definition of 'blocks' that can be inserted into other 'blocks', thus providing some support for encapsulation and reuse. COSA supports parent-sibling processes, where data can be passed to/from a process to a sub-process. MQ Workflow allows sub-processes to be defined and called statically from within a process.

A new, optional component of Staffware is the *Process Orchestrator* [35], which provides for the dynamic allocation of sub-processes at runtime. It requires a construct called a "dynamic event" to be explicitly modelled that will execute a number of sub-processes listed in an 'array' when execution reaches that event. Which sub-processes execute depend on predefined data conditionals matching the current case. The listed sub-processes are statically defined, as are the conditionals. There is no scope for dynamically refining conditionals, nor adding sub-processes at runtime.

The *OPERA* prototype [36] allows for exceptions to be handled at the task level, or propagated up various ancestor levels throughout the running instance. It also removes the need to define the exception handler *a-priori*, although the types of exceptions handled are transactional rather than control flow oriented. The *eFlow* system [37] uses rules to define exceptions, although they cannot be defined separately to the standard model. *ADEPT* [38] supports modification of a process during execution (i.e. add, delete and change the sequence of tasks). Such changes are made to a traditional monolithic model and must be achieved via manual intervention. The *ADOME* system [39] provides templates that can be used to build a workflow model, and provides some support for (manual) dynamic change. A catalog of 'skeleton' patterns that can be instantiated or specialised at design time is supported by the *WERDE* system [2]. Again, there is no scope for specialisation changes to be made at runtime. It should be noted that only a small number of academic prototypes have had any impact on the frameworks offered by commercial systems [40].

Note: The information on commercial products was gleaned from their relevant manuals and other literature (unless otherwise stated). The versions examined were Staffware Process Suite v9 (2003), MQ Workflow version 3.4 (2003), Cosa Workflow version 4.2 (2003) and SAP R/3 Release 6.20 (2004).

7 Conclusion and Future Work

Workflow management systems impose a certain rigidity on process definition and enactment because they use proprietary frameworks based on assembly line metaphors rather than on ways work is actually planned and carried out. An analysis of Activity Theory provides principles of work practices that can be used as a template on which a workflow management system can be built that better supports flexibility and dynamic evolution. By capturing contextual data, a repertoire of actions may be developed that allow for contextual choices to be made from the repertoire at runtime to efficiently carry out work tasks. These actions, or worklets, directly provide for process evolution, flexibility and exception handling, and mirror accepted work practices.

The worklet approach presents the promise of several key benefits, including:

- A process modeller can describe the standard activities and actions for a workflow process, and any exceptional activities, using the same methodology;
- It allows re-use of existing process and exception handling components. Removing the differentiation between exception handling processes and the 'normal' workflow aids in the development of fault tolerant workflows out of pre-existing building blocks [36];
- Its modularity simplifies the logic and verification of the standard model, since individual worklets are less complex to build and therefore verify than monolithic models;
- It provides for workflow views of differing granularity, which offers ease of comprehensibility for all stakeholders;
- It allows for gradual and ongoing evolution of the model, so that global modification each time a business practice changes or an exception occurs is unnecessary; and
- In the occurrence of an unexpected event, the modeller needs simply to choose an existing handler or build a new one for that exception, which can be automatically added to the repertoire for future use as necessary, thus avoiding complexities including downtime, model restructuring, versioning problems and so on.

A future prospect of this work involves the formalisation and implementation of the worklet paradigm, integrating it with a workflow 'engine' within which it can execute. Since it is extendible and Open Source, an engine that is particularly well-suited to such an integration is the YAWL (*Yet Another Workflow Language*) workflow engine, currently being developed at the Queensland University of Technology.

YAWL is an ideal language for the specification of control flow in workflows; it is highly expressive and provides direct support for 19 out of 20 identified workflow patterns [41]. In addition, YAWL has formal semantics and offers graphical representations for workflow models (including worklets and the examples in this paper).

References

1. Poitr Chrząstowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O'Dell, and Adi Susanto. A top-down petri net-based approach for dynamic workflow modeling. In W.M.P van der Aalst et al., editor, *International Conference on Business Process Management (BPM '03)*, volume 2678 of *LNCS*, pages 336–353, Eindhoven, The Netherlands, June 2003.

2. Fabio Casati. A discussion on approaches to handling exceptions in workflows. In *1998 Conference on Computer-Supported Cooperative Work*, Seattle, USA, 1998.
3. H.A. Reijers, J.H.M. Rigter, and W.M.P. van der Aalst. The case handling case. *International Journal of Cooperative Information Systems*, 12(3):365–391, 2003.
4. Alan Rickayzen, Jocelyn Dart, Carsten Brennecke, and Marcus Schneider. *Practical Workflow for SAP*. Galileo Press, Bonn, 2002.
5. Thomas Schael. *Workflow management systems for process organisations*, volume 1096 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2nd edition, 1998.
6. W.M.P. van der Aalst, Mathias Weske, and Dolf Grünbauer. Case handling: A new paradigm for business process support. *Data & Knowledge Engineering*, (to appear), 2004.
7. Gregor Joeris. Defining flexible workflow execution behaviors. In Peter Dadam and Manfred Reichert, editors, *Enterprise-wide and Cross-enterprise Workflow Management - Concepts, Systems, Applications*. University of Ulm, Ulm, 1999.
8. Alex Borgida and Takahiro Murata. Tolerating exceptions in workflows: a unified framework for data and processes. In *International Joint Conference on Work Activities, Coordination and Collaboration (WACC'99)*, pages 59–68, San Francisco, CA, 1999. ACM Press.
9. Peter J. Kammer, Gregory Alan Bolcer, Richard N. Taylor, and Mark Bergman. Techniques for supporting dynamic and adaptive workflow. Technical report, Department of Information and Computer Science, University of California, Irvine, December 1998.
10. Jakob E. Bardram. I love the system - I just don't use it! In *Proceedings of the 1997 International Conference on Supporting Group Work (GROUP'97)*, Phoenix, Arizona, 1997.
11. Claus Hagen and Gustavo Alonso. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering*, 26(10):943–958, October 2000.
12. Mark S. Ackerman and Christine Halverson. Considering an organization's memory. In *Proceedings of the ACM 1998 Conference on Computer Supported Cooperative Work*, pages 39–48. ACM Press, 1998.
13. Peter A. K. Larkin and Edward Gould. Activity theory applied to the corporate memory loss problem. In L. Svennson, U. Snis, C. Sorensen, H. Fagerlind, T. Lindroth, M. Magnusson, and C. Ostlund, editors, *Proceedings of IRIS 23 Laboratorium for Interaction Technology*, University of Trollhattan Uddevalla, 2000.
14. W.M.P. van der Aalst. Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3):297–317, 2001.
15. S. Rinderle, M. Reichert, and P. Dadam. Evaluation of correctness criteria for dynamic workflow changes. In W.M.P van der Aalst et al., editor, *International Conference on Business Process Management (BPM '03)*, LNCS, pages 41–57, Eindhoven, The Netherlands, June 2003.
16. W.M.P. van der Aalst and P.J.S. Berens. Beyond workflow management: Product-driven case handling. In S. Ellis, T. Rodden, and I. Zigurs, editors, *International ACM SIGGROUP Conference on Supporting Group Work (GROUP 2001)*, pages 42–51, New York, 2001. ACM Press.
17. Diane M. Strong and Steven M. Miller. Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems*, 13(2):206–233, 1995.
18. P. Barthelmess and J. Wainer. Workflow systems: a few definitions and a few suggestions. In *ACM Conference on Organizational Computing Systems (COOCS'95)*, pages 138–147, San Jose, California, 1995.
19. Jakob E. Bardram. Plans as situated action: an Activity Theory approach to workflow systems. In *Proceedings of the 1997 European Conference on Computer Supported Cooperative Work (ECSCW'97)*, pages 17–32, Lancaster U.K., 1997.
20. Bonnie A. Nardi. *Activity Theory and Human-Computer Interaction*, pages 7–16. In Nardi [22], 1996.

21. Y. Engestrom. *Learning by Expanding: An Activity-Theoretical Approach to Developmental Research*. Orienta-Konsultit, Helsinki, 1987.
22. Bonnie A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, Cambridge Massachusetts, 1996.
23. Michael Adams, David Edmond, and Arthur H.M. ter Hofstede. The application of activity theory to dynamic workflow adaptation issues. In *Proceedings of the 2003 Pacific Asia Conference on Information Systems (PACIS 2003)*, pages 1836–1852, Adelaide, Australia, July 2003.
24. W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede. Design and implementation of the YAWL system. In A. Persson and J. Stirna, editors, *Proceedings of The 16th International Conference on Advanced Information Systems Engineering (CAiSE 04)*, volume 3084 of *lncs*, pages 142–159, Riga, Latvia, June 2004. Springer Verlag.
25. Paolo Bouquet, Chiara Ghidini, Fausto Giunchiglia, and Enrico Blanzieri. Theories and uses of context in knowledge representation and reasoning. *Journal of Pragmatics*, 35(3), 2003.
26. Debbie Richards. Combining cases and rules to provide contextualised knowledge based systems. In *CONTEXT 2001, Lecture Notes in Artifical Intelligence*, volume 2116, pages 465–469. Springer-Verlag Berlin, 2001.
27. P. Compton and B. Jansen. Knowledge in context: A strategy for expert system maintenance. In J.Siekmann, editor, *Lecture Notes in Artificial Intelligence*, volume 406, pages 292–306. Springer, 1988.
28. Tobias Scheffer. Algebraic foundation and improved methods of induction of ripple down rules. In *Procceedings Pacific Rim Workshop on Knowledge Acquisition*, Sydney, Australia, 1996.
29. Pacific Knowledge Systems. Products: Rippledown, `http://www.pks.com.au/products/validator.htm`, September 2003.
30. B. Drake and G. Beydoun. Predicate logic-based incremental knowledge acquisition. In P. Compton, A. Hoffmann, H. Motoda, and T. Yamaguchi, editors, *Proceedings of the sixth Pacific International Knowledge Acquisition Workshop*, pages 71–88, Sydney, December 2000.
31. Byeong Ho Kang, Phil Preston, and Paul Compton. Simulated expert evaluation of multiple classification ripple down rules. In *Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling and Management*, Banff, Alberta, Canada, April 1998.
32. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
33. Fabio Casati and Giuseppe Pozzi. Modelling exceptional behaviours in commercial workflow management systems. In *1999 IFCIS International Conference on Cooperative Information Systems*, pages 127–138, Edinburgh, Scotland, 1999.
34. W.M.P. van der Aalst and T. Basten. Inheritance of workflows: An approach to tackling problems related to change. In *Theoretical Computer Science*, volume 270(1-2), pages 125–203, 2002.
35. Michael Georgeff and Jon Pyke. Dynamic process orchestration. White paper, Staffware PLC `http://tmitwww.tm.tue.nl/bpm2003/download/WP%20Dynamic%20%Process%20Orchestration%20v1.pdf`, March 2003.
36. Claus Hagen and Gustavo Alonso. Flexible exception handling in process support systems. Technical report no. 290, ETH Zurich, 1998.
37. Fabio Casati, Ski Ilnicki, LiJie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. Adaptive and dynamic composition in eFlow. In *12th International Conference, CAiSE 2000*, pages 13–31, Stockholm, Sweden, 2000.

38. Clemens Hensinger, Manfred Reichert, Thomas Bauer, Thomas Strzeletz, and Peter Dadam. ADEPT$_{workflow}$ - advanced workflow technology for the efficient support of adaptive, enterprise-wide processes. In *Conference on Extending Database Technology*, Konstanz, Germany, March 2000.
39. Dickson Chiu, Qing Li, and Kamalakar Karlapalem. A logical framework for exception handling in ADOME workflow management system. In *12th International Conference CAiSE 2000*, pages 110–125, Stockholm, Sweden, 2000.
40. Michael zur Muehlen. *Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems.*, volume 6 of *Advances in Information Systems and Management Science*. Logos, Berlin, 2004.
41. 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, July 2003.