DYNAMIC AND CONTEXT-AWARE PROCESS ADAPTATION

MICHAEL ADAMS, ARTHUR TER HOFSTEDE, NICK RUSSELL AND WIL VAN DER AALST

Abstract. This Chapter re-examines the principles that underpin business process technologies to derive a novel approach that moves beyond the traditional assembly-line metaphor. Using a set of principles derived from *Activity Theory*, a system has been implemented, using a Service Oriented Architecture, that provides support for dynamic and extensible flexibility, evolution and exception handling in business processes, based on accepted ideas of how people actually perform their work tasks. The resulting system, called the *Worklet Service*, makes available all of the benefits offered by Process Aware Information Systems to a wider range of organisational environments.

1. Introduction

Organisations are constantly seeking efficiency improvements for their business processes. To help achieve those goals, many are turning to Process-Aware Information Systems (PAIS) to configure and control those processes (Dumas et al., 2005; van der Aalst & van Hee, 2004) by supporting their modelling, analysis, enactment and management (zur Muehlen, 2004; Casati, 1998). The key benefits organisations seek by implementing PAIS solutions include: improved efficiency, better process control, improved customer service and business process improvement.

The use of PAIS has grown by concentrating on modelling rigidly structured business processes that in turn derive well-defined workflow instances (Bider, 2005; Joeris, 1999; Reichert & Dadam, 1997). However, the proprietary process definition frameworks imposed make it difficult to support (i) dynamic evolution and adaptation (i.e. modifying process definitions during execution) following unexpected or developmental change in the business processes being modelled (Borgida & Murata, 1999); and (ii) deviations from the prescribed process model at runtime (Rinderle et al., 2004; Casati, 1998).

But change is unavoidable in the modern workplace. To remain effective and competitive, organisations must continually adapt their business processes to manage the rapid changes demanded by the dynamic nature of the marketplace or service environment. Also, a large

proportion of workplaces undertake activities that do not easily conform to rigid or constricting representations of their work practices. And even in the most concrete processes deviations will occur within almost every instantiation.

If PAIS could be extended to meet the challenges of evolutionary and unexpected change in business processes, then their applicability would widen to include a far greater proportion of workplaces. Such support would not only benefit existing users of process-aware technologies, but would also introduce those businesses which employ more creative or ad-hoc processes to the range of benefits that PAIS offer.

This Chapter offers one solution designed to meet that challenge. The primary objectives of this Chapter are to provide:

- an overview of literature on approaches to exception handling and flexibility in Process-Aware Information Systems
- a discussion of theoretical underpinnings of work practices
- a discussion of a comprehensive framework for exception handling
- the introduction of a concrete approach for exception handling based on this framework
- the introduction of a concrete approach to processes that require on-the-fly change
- a description of an (open source) implementation of these approaches within a state-of-the-art workflow system; and
- the presentation of an elaborated example.

2. Background

A business process can be defined as a composite set of tasks that comprise coordinated computer-based and human activities (Leymann, 2006). A business process model or schema is a formal representation of work procedures that controls the sequence of performed tasks and the allocation of resources to them (Oberweis, 2005).

The development of a business process model typically begins with an analysis of current business procedures and processes. Subsequently, a model is developed based on those practices and business rules, then input into a PAIS and repetitively executed, supporting and giving formal structure and flow control to those processes. However, translating abstract concepts and descriptions of business practices and rules into tangible process models is a far from trivial exercise. There are sizeable development costs involved in mapping an abstract process to a structured schema, which must be weighed against the perceived cost benefits that the system will deliver. Therefore, current systems are most advantageous where they provide support for standardised, repetitive activities that do not vary between execution instances.

But even in highly structured environments, it is difficult (if not impossible) to successfully capture all work activities, and in particular all of the task sequences possible, in a static process model. It is also the case that, for any given human activity, the process for successfully completing the activity is constantly evolving. Change can also be introduced via many sources, including government regulation, new competitors, new markets, improvements in plant and equipment, workforce and resource availability and so on.

A recent Workflow Management Coalition survey found that 75 per cent of respondents reported they were currently performing work on improving existing processes (up to 92 per cent for the Finance sector) and 56 per cent were currently involved in a major business process redesign (Palmer, 2007). Such statistics underscore the frequency of organisational change and importance of providing a process management system which supports flexibility and the ability to adapt to change.

It is because of the discrepancies between real-world activities and formal representations of them that process instances typically experience *exceptions* during their execution. Rather than being considered an error, an exception in a business process is simply a deviation from the expected control flow or was unaccounted for in the original process model. Exceptions are a fundamental part of most organisational processes (Kammer et al., 2000); in fact, a substantial proportion of the everyday tasks carried out in a business can be categorised as exception handling work (Barthelmess & Wainer, 1995). Historically, exception handling within PAIS has fallen well short, particularly after execution has commenced (Kammer et al., 2000).

Thus a large group of business processes do not easily map to the rigid modelling structures provided, due to the lack of flexibility inherent in a framework that, by definition, imposes rigidity. This inflexibility extends to the management of exceptions, which places further limits on how accurately a process model can reflect the actual business process it is based on. Rather, process models are 'system-centric', meaning that work processes are *straight-jacketed* (van der Aalst et al., 2005) into the paradigm supplied, rather than the paradigm reflecting the way work is actually performed (Bider, 2005). As a result, users are forced to work outside of the system, and/or constantly revise the process model, in order to successfully complete their activities, thereby negating the perceived efficiency gains sought by implementing a process management solution in the first place.

Survey of Literature and Related Systems. The following reviews the levels of support for flexibility and exception handling in several of the leading commercial process management products and a number of

academic prototypes[1]. Unless explicitly stated otherwise, information regarding the products has been gleaned from product manuals, published literature and white papers. The version numbers specified for the commercial products are the versions that were reviewed.

Since the mid-nineties much research has been carried out on issues related to dynamic flexibility and exception handling in workflow management systems. Such research was initiated because, generally, commercial workflow management systems provide only basic support for handling exceptions (Russell et al., 2006; zur Muehlen, 2004) (besides modelling them directly in the main 'business logic'), and, where extant, each deals with them in a proprietary manner; they typically require the model to be fully defined before it can be instantiated; and changes must be incorporated by modifying the model statically. Further, there is minimal support for handling: workitem failures (and even when that support is offered, they must be manually terminated); external triggers; and only one system reviewed offers some constraint violation management (Russell et al., 2006).

Tibco iProcess Suite (version 10.5) (formerly *Staffware*) (TIBCO, 2006) provides constructs called *event nodes*, from which a separate pre-defined exception handling path or sequence can be activated when an exception occurs at that point. It may also suspend a process either indefinitely or wait until a deadline occurs. If a workitem cannot be processed it is forwarded to a 'default exception queue' where it may be manually purged or re-submitted. A compensation workitem may be initiated when a deadline occurs. Also, a workitem may be externally triggered, or 'wait' until an external trigger occurs. Certain tasks may be manually skipped at runtime.

An optional component of the iProcess Suite is the *Process Orchestrator* (Georgeff & Pyke, 2003), 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 a predefined 'array' when execution reaches that event. Which sub-processes execute depend on predefined data conditionals matching the current case. There is no scope for dynamically refining conditionals, nor adding sub-processes at runtime.

COSA (version 5.4) (COSA, 2005) 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, although models can be modified at runtime (but only for future instantiations). When a workitem fails the activity can be rolled back or restarted. A compensating activity can be triggered either externally or on deadline expiry. *COSA* also allows *manual* ad-hoc runtime adaptations such as reordering, skipping, repeating, postponing or terminating steps.

[1]Space considerations limit this discussion to the more popular and/or recent systems and prototypes; a more complete discussion can be found in Adams (2007)

WebSphere MQ Workflow (version 6.0) (IBM, 2005) supports deadlines and, when they occur, will branch to a pre-defined exception path and/or send a notification message to a pre-defined user or administrator. Administrators can manually suspend, restart or terminate processes, or reallocate tasks. Only transaction-level exceptions are recognised, and they are simply recorded in the audit log.

SAP Workflow (version 6.20) (SAP, 2006) supports conditional branching, where a list of conditions (each linked to a process branch) is parsed and the first evaluating to true is taken; all branches are pre-defined. Exception events are provided for cancelling workflow instances, for checking workitem pre and post constraints, and for 'waiting' until an external trigger occurs. Exception handling processes may be assigned to a workflow based on the type of exception that has occurred, although the handlers for each are specified at design time, and only one may be assigned to each type — that is, filtering through a set of possible handlers based on the context of the case is not supported. When an exception occurs and a corresponding handler is found, all tasks in the block where the exception is caught are cancelled.

FLOWer (version 2.1) (Berens, 2005), is of the 'case-handling' paradigm); the process model (or 'plan') describes only the preferred way of doing things and a variety of mechanisms are offered to allow users to deviate in a controlled manner (van der Aalst et al., 2005). For example, a deadline expiry can automatically complete a workitem. Also, some support for constraint violation is offered: a plan may be automatically created or completed when a specified condition evaluates to true (Russell et al., 2006).

There have been a number of academic prototypes developed in the last decade (although activity was greater during the first half); very few have had any impact on the offerings of commercial systems (zur Muehlen, 2004). Several of the more widely acknowledged are discussed here.

ADEPT (Reichert et al., 2005) supports modification of a process during execution (i.e. add, delete and change the sequence of tasks) both at the model (dynamic evolution) and instance levels (ad-hoc changes). Such changes are made to a traditional monolithic model and must be achieved via manual intervention, abstracted to a high level interaction. The system also supports forward and backward 'jumps' through a process instance, but only by authorised staff who instigate the skips manually (Reichert et al., 2003).

The *AdaptFlow* prototype (Greiner et al., 2004) provides a hybrid approach to flexibility and exception handling. It supports ECA rules-based detection of exceptions and the dynamic adaptation of process instances, although each adaptation must be confirmed manually by an authorised user before it is applied (alternate manual handling to override the dynamic adaptation offered is also supported). Also, the

rule classifications and available exception handling actions are limited to medical treatment scenarios.

AgentWork (Muller et al., 2004) provides the ability to modify process instances by dropping and adding individual tasks based on events and ECA rules. However, changes are limited to individual tasks, rather than the task-process-specification hierarchy. Also, the possibility exists for conflicting rules to generate incompatible actions, which requires manual intervention and resolution.

A further approach using incompletely specified process definitions is found in the *SwinDeW (Swinburne Decentralised Workflow)* project (Yan et al., 2004). SwinDew is a peer-to-peer based decentralised model, where a process definition is split into a set of task partitions and distributed to peers, and on-the-fly process elaboration is performed at runtime. Thus, a multi-tiered process modelling and execution framework is provided.

CBRFlow (Weber et al., 2004) uses a case-based reasoning approach to support adaptation of predefined workflow models to changing circumstances by allowing (manual) annotation of business rules during run-time via incremental evaluation by the user. Thus users must be actively involved in the inference process during each case.

An approach, which integrates *CBRFlow* into the *ADEPT* framework, is described in (Rinderle et al., 2005). In doing so, semantic information about the reasons for change, and traceability data, are presented to the *ADEPT* user/administrator to support decision making processes. The information can also be used to facilitate reuse of ad-hoc changes from similar scenarios. When deviations from a process schema are required, the case-based reasoning component assists the user to find similar previous cases through a series of questions and answers, one of which may then be applied to the current instance. While the process is quite user-intensive, the approach does provide a good example of the combination of contextual information with exception handling techniques.

Pesic and van der Aalst (2006) point out that the majority of languages used to described and define business process models are of a procedural nature, which limits their effectiveness in very flexible environments, and introduce a declarative approach to process modelling, called *DecSerFlow*. A graphical language, it avoids many of the assumptions, constraints, conditions and rules that must be explicitly specified in procedural languages to perform flexible activities, the inclusion of which typically lead to an over-specification of the process.

In summary, approaches to flexibility and exception handling usually rely on a high-level of runtime user and/or administrator interactivity, which directly impedes on the basic aim of PAIS (to bring greater efficiencies to work practices) and distracts users from their primary work roles into process support activities. Another common limitation

is the complex update, modification and migration issues required to evolve process models.

3. Theoretical Foundation

Whenever a series of actions is undertaken with a view to achieving a pre-conceived result, some plan or set of principles is implemented that guide and shape those actions towards that goal. To be effective, a plan must be described using constructs and language that are relevant to both the actions being performed and the desired result, and be comprehensible by its participants and stakeholders. In business process terms, analysts seek to model some aspect of the real world by using a metaphor that bears some resemblance to the real world, but also represents an understanding of computational processes. Such metaphors are abstract constructs that form a common reference model that assist in representing the external world through computers.

The fundamental and widely understood *computational* metaphor (Stein, 1999) takes a set of inputs, performs a series of functional steps in a strict sequence, and, on completion, produces some output that represents the goal of the process. This metaphor describes a single, centralised thread of control, which very much reflects its mathematical ancestry but also reveals an underlying misconception in the common perception of technological 'progress'. Technological developments are, according to Holt (1997), "as much affected by fashion as clothing". Technologies do not evolve automatically (as Marx assumed) but rather reflect prevailing human culture (Mumford, 1963). That is, new technologies are derived from perceived needs and realised, not in isolation, but through the conventions and norms of their social *milieu*.

Thus the traditional computational metaphor reveals the influence of pioneers such as von Neumann and his team, and especially Turing, whose abstract machine proposed 'step-at-a-time' processing (Turing, 1936), and which in turn reflects the influence on prevailing thought of the contemporaneous development of assembly-line manufacturing (Hendriks-Jansen, 1996).

As contempory technological advances influenced the structure of early computers, so too has the computational metaphor become a significant model system for the conceptualisation and interpretation of complex phenomena, from cognition to economics to ecology (Stein, 1999). Of particular interest is the way the metaphor has been applied to the definition of organisational work processes. The computational metaphor remains applicable to well-defined problem domains where goal-directed, sequential, endpoint-driven planning is required (Stein, 1999). Such domains were the early beneficiaries of process management systems. Consequently, current commercial process management systems provide support for standardised, repetitive activities that do not vary between execution instances.

Adherence to the metaphor by PAIS has been an important factor in their acceptance by organisations with structured work practices. Descriptions can be found throughout the workflow literature to the 'processing', 'manufacturing' and 'assembly-line' modelling metaphors that are employed by commercial systems. However, while the Workflow Management Coalition claims that "even office procedures can be processed in an assembly line" (Workflow Management Coalition, 2002), there are many aspects where administrative and service processes differ from manufacturing processes (van der Aalst & Berens, 2001). It may be that the computational metaphor has been an inhibiting factor in the development of systems able to effectively support flexible work practices.

A process management system that better supports flexible work environments requires a sound theoretical foundation that avoids the computing metaphor, but rather describes how work is actually conceived, carried out and reflected upon. One such theoretical base can be found in *Activity Theory*.

Activity Theory is a powerful and clarifying descriptive tool, rather than a strongly predictive theory, and incorporates notions of intentionality, history, mediation, collaboration and development, focussing on understanding everyday practice in the real world (Nardi, 1996).

Activity Theory originated in the former Soviet Union in the 1920's and 30's as part of the cultural-historical school of psychology founded by Vygotsky, who began working on the theory at a time when the prevailing psychological theories were based on reflexology (which attempted to reduce all psychological phenomena to a series of stimulus-response chains).

In the form presented by Leontiev (1974), Activity Theory subsequently became one of the major Soviet psychology theories, and was used in areas such as the education of disabled children and the ergonomic design of equipment control panels.

In the 1980's and 90's, Activity Theory came to the attention of Scandinavian information technology researchers (for example: (Nardi, 1996), (Kuutti, 1996) and (Bødker & Greenbaum, 1993). Their contribution was a revised formulation of Activity Theory, and also the application of Activity Theory to human-computer interface design.

Briefly, Activity Theory states that human activity has four basic characteristics (Bardram, 1997):

(1) Every activity is directed towards a material or ideal object satisfying a need, which forms the overall motive of the activity.
(2) Every activity is mediated by artefacts, either external (a tool) or internal (cognitive: using concepts, knowledge and experience).
(3) Each individual activity is almost always part of collective activities, structured according to the work practice in which they

take place. For example, a patient diagnosis can seldom be established without reference to a diversity of medical information. Thus collective activities are organised according to a division of labour.

(4) Finally, human activity can be described as a hierarchy with three levels: *activities* realised through chains of *actions*, and performed through *operations*:
   - An *activity* consists of one or more actions, and describes the overall objective or goal.
   - An *action* equates to a single task carried out to achieve some pre-conceived result. Each action is achieved through operations determined by the actual conditions in the context of the activity.
   - *Operations* describe the actual performance of the action, and are dependant on the context, or conditions that exist for each action.

In Adams et al. (2003), ten fundamental principles, representing an interpretation of the central themes of Activity Theory applicable to an understanding of organisational work practices, were derived and are summarised below.

- **Principle 1: Activities are hierarchical** An activity consists of one or more actions. Each action consists of one or more operations.
- **Principle 2: Activities are communal** An activity almost always involves a community of participants working towards a common objective.
- **Principle 3: Activities are contextual** Contextual conditions and circumstances deeply affect the way the objective is achieved in any activity.
- **Principle 4: Activities are dynamic** Activities are never static but evolve asynchronously, and historical analysis is often needed to understand the current context of the activity.
- **Principle 5: Activities are mediated** An activity is mediated by tools, rules and divisions of labour.
- **Principle 6: Actions are chosen contextually** A repertoire of actions and operations is created, maintained and made available to any activity, which may be performed by making contextual choices from the repertoire.
- **Principle 7: Actions are understood contextually** The immediate goal of an action may not be identical to the objective of the activity of which the action is a component. It is enough to have an understanding of the overall objective of the activity to motivate successful execution of an action.
- **Principle 8: Plans guide work** A plan is not a blueprint or prescription of work to be performed, but merely a guide which

is modified depending on context during the execution of the work.

- **Principle 9: Exceptions have value** Exceptions are merely deviations from a pre-conceived plan. Deviations will occur with almost every execution of the plan, and give rise to a learning experience which can then be incorporated into future executions.
- **Principle 10: Granularity based on perspective** A particular piece of work might be an activity or an action depending on the perspective of the viewer.

Table 1 shows a summary mapping of the 10 principles to a set of criteria that a PAIS supporting the principles would meet.

Activity Theory offers a number of interesting insights into process management domains, particularly the related issues of adaptability, flexibility, evolution and exception handling. The principles derived in this chapter have formed the theoretical foundations for the implementation and deployment of the system described in the following sections.

4. Conceptual Framework

In the previous section, a set of principles was derived from Activity Theory and applied to the issues of flexibility and exception handling for workflow systems. From that mapping of principles to issues, it was found that:

(1) Workflow management systems typically have trouble supporting all but the most rigid business processes precisely because their frameworks are based on computing metaphors rather than accepted ideas of actual work practices.
(2) A workflow management system that sought to overcome those issues must be built around a framework that better mirrors the way people perform work activities in organisational environments.

The consideration of these findings formed the conceptual foundations of a discrete service that transforms otherwise static workflow processes into fully flexible and dynamically extensible process instances that are also supported by dynamic exception handling capabilities. That service has been named the *Worklet Service*.

4.1. **Worklets: A Conceptualisation.**

Fundamentally, a workflow management system that is based on the principles derived from Activity Theory would satisfy the following criteria:

- *A flexible modelling framework* — a process model is to be regarded as a guide to an activity's objective, rather than a prescription for it;

TABLE 1. Summary mapping of Activity Theory principles vs. workflow functionality criteria

	Flexibility & Re-use	Adaptation via Reflection	Dynamic Evolution	Locality of Change	Comprehensibility of Models	Exceptions as 'First-Class Citizens'
Activities are Hierarchical	✓			✓	✓	
Activities are Communal			✓			
Activities are Contextual	✓	✓				
Activities are Dynamic		✓	✓	✓		
Mediation of Activity	✓	✓	✓			
Actions are Chosen Contextually	✓					✓
Actions are Understood Contextually			✓	✓		
Plans Guide Work		✓			✓	✓
Exceptions have Value		✓	✓			✓
Granularity Based on Perspective					✓	

- *A repertoire of actions* — extensible at any time, the repertoire would be made available for each task during each process instance;
- *Dynamic, contextual choice* — from the repertoire at runtime by considering the specific context of the executing instance; and
- *Dynamic process evolution* — allow the repertoire to be dynamically extended, thus providing support for unexpected process deviations for all current and future instantiations of the process, leading to natural process evolution.

Thus, to accommodate flexibility, such a system would provide each task of a process instance with the ability to be linked to an extensible repertoire of actions, one of which to be contextually and dynamically chosen at runtime to carry out the task. To accommodate exception handling, such a system would provide an extensible repertoire of exception-handling processes to each process instance, members of which to be contextually and dynamically chosen to handle exceptions as they occur.

Using a service-oriented architecture, such a system, the Worklet Service, has been implemented. To support flexibility, the service presents the 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).[2] A top-level or parent process model is developed that describes the workflow at a macro level. From a manager process instance, worklets may be contextually selected and invoked from the repertoire of each enabled task, using an associated extensible set of selection rules. New worklets may be added to the repertoire of a task at any time (even during process execution) as different approaches to completing a task are developed, derived from the context of each process instance. Importantly, the new worklet becomes an implicit part of the process model for all current and future instantiations, avoiding issues of migration and version control (van der Aalst & Basten, 2002; van der Aalst, 2001; Kradolfer & Geppert, 1999; Joeris & Herzog, 1998). In this way, the process model undergoes implicit, dynamic, natural evolution.

In addition, for each anticipated exception, a separate repertoire of exception handling processes, known as *exlets* may be defined, to be dynamically incorporated into a running process instance on an as-needed basis. That is, for any exception that may occur at the task, case instance or specification level, a repertoire of exlets may be provided, the most appropriate one system-selected at runtime based on the context of the case and the type of exception that has occurred.

[2]In Activity Theory terms, a worklet may represent one action within an activity, or may represent an entire activity.

Further, worklets that are invoked as compensation processes within an exlet are constructed *in exactly the same way* as those created to support flexibility, which in turn are constructed in the same way as ordinary, static process models.

In the occurrence of an unanticipated exception (i.e. an event for which a handling exlet has not yet been defined), then either an existing exlet can be manually selected (re-used) from the repertoire, one may be adapted on the fly to handle the immediate situation, or a new exlet constructed and immediately deployed, in each case allowing execution of the process instance that raised the exception to take the necessary action and either continue unhindered, or, if specified in the exception handler, to terminate, as required. Crucially, the method used to handle the new exception and a record of its context are captured by the system and immediately become an implicit part of the parent process model, and so a history of the event and the method used to handle it is recorded for future instantiations.

4.2. **Context, Rules and Worklet Selection.** For any situation, there are multiple situational and personal factors that combine to influence a choice of action. That set of factors that are deemed to be *relevant* to the current situation we call its *context*.

A taxonomy of contextual data that may be recorded and applied to a workflow instance may be categorised as follows (examples are drawn from a medical treatment 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 when created, created by, times invoked, last invoked, current status; and role or agent descriptors such as experience, skills, rank, history with this process and/or task and so on. Process execution states and process log data also belong to this category.
- **Case dependent with *a priori* knowledge:** that set of data that are known to be pertinent to a particular case when it is instantiated. Generally, this data set reflects the data variables of a particular process instance. Examples are: patient name and id, blood pressure readings, height, weight, symptoms and so on; deadlines both approaching and expired; and diagnoses, treatments and prescribed medications.
- **Case dependent with no *a priori* knowledge:** that set of data that only becomes known when the case is active and deviations from the known process occur. Examples in this category may include complications that arise in a patient's condition after triage, allergies to certain medications and so on.

Methods for capturing contextual propositions typically focus on collecting a complete set of knowledge from an 'expert' and representing it in a computationally suitable way (Kang et al., 1998). Such approaches depend heavily on the expert's ability to interpret their own expertise and express it in non-abstract forms (Manago & Kodratoff, 1987). Consequently, the lack of systematic dissemination of expertise has proved a major barrier to the development and use of improvements in exception handling methods in business processes (Klein & Dellarocas, 2000).

One bottom-up approach to the capture of contextual data that offers an alternative method to global knowledge construction is *Ripple Down Rules* (RDR), which comprise a hierarchical set of rules with associated exceptions, first devised by Compton and Jansen (1988).

The fundamental feature of RDR is that it avoids the difficulties inherent in attempting to pre-compile a systematic understanding, organisation and assembly of all knowledge in a particular domain. The RDR method is well established and fully formalised (Scheffer, 1996) 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 (Pacific Knowledge Systems, 2003).

An RDR Knowledge Base is a collection of rules of the form "if *condition* then *conclusion*" (together with other associated descriptors), conceptually arranged in a binary tree structure (cf. Figure 1. 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 (Drake & Beydoun, 2000). 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.

If the conclusion returned is found to be unsuitable for a particular instance — that is, while the conclusion was correct based on the current rule set, the context of the instance make the conclusion an inappropriate choice — a new rule is formulated that defines the contextual differences between the instance and the selected rule 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.

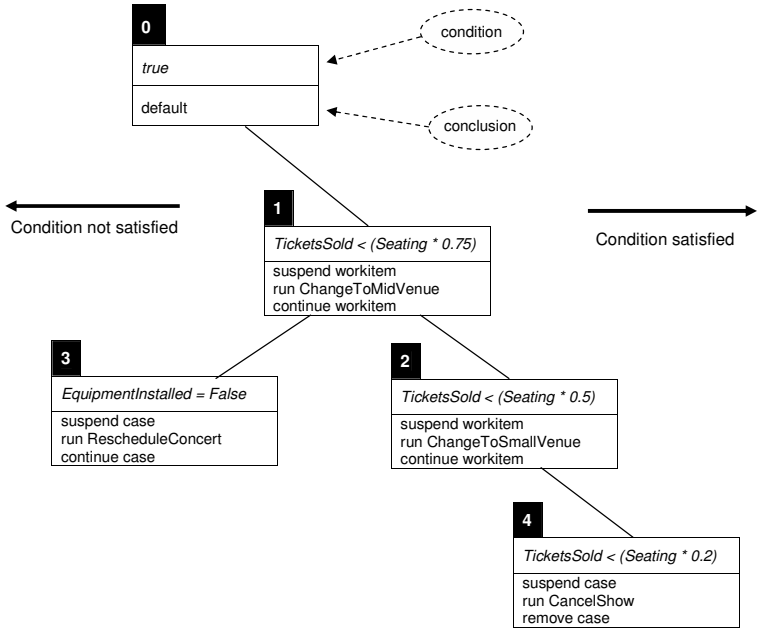

FIGURE 1. Example rule tree (ItemPreConstraint tree for DoShow task of OrganiseConcert)

- 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 rule set 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 (Kang et al., 1998).

Each rule node incorporates a set of case descriptors, called the 'cornerstone case', which describe the actual case context that was the catalyst for the creation of the rule. When a new rule is added to the rule set, its condition 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 — only the factor or factors that make it necessary to handle the current case in a different fashion to the cornerstone case are required 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-value pairs 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 rule sets evolve dynamically, from general to specific, through experience gained as they are applied.

Ripple-Down Rules are well suited to the worklet and exlet selection processes, since they:

- provide a method for capturing relevant, localised contextual data;
- provide a hierarchical structuring of contextual rules;
- do not require the top-down construction of a global knowledge base of the particular domain prior to implementation;
- explicitly provide for the definition of exceptions at a local level;
- do not require expert knowledge engineers for its maintenance; and
- allow a rule set to evolve and grow, thus providing support for a dynamic learning system.

Each worklet is a representation of a particular situated action that relies on the relevant context of each instance, derived from case data and other (archival) sources, to determine whether it is invoked to fulfil a task in preference to another worklet within the repertoire. When a new rule is added, a worker describes the contextual conditions as a natural part of the work they perform[3]. 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.

4.3. **A conceptual framework for exception handling.** The workflow exception patterns (Russell et al., 2006) were developed with the general aim of providing a conceptual framework for exception handling in workflow systems. They aim to describe the notion of a workflow exception in a general sense and the various ways in which they can be triggered and handled. An exception is anticipated to be a distinct, identifiable event which occurs at a specific point in time during the execution of a process instance. The manner in which the exception is handled will depend on the type of exception that has been detected. The types of events that give rise to exceptions can be classified into five distinct groups:

Work Item Failure where during the course of its execution, a work item to unable progress any further. This may be a consequence of

[3]In practice, the worker's contextual description would be passed to an administrator, who would add the new rule.

software or hardware failure or may be triggered by the user themselves as a means of signalling failure to the workflow engine;

Deadline Expiry where a deadline that is associated with a work item (either for commencement or completion) is not met;

Resource Unavailability where the resources that are required in order to commence or complete a work item are not available;

External Trigger where signals are received from the operating environment that an event has occurred that impacts on the work item or process instance and requires some form of handling; and

Constraint Violation where constraints have been specified in relation to 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.

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.

Exception handling at work item level. In general an exception will relate to a specific work item in a process instance and the way in which the exception is handled depends on the current state of execution of the work item. Figure 2 illustrates as solid arrows the states through which a work item progresses during normal execution. Figure 2 also shows fifteen strategies as dashed arcs from one work item state to another, which to distinct approaches for handling the current item in various states when a specific type of exception is detected.

Exception handling at case level. Exceptions always occur in the context of one or more cases (process instances) 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;

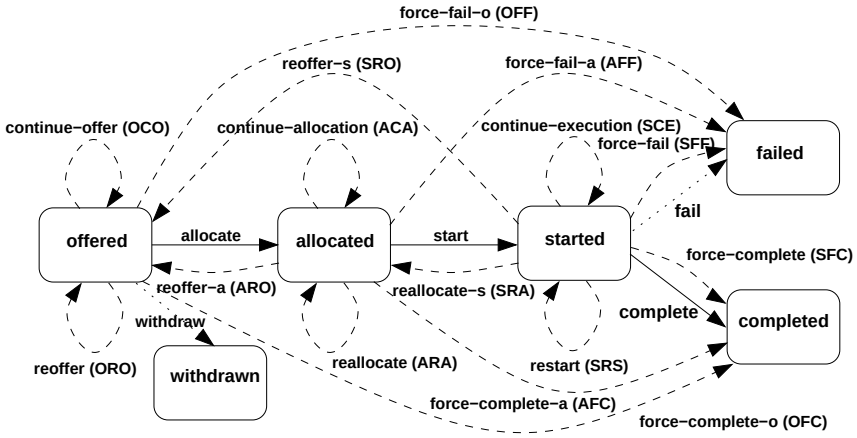

FIGURE 2. Options for handling work items

(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 both this and all other currently executing 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).

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. 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.

5. Implementation

The *Worklet Service* has been implemented as a YAWL Custom Service (van der Aalst & ter Hofstede, 2005; van der Aalst et al., 2004). The YAWL environment was chosen as the implementation platform

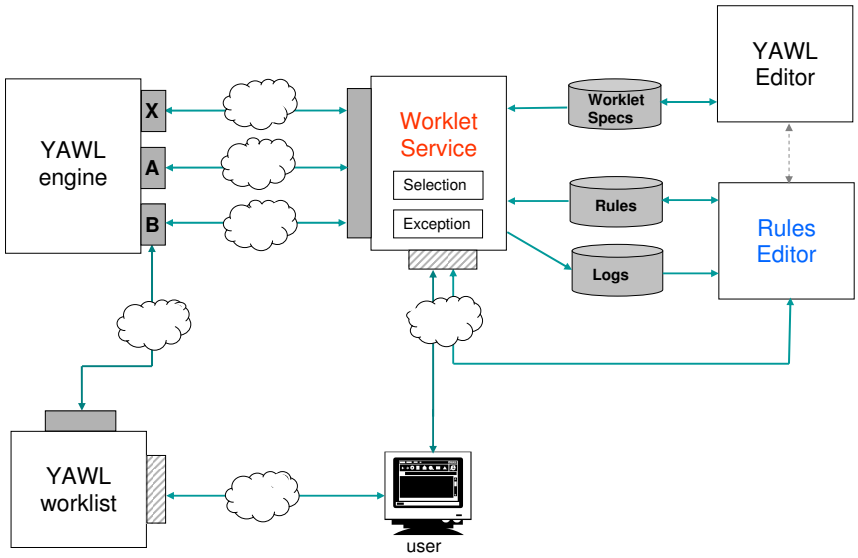

FIGURE 3. External Architecture of the Worklet Service

since it provides a very powerful and expressive workflow language based on the workflow patterns identified together with a formal semantics (van der Aalst et al., 2003). It also provides a workflow enactment engine, and an editor for process model creation, that support the control flow, data and resource perspectives. The YAWL environment is open-source and has a service-oriented architecture, allowing the worklet paradigm to be developed as a service independent to the core engine. Thus the deployment of the Worklet Service is in no way limited to the YAWL environment, but may be ported to other environments (for example, BPEL engines) by making the necessary links in the service interface.

Custom YAWL services interact with the YAWL engine through XML/HTTP messages via certain interface endpoints. Specifically, custom services may elect to be notified by the engine when certain events occur in the life-cycle of nominated process instantiations (for example: when a workitem becomes enabled, when a workitem is cancelled, when a case completes), to signal the creation and completion of process instances and workitems, or to notify of certain events or changes in the status of existing workitems and cases.

The Worklet Service (including its rules editor), source code and accompanying documentation, can be freely downloaded from `http://www.yawl-system.com`.

5.1. **Service Architecture.** The service consists of a number of J2EE classes and servlet pages, organised in a series of packages.

The external architecture of the Worklet Service is shown in Figure 3. The entities 'Worklet Specs', 'Rules' and 'Logs' comprise the *worklet repository*. The service uses the repository to store rule sets, worklet

specifications for uploading to the engine, and generated process and audit logs. Any YAWL specification may have an associated rule set. The rule set for each specification is stored as XML data in a disk file within the worklet repository. The YAWL Process Editor is used to create new worklet specifications, and may be invoked from the Worklet Rules Editor, which is used to create new or augment existing rule sets, making use of certain selection logs to do so, and may communicate with the Worklet Service via a JSP/Servlet interface to override worklet selections following rule set additions. The service also provides servlet pages that allow users to directly communicate with the service to raise external exceptions and to create and carry out administration tasks.

The *Worklet Service* comprises two discrete but complementary sub-services: a *Selection Service*, which enables dynamic flexibility for process instances, and an *Exception Service*, which provides facilities to handle both expected and unexpected process exceptions at runtime.

The Selection Service. The Selection Service enables dynamic flexibility by allowing a process designer to designate certain workitems to each be substituted at runtime with a dynamically selected *worklet*, which contextually handles one specific task in a larger, composite process activity. Each worklet is a complete extended workflow net (EWF-net) compliant with Definition 1 of the YAWL semantics (van der Aalst & ter Hofstede, 2005). Each worklet may be designed and provided to the Selection Service at any time, *even while a parent process instance is executing*, as opposed to a static sub-process that must be defined at the same time as, and remains a static part of, the main process model.

An extensible repertoire of worklets is maintained by the service for each task in a specification. Each time the service is invoked for a workitem, a choice is made from the repertoire based on the contextual data values within the workitem and other sources, using a set of ripple-down rules to determine the most appropriate substitution.

The workitem is checked out of the workflow enactment engine, the corresponding data inputs of the original workitem are mapped to the inputs of the worklet, and the selected worklet is launched in the engine as a separate case. When the worklet has completed, its output data is mapped back to the original workitem, which is then checked back into the engine, allowing the original process to continue.

From an engine perspective, the worklet and its parent are two distinct, unrelated cases. The Worklet Service tracks the relationships, data mappings and synchronisations between cases, and creates a process log that may be combined with the engine's process logs via case identifiers to provide a complete operational history of each process instance and may be used as a data source for the evaluation of rule conditions.

Each task that is associated with a worklet repertoire is said to be 'worklet-enabled'. This means that a process may contain both worklet-enabled tasks and non-worklet-enabled (or ordinary) tasks. Any process instance that contains a worklet-enabled task will become the parent process instance for any worklets invoked from it.

Importantly, a worklet-enabled task remains a valid (ordinary) task definition, rather than being considered as a vacant 'placeholder' for some other activity (i.e. a worklet). The distinction is crucial because, if an appropriate worklet for a worklet-enabled task cannot be found at runtime (based on the context of the case and the rule set associated with the task), the task is allowed to run as an 'ordinary' task, as it normally would in a process instance. So, instead of the parent process being conceived as a template schema or as a container for a set of placeholders, it is to be considered as a complete process containing one or more worklet-enabled tasks, each of which *may* be contextually and dynamically substituted at runtime.

Worklets may be associated with either an atomic task, or a multiple-instance atomic task. Any number of worklets can form the repertoire of an individual task, and any number of tasks in a particular specification can be associated with the Worklet Service. A worklet may be a member of one or more repertoires — that is, it may be re-used for several distinct tasks within and across process specifications. In the case of multiple-instance tasks, a separate worklet is launched for each child workitem. Because each child workitem may contain different data, the worklets that substitute for them are individually selected, and so may all be instances of different worklet specifications.

The Exception Service. The Exception Service allows designers to define exception handling processes (called *exlets* for parent workflow instances, to be invoked when certain events occur. It has been designed so that the enactment engine, besides providing notifications at certain points in the life cycle of a process instance, needs no knowledge of an exception occurring, or of any invocation of handling processes — all exception checking and handling is provided by the service.

The exception service uses the same repertoire and dynamic rules framework as the selection service. There are, however, two fundamental differences between the two sub-services. First, where the selection service selects a *worklet* as the result of satisfying a rule in a rule set, the result of an exception service rule being satisfied is an *exlet*. Second, while the selection service is invoked for certain nominated tasks in a process, the exception service, when enabled, is invoked for *every* case and task instance executed by the enactment engine, and will detect and handle up to ten different kinds of process exceptions.

As part of the exlet definition, a process designer may choose from various actions (such as cancelling, suspending, completing, failing and

restarting) and apply them at a workitem, case and/or specification level. And, since the exlets can include compensatory worklets, the original parent process model only needs to reveal the actual business logic for the process, while the repertoire of exlets grows as new exceptions arise or different ways of handling exceptions are formulated.

An extensible repertoire of exlets is maintained by the service for each type of potential exception within each workflow specification. Each time the service is notified of an exception event, either actual or potential (i.e. a constraint check) the service first determines whether an exception has in fact occurred, and if so, where a rule tree for that exception type has been defined, makes a choice from the repertoire based on the type of exception and the context of the workitem/case.

If an exlet executed by the exception service contains a compensation action (i.e. a worklet to be executed as a compensatory process) it is run as a separate case in the enactment engine, so that from an engine perspective, the worklet and its 'parent' (i.e. the process that invoked the exception) are two distinct, unrelated cases. Figure 4 shows the relationship between a 'parent' process, an exlet repertoire and a compensatory worklet, using an *Organise Concert* process as an example. Since a worklet is launched as a separate case, it is treated as such by the Worklet Service and so may have its own worklet/exlet repertoire.

The Service responds to the following exception types:

Constraint Types. Constraints are rules that are applied to a workitem or case immediately before and after execution. Thus, there are four sub-types of constraint exception: *CasePreConstraint*, *ItemPreConstraint*, *ItemPostConstraint*, and *CasePostConstraint*.

The service receives notification from the workflow engine when each of these constraint events occurs within each case instance, then checks the rule set associated with the specification to determine, firstly, if there are any rules of that exception type defined for the specification, and if so, if any of the rules evaluate to true using the contextual data of the case or workitem. If the rule set finds a rule that evaluates to true for the exception type and data, an associated exlet is selected and invoked.

TimeOut. A timeout event occurs when a deadline set for a workitem is reached. In this case, the workflow engine notifies the service of the timeout event, passing to the service a reference to the workitem. If the workitem has an associated timeout rule set, the relevant exlet is invoked.

Externally Triggered Types. Externally triggered exceptions occur, not through context internal to the process instance, but because of the occurrence of an event in the external environment, that may have an effect on the continuing execution of the process. Notification of these events is typically triggered by a user or administrator. Depending on the actual event and the context of the case or workitem, a particular

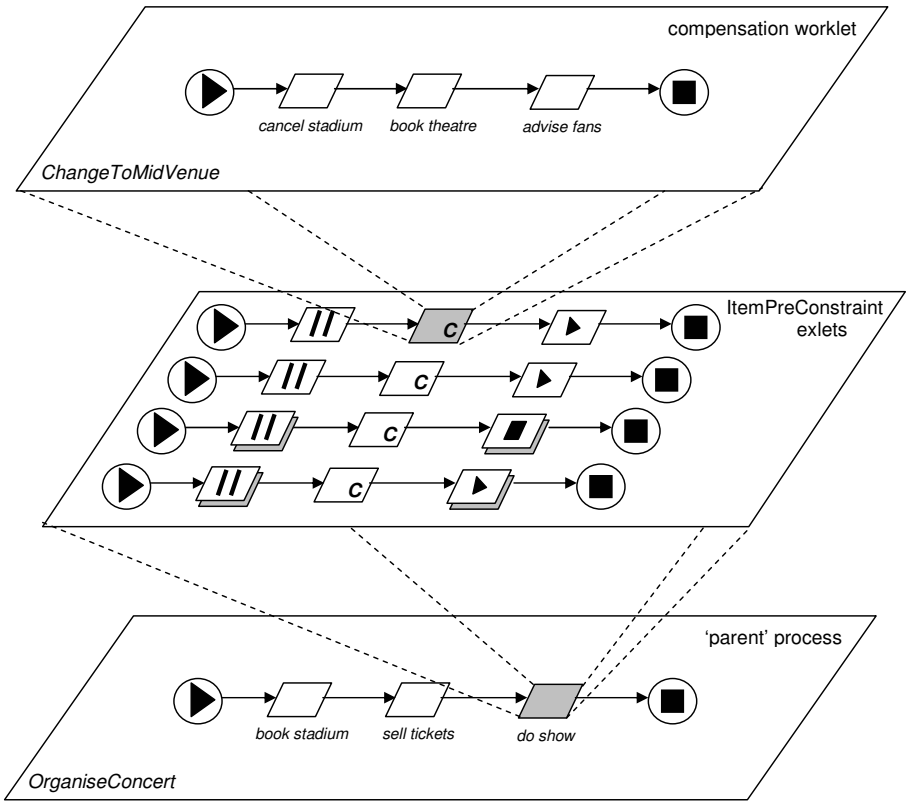

FIGURE 4. Process – Exlet – Worklet Hierarchy

exlet will be invoked if the associated rule exists. There are two types of externally triggered types, *CaseExternalTrigger* (for case-level events) and *ItemExternalTrigger* (for item-level events).
ItemAbort. An ItemAbort event occurs when a workitem reports that it has been aborted before normal completion.
ResourceUnavailable. This event occurs when an attempt has been made to allocate a workitem to a resource and the resource reports that it is unable to accept the allocation or the allocation cannot proceed.
ConstraintViolation. This event occurs when a data constraint has been violated for a workitem *during* its execution (as opposed to pre- or post-execution).

When any of the above exception event notifications occur, an appropriate exlet for that event, if defined, will be invoked. Each exlet may contain any number of steps, or *primitives*, and is defined graphically using a Rules Editor (cf. Figure 5.

The set of primitives that may be used to construct an exlet (as seen left to right on the left of Figure 5) are:

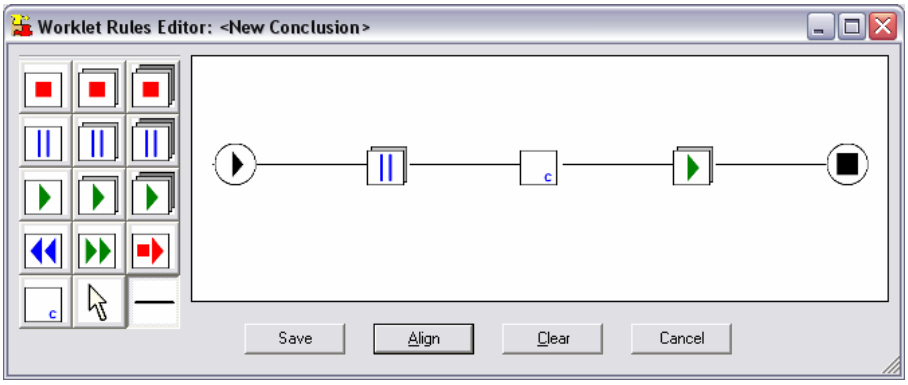

FIGURE 5. Example Handler Process in the Rules Editor

- *Remove WorkItem*: removes (or cancels) the workitem; execution ends, and the workitem is marked with a status of cancelled. No further execution occurs on the process path that contains the workitem.
- *Remove Case*: removes the case. Case execution ends.
- *Remove All Cases*: removes all case instances for the specification in which the task of which the workitem is an instance is defined, or of which the case is an instance.
- *Suspend WorkItem*: suspends (or pauses) execution of a workitem, until it is continued, restarted, cancelled, failed or completed, or the case that contains the workitem is cancelled or completed.
- *Suspend Case*: suspends all 'live' workitems in the current case instance (a live workitem has a status of fired, enabled or executing), effectively suspending execution of the entire case.
- *Suspend All Cases*: suspends all 'live' workitems in all of the currently executing instances of the specification in which the task of which the workitem is an instance is defined, effectively suspending all running cases of the specification.
- *Continue WorkItem*: un-suspends (or continues) execution of the previously suspended workitem.
- *Continue Case*: un-suspends execution of all previously suspended workitems for the case, effectively continuing case execution.
- *Continue All Cases*: un-suspends execution of all workitems previously suspended for all cases of the specification in which the task of which the workitem is an instance is defined or of which the case is an instance, effectively continuing all previously suspended cases of the specification.
- *Restart WorkItem*: rewinds workitem execution back to its start. Resets the workitem's data values to those it had when it began execution.

- *Force Complete WorkItem*: completes a 'live' workitem. Execution of the work-item ends, and the workitem is marked with a status of *ForcedComplete*, which is regarded as a successful completion, rather than a cancellation or failure. Execution proceeds to the next workitem on the process path.
- *Force Fail WorkItem*: fails a 'live' workitem. Execution of the workitem ends, and the workitem is marked with a status of *Failed*, which is regarded as an unsuccessful completion, but not as a cancellation — execution proceeds to the next workitem on the process path.
- *Compensate*: run one or more compensatory processes (i.e. worklets). Depending on previous primitives, the worklets may execute simultaneously to the parent case, or execute while the parent is suspended.

Optionally, an *array* of worklets may be defined for a particular compensation primitive — when multiple worklets are defined for a particular compensation primitive via the Rules Editor, they are launched concurrently as a composite compensatory action when the exlet is executed.

The Selection and Exception sub-services can be used in combination within particular case instances to achieve dynamic flexibility *and* exception handling simultaneously.

6. Exemplary Study

Film and television production is a multi-billion dollar industry. In Australia alone, there are over two thousand film and video production services actively employing almost twenty thousand people (Trewin, 2004). However, the industry is extremely competitive and has become progressively global in its scope. Even though the work processes of the industry are highly creative and goal-oriented, organisations are increasingly recognising the value of more conventional business management strategies, such as PAIS, to gain and maintain a competitive edge (Lee & Holt, 2006; Irving & Rea, 2006).

That is not to say that any workflow solution is able to be applied across the board to support all aspects of a film production process. But there are many aspects of the industry where meaningful benefits can be gained through the use of a workflow solution to assist in the management of a project, including:

- back-office administrative and support processes;
- the allocation of resources to tasks;
- routing of film stock, documentation and other materials amongst employees; and
- facilitating inter-team communication and goal-setting.

This study will examine a process occurring in the post-production phase and discuss the applicability of implementing a worklet-service-based solution. The process originates from a cooperative project between the *Queensland University of Technology* (QUT) and the *Australian Film, Television, and Radio School* (AFTRS) within the context of QUT's *Centre of Excellence for Creative Industries and Innovation*.

Rather than coming at the end of production (as the name might imply), work within the post-production phase operates concurrently with several other phases of production. Tasks in this phase include the merging of video and audio components (voice, sound effects, music and so on) and editing to produce a coherent, final piece, and as such includes tasks that are both acutely technical and highly creative. The process, referred to as the *Master* process, can be logically divided into three phases:

- **Pre-Edit:** The pre-edit phase begins with the delivery of the day's footage ('the rushes') to the post-production team. There are two possible entry points into the process, one for each type of media that may be used (film and videotape). It may be the case that both types of media are used for a particular set of rushes, so in terms of the model's entry points, inputs may arrive at both simultaneously. Videotape does not require the same degree of processing as film, but for both media types, a low resolution copy is digitised and stored on computer file to be used as a guide for the remaining process. Accompanying the film and/or tape is the 'rushes paperwork', a set of documentation which may include items such as an annotated script, and video and audio reports. This documentation is regarded as an important source of information about the footage, and is thus made available throughout the post production process.
- **Edit:** In the edit phase, the video and audio components are handled separately. Further, video editing is divided into low and high resolution edits. Low resolution editing is represented by the *Offline* task, which allows editing decisions to be made and documented before the high resolution editing begins. The result of the *Offline* task is the EDL (Edit Decision List). Video Effects Production takes place concurrently with the *Offline* task. When the *Offline* task completes, the high-resolution editing, along with the sound and music editing, can begin. For film, the high resolution editing takes place in the *Film Finishing* task, where the original negative is spliced into pieces, some of which are rejoined; for tape, it occurs in the *Online* task, where the video is rearranged using an editing suite and recorded to a tape master. Both take the EDL output from the low resolution edit and each performs the actions listed in the respective EDL on distribution quality media.

- **Post-Edit:** After the edit phase, an edited, high resolution, distribution quality film and/or tape, together with completed visual effects and sound and music, is completed, and now must be 'finished' for distribution. The finishing may be required for any or all of the film, tape and disk mediums, which are output in the form of a release print, master tape or release version respectively.

The process description reveals some of the complicating factors that come in to play when rendering this process to a particular modelling framework. For instance, it contains constraints which are designed to remove some tasks from the eventual process if certain preceding tasks were not included in a particular configuration of the process. For example, the removal of the *Prepare Film for Edit* and *Film Finish* tasks is required if the rushes were not received on film; similarly, the *Online* task will not be required if tape media was not received. In addition, because there are two entry points, there are three possible media combinations that may start a case instance (i.e. tape, film, or both tape and film), and so a nominal model will require a number of OR splits and joins to accommodate the various combinations and the tasks they entail.

A representation of the process in the YAWL language shows complications in the describing the process and its possible flow paths via a static model (Figure 6). There are several OR splits and joins; conditionals are required to be embedded into each OR split output arc to determine whether they 'fire' or not. All are dependent simply on which media formats have been supplied to the process. For example, the first OR split task controls whether one or both arcs fire (one for tape, one for film); the OR split preceding the *Online* and *File Finish* tasks has a similar function, and so on. Thus in static representations such as this, the control flow logic is embedded into the business process logic. As a result, it is not obvious from the model which path may be taken during a particular instance.

With the flexibility mechanisms available through the Worklet Service, the process can be modelled without much of the complexity, particularly by negating the need for the OR splits and joins. Figure 7 shows the worklet-enabled process. Immediately apparent is the fact that, in this case, all of the OR splits and joins have been removed from the process model.

The first task in the process, *PrepareForEdit*, is worklet-enabled. Associated with this task is the Selection rule tree shown in Figure 8. The rule tree shows that, if either tape or film has been supplied, the corresponding rule will be satisfied and the service will launch the appropriate worklet for that media. If the rushes have been delivered on both film and tape media, node 2 will be last satisfied, resulting in the launching of two discrete worklets, one for each medium (the two

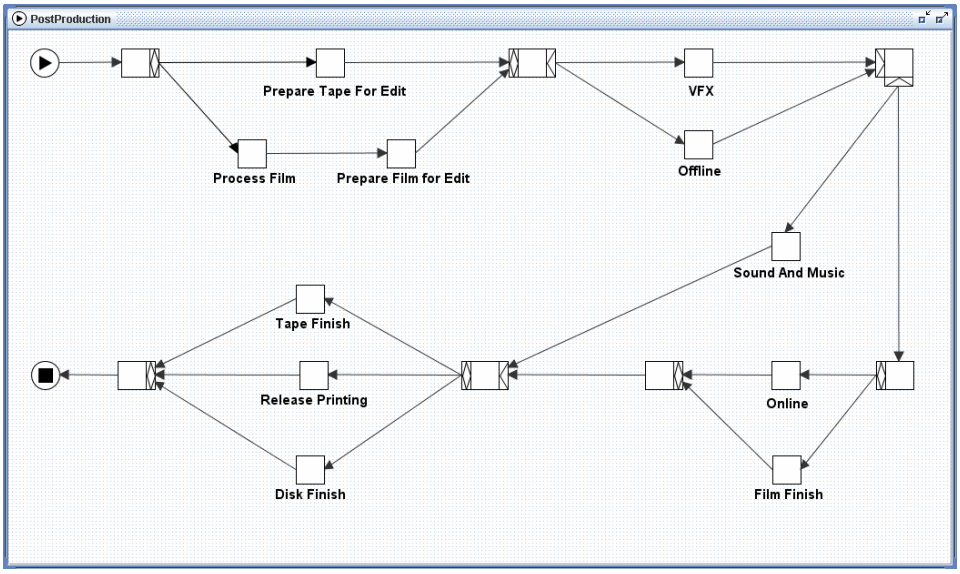

FIGURE 6. Static Post Production Master Process (YAWL language)

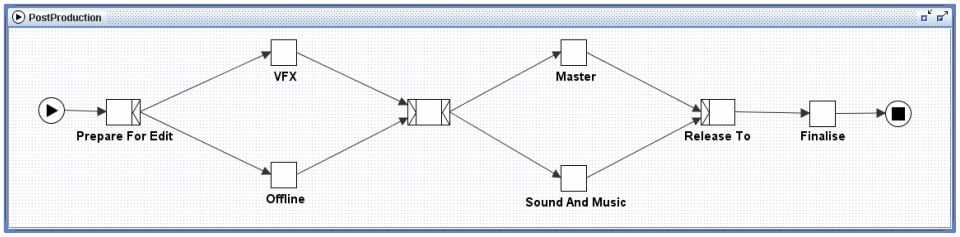

FIGURE 7. Post Production Master Process (Worklet-Enabled)

worklets are shown in Figure 9). Note that the conditional expressions for nodes 2 and 3 are identical in this tree, but their conclusions differ — node 3 will be tested if node 1 evaluates to false (i.e. there is no tape media), while node 2 will be tested only if node 1 evaluates to true.

The worklet-enabled *Master* task performs a similar service to *PrepareForEdit* — it will launch a worklet to carry out the *Online* process if tape media is provided, and/or for the *Film Finish* process if film media is provided.

The *Finalise* task models the processing of 'finishing' the output for distribution. There are three possible sub-processes to perform: tape finish, disk finish and release printing — each has a corresponding worklet in the specification's repertoire. Therefore, there are six possible worklet launch combinations, as specified in the selection rule tree for the *Finalise* task (Figure 10). Each worklet consists of one task, corresponding to each of the three tasks at the post-edit end of the original static process model.

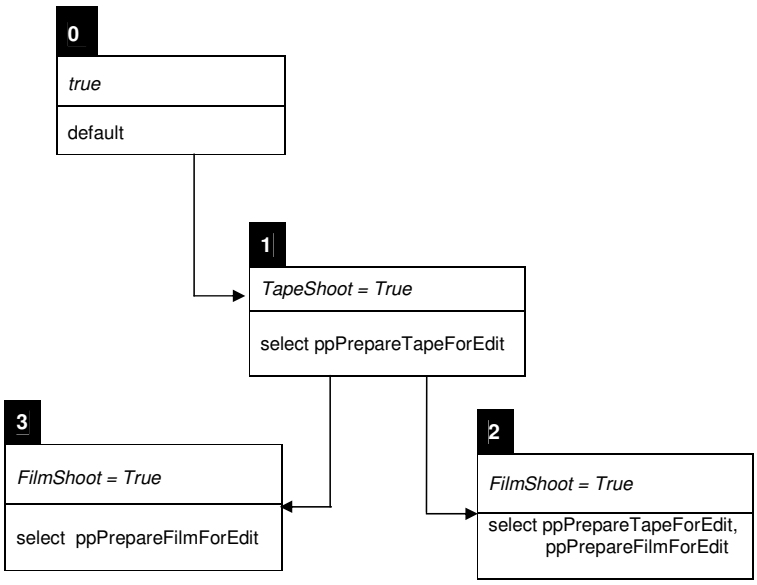

FIGURE 8. Selection Rule Tree for the PrepareForEdit task

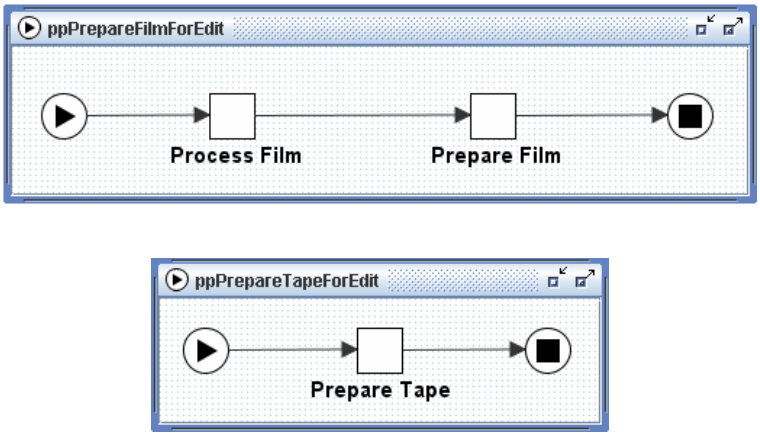

FIGURE 9. Worklets ppPrepareFileForEdit and ppPrepareTapeForEdit

In summary, the Worklet Service allows a parent or master process to be defined without much of the explicit branching mechanisms necessary in the control flow of static models. As a result, the parent process models are cleaner, easier to verify and maintain, and easier for stakeholders to gain an understanding of the process logic. Once the parent process is worklet-enabled, it is able to access all the features of the worklet paradigm, including support for exception handling. Some exceptions that may occur in a post production process include damaged film or tape stock, equipment malfunctions and breakdowns, and time and budget overruns, to name but a few. All of these exceptions

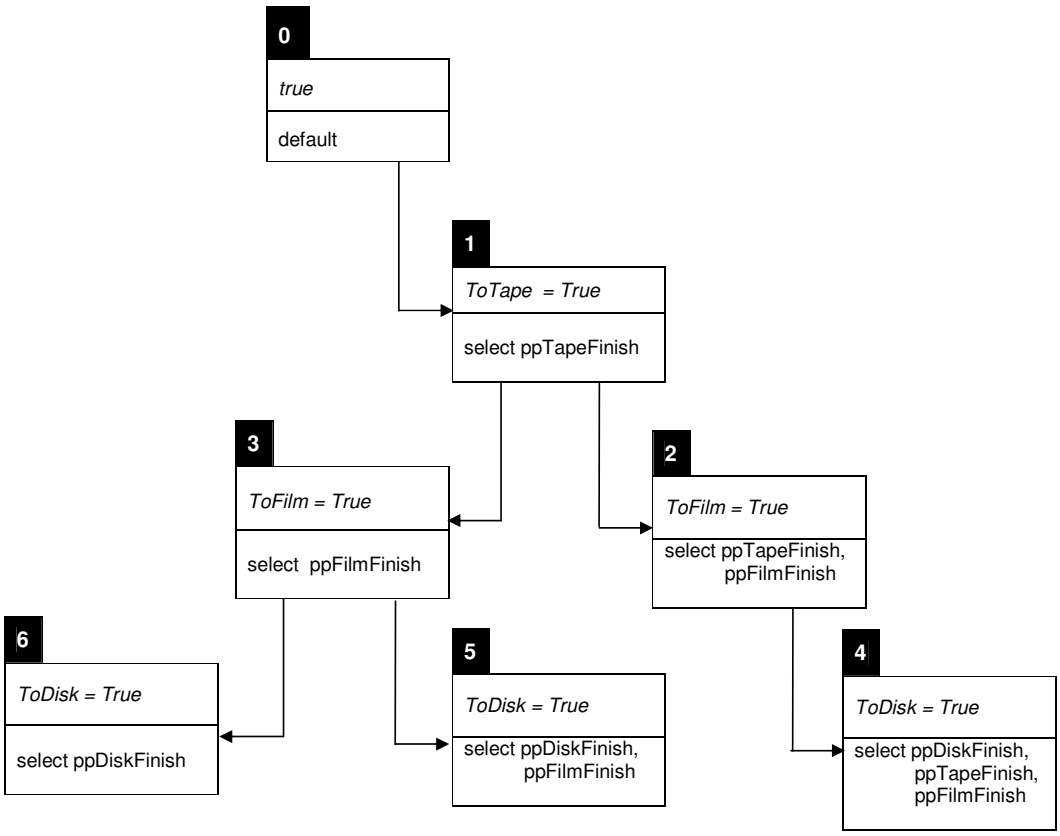

FIGURE 10. Selection Rule Tree for the Finalise Task

may be handled by adding an appropriate exlet to the specification's repertoire.

7. CONCLUSION

This chapter began by identifying key problems that describe the fundamental limitations of current workflow technologies with respect to the rigidity enforced by the inflexible frameworks employed, and the consequent difficulties in placing more dynamic, information intensive processes within those frameworks. Then, a description of ten principles derived from Activity Theory that represent an interpretation of its central themes applicable to understanding organisational work practices was provided.

Based on the derived principles of Activity Theory, the Worklet Service was then conceptualised, implemented and validated. A primary feature of the service is that it has been designed and implemented as a discrete service, and so offers all of its benefits to a wide range of workflow management systems, allowing them to fully 'worklet-ise' their otherwise static processes. The Worklet Service:

- Keeps the parent model clean and relatively simple;

- Promotes the reuse of sub-processes in different models;
- Allows standard processes to be used as exception handling compensation processes, and vice versa;
- Maintains an extensible repertoire of actions that can be constructed during design and/or runtime and can be invoked as required;
- Allows a specification to implicitly build a history of executions, providing for a learning system that can take the appropriate actions for certain contexts automatically;
- Maintains a repertoire of fully encapsulated, discrete worklets that allow for easier verification and modification;
- Allows a model to evolve without the need to stop and modify the design of the whole specification when an exception occurs;
- By de-coupling the monolithic process model, models can be built that vary from loosely to tightly defined and so supports late binding of processes; and
- Allows a model to be considered from many levels of granularity.

There are a number of further research topic possibilities that arise from this work, such as: deeper empirical studies comparing the worklet approach to classic workflow approaches and measuring the relative benefits of each; porting the Worklet Service to other workflow systems (for example, IBM Websphere and/or Oracle BPEL); exploring the advantages of mixing different modelling styles and approaches, leading to recommendations of in what circumstances the various approaches are best used; and stronger support for process mining analysis using both the process logs generated by the service and the structure, content and evolution of the various ripple-down rule sets of specifications.

In summary, through a combination of the framework on which it is built and the mechanisms available through both its selection and exception handling sub-services, the Worklet Service offers a wide-ranging solution to the issues of flexibility and exception handling in process-aware information systems. In fact, the benefits offered through each sub-service can be combined to deliver a far-reaching set of capabilities that serve the needs of a wide variety of work environments and processes.

References

van der Aalst, W. (2001). Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, *3*(3), 297–317.

van der Aalst, W., Aldred, L., Dumas, M., & ter Hofstede, A. (2004). Design and implementation of the YAWL system. In A. Persson, & J. Stirna (Eds.) *Proceedings of The 16th International Conference on Advanced Information Systems Engineering (CAiSE 04)*, vol. 3084

of *Lecture Notes in Computer Science*, (pp. 142–159). Riga, Latvia: Springer Verlag.

van der Aalst, W., & Basten, T. (2002). Inheritance of workflows: An approach to tackling problems related to change. *Theoretical Computer Science*, *270*((1-2)), 125–203.

van der Aalst, W., & Berens, P. (2001). Beyond workflow management: Product-driven case handling. In S. Ellis, T. Rodden, & I. Zigurs (Eds.) *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, (pp. 42–51). New York: ACM Press.

van der Aalst, W., & van Hee, K. (2004). *Workflow Management: Models, Methods and Systems*. Cambridge, Massachusetts: The MIT Press, New Ed ed.

van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., & Barros, A. (2003). Workflow patterns. *Distributed and Parallel Databases*, *14*(3), 5–51.

van der Aalst, W., & ter Hofstede, A. (2005). YAWL: Yet Another Workflow Language. *Information Systems*, *30*(4), 245–275.

van der Aalst, W., Weske, M., & Grünbauer, D. (2005). Case handling: A new paradigm for business process support. *Data & Knowledge Engineering*, *53*(2), 129–162.

Adams, M. (2007). *Facilitating Dynamic Flexibility and Exception Handling for Workflows*. Phd thesis, Queensland University of Technology.

Adams, M., Edmond, D., & ter Hofstede, A. H. (2003). The application of activity theory to dynamic workflow adaptation issues. In *Proceedings of the 2003 Pacific Asia Conference on Information Systems (PACIS 2003)*, (pp. 1836–1852). Adelaide, Australia.

Bardram, J. E. (1997). I love the system - I just don't use it! In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work (GROUP'97)*, (pp. 251–260). Phoenix, Arizona, USA: ACM.

Barthelmess, P., & Wainer, J. (1995). Workflow systems: a few definitions and a few suggestions. In *Proceedings of the ACM Conference on Organizational Computing Systems (COOCS'95)*, (pp. 138–147). Milpitas, California, USA: ACM.

Berens, P. (2005). *The FLOWer Case Handling Approach: Beyond Workflow Managment*, chap. 15, (pp. 363–395). In Dumas et al. (2005).

Bider, I. (2005). Masking flexibility behind rigidity: Notes on how much flexibility people are willing to cope with. In J. Castro, & E. Teniente (Eds.) *Proceedings of the CAiSE'05 Workshops*, vol. 1, (pp. 7–18). Porto, Portugal: FEUP Edicoes.

Bødker, S., & Greenbaum, J. (1993). Design of information systems: Things versus people. In E. Green, J. Owen, & D. Pain (Eds.)

Gendered by Design?: Information Technology and Office Systems, chap. 3, (pp. 53–63). London: Taylor and Francis.

Borgida, A., & Murata, T. (1999). Tolerating exceptions in workflows: a unified framework for data and processes. In *Proceedings of the International Joint Conference on Work Activities, Coordination and Collaboration (WACC'99)*, (pp. 59–68). San Francisco, California, USA: ACM Press.

Casati, F. (1998). A discussion on approaches to handling exceptions in workflows. In *Proceedings of the CSCW Workshop on Adaptive Workflow Systems*. Seattle, USA.

Compton, P., & Jansen, B. (1988). Knowledge in context: A strategy for expert system maintenance. In J.Siekmann (Ed.) *Proceedings of the 2nd Australian Joint Artificial Intelligence Conference*, vol. 406 of *Lecture Notes in Artificial Intelligence*, (pp. 292–306). Adelaide, Australia: Springer-Verlag.

COSA (2005). COSA BPM product description. `http://www.cosa-bpm.com/project/docs/COSA_BPM_5_Productdescription_eng.pdf`. Accessed 13 March, 2008.

Drake, B., & Beydoun, G. (2000). Predicate logic-based incremental knowledge acquisition. In P. Compton, A. Hoffmann, H. Motoda, & T. Yamaguchi (Eds.) *Proceedings of the sixth Pacific International Knowledge Acquisition Workshop*, (pp. 71–88). Sydney, Australia.

Dumas, M., van der Aalst, W., & ter Hofstede, A. (Eds.) (2005). *Process-Aware Information Systems: Bridging People and Software through Process Technology*. New York: Wiley-Interscience.

Georgeff, M., & Pyke, J. (2003). Dynamic process orchestration. White paper, Staffware PLC.

Greiner, U., Ramsch, J., Heller, B., Löffler, M., Müller, R., & Rahm, E. (2004). Adaptive guideline-based treatment workflows with adaptflow. In K. Kaiser, S. Miksch, & S. Tu (Eds.) *Proceedings of the Symposium on Computerized Guidelines and Protocols (CGP 2004)*, (pp. 113–117). Prague: IOS Press.

Hendriks-Jansen, H. (1996). *Catching ourselves in the act : situated activity, interactive emergence, evolution, and human thought*. Cambridge, Mass: MIT Press.

Holt, A. W. (1997). *Organized Activity and Its Support by Computer*. Kluwer Academic Publishers, Dordrecht.

IBM (2005). IBM WebSphere MQ Workflow: Concepts and architecture. `http://publibfp.boulder.ibm.com/epubs/pdf/h1262857.pdf`. Accessed 14 March, 2008.

Irving, D. K., & Rea, P. W. (2006). *Producing and Directing the Short Film and Video*. Burlington, Oxford, United Kingdom: Focal Press, 3rd ed.

Joeris, G. (1999). Defining flexible workflow execution behaviors. In P. Dadam, & M. Reichert (Eds.) *Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications*, vol. 24 of *CEUR Workshop Proceedings*, (pp. 49–55). Paderborn, Germany.

Joeris, G., & Herzog, O. (1998). Managing evolving workflow specifications. In *Proceedings of the 3rd IFCIS International Conference on Cooperative Information Systems (CoopIS '98)*, (pp. 310–319). New York, New York, USA: IEEE Computer Society.

Kammer, P., Bolcer, G., Taylor, R., Hitomi, A., & Bergman, M. (2000). Techniques for Supporting Dynamic and Adaptive Workflow. *Computer Supported Cooperative Work (CSCW)*, *9*(3), 269–292.

Kang, B. H., Preston, P., & Compton, P. (1998). Simulated expert evaluation of multiple classification ripple down rules. In *Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling and Management*. Banff, Alberta, Canada.

Klein, M., & Dellarocas, C. (2000). A systematic repository of knowledge about handling exceptions. ASES Working Paper ASES-WP-2000-03 ASES-WP-2000-03, Massachusetts Institute of Technology, Cambridge, MA, United States.

Kradolfer, M., & Geppert, A. (1999). Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In *Proceedings of the 1999 IFCIS International Conference on Cooperative Information Systems (CoopIS'99)*, (pp. 104–114). Edinburgh, Scotland: IEEE Computer Society.

Kuutti, K. (1996). *Activity Theory as a Potential Framework for Human-Computer Interaction Research*, (pp. 17–44). In Nardi (1996).

Lee, J. J., & Holt, R. (2006). *The Producer's Business Handbook*. Burlington, Oxford, United Kingdom: Focal Press, 2nd ed.

Leontiev, A. (1974). The problem of activity in psychology. *Soviet Psychology*, *13*(2), 4–33.

Leymann, F. (2006). Workflow-based coordination and cooperation in a service world. In R. Meersman, & Z. Tari (Eds.) *Proceedings of the 14th International Conference on Cooperative Information Systems (CoopIS'06)*, vol. 4275 of *Lecture Notes in Computer Science*, (pp. 2–16). Montpellier, France: Springer-Verlag.

Manago, M. V., & Kodratoff, Y. (1987). Noise and knowledge acquisition. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, vol. 1, (pp. 348–354). Milano, Italy: Morgan Kaufmann.

zur Muehlen, M. (2004). *Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems*, vol. 6 of *Advances in Information Systems and Management Science*. Berlin: Logos.

Muller, R., Greiner, U., & Rahm, E. (2004). AgentWork: a workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, *51*(2), 223–256.

Mumford, L. (1963). *Technics and Civilization*. Harcourt Brace Jovanovich, New York.

Nardi, B. A. (1996). *Activity Theory and Human-Computer Interaction*, (pp. 7–16). In Nardi (1996).

Nardi, B. A. (Ed.) (1996). *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, Cambridge, Massachusetts.

Oberweis, A. (2005). *Person-to-Application Processes: Workflow Management*, chap. 2, (pp. 21–36). In Dumas et al. (2005).

Pacific Knowledge Systems (2003). Products: Rippledown, `http://www.pks.com.au/products/validator.htm`. Accessed 23 April, 2002.

Palmer, N. (2007). A survey of business process initiatives. `http://wfmc.org/researchreports/Survey_BPI.pdf`. Accessed 4 April, 2008.

Pesic, M., & van der Aalst, W. (2006). A declarative approach for flexible business processes. In J. Eder, & S. Dustdar (Eds.) *Proceedings of the First International Workshop on Dynamic Process Management (DPM 2006)*, vol. 4103 of *Lecture Notes in Computer Science*, (pp. 169–180). Vienna, Austria: Springer-Verlag, Berlin, Germany.

Reichert, M., & Dadam, P. (1997). A framework for dynamic changes in workflow management systems. In *Proceedings of the 8th International Workshop on Database and Expert Systems Applications (DEXA 97)*, (pp. 42–48). Toulouse, France: IEEE Computer Society Press.

Reichert, M., Dadam, P., & Bauer, T. (2003). Dealing with forward and backward jumps in workflow management systems. *Software and Systems Modeling*, *2*(1), 37–58.

Reichert, M., Rinderle, S., Kreher, U., & Dadam, P. (2005). Adaptive process management with ADEPT2. In *Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, (pp. 1113–1114). Tokyo, Japan: IEEE Computer Society Press.

Rinderle, S., Reichert, M., & Dadam, P. (2004). Correctness criteria for dynamic changes in workflow systems: a survey. *Data and Knowledge Engineering*, *50*(1), 9–34.

Rinderle, S., Weber, B., Reichert, M., & Wild, W. (2005). Integrating process learning and process evolution a semantics based approach. In W. van der Aalst, B. Benatallah, F. Casati, & F. Curbera (Eds.) *Proceedings of the 3rd International Conference on Business Process Management (BPM'05)*, vol. 3649 of *Lecture Notes in Computer Science*, (pp. 252–267). Nancy, France: Springer Verlag.

Russell, N., van der Aalst, W., & ter Hofstede, A. (2006). Workflow exception patterns. In E. Dubois, & K. Pohl (Eds.) *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006)*, (pp. 288–302). Luxembourg: Springer.

SAP (2006). SAP advanced workflow techniques. `https://www.sdn.sap.com/irj/servlet/prt/portal/prtroot/docs/library/uuid/82d03e23-0a01-0010-b482-dccfe1c877c4`. Accessed 17 March, 2008.

Scheffer, T. (1996). Algebraic foundation and improved methods of induction of ripple down rules. In *Proceedings of the 2nd Pacific Rim Workshop on Knowledge Acquisition*, (pp. 279–292). Sydney, Australia.

Stein, L. A. (1999). Challenging the computational metaphor: Implications for how we think. *Cybernetics and Systems*, *30*(6).

TIBCO (2006). TIBCO iProcess Suite whitepaper. `http://www.staffware.com/resources/software/bpm/tibco_iprocess_suite_whitepaper.pdf`. Accessed 13 March, 2008.

Trewin, D. (2004). Televison, film and video production in Australia (publication 8679.0). Australian Bureau of Statistics `http://www.ausstats.abs.gov.au/ausstats/subscriber.nsf/0/14F1A528655E8486CA256EDE00782780/File/86790_2002-03.pdf`. Accessed 13 April, 2008.

Turing, A. (1936). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, *2*(42), 230–265.

Weber, B., Wild, W., & Breu, R. (2004). CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In P. Funk, & P. A. González Calero (Eds.) *Proceedings of the 7th European Conference for Adavances in Case Based Reasoning (ECCBR'04)*, vol. 3155 of *Lecture Notes In Computer Science*, (pp. 434–448). Madrid, Spain: Springer.

Workflow Management Coalition (2002). Introduction to workflow. `http://www.wfmc.org/introduction_to_workflow.pdf`. Accessed 14 November 2004.

Yan, J., Yang, Y., & Raikundalla, G. (2004). Towards incompletely specified process support in SwinDeW - a peer-to-peer based workflow system. In W. Shen, Z. Lin, J. Barthès, & T. Li (Eds.) *Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2004)*, vol. 3168 of *Lecture Notes in Computer Science*, (pp. 328–338). Xiamen, China: Springer.

Key Terms

Activity Theory. A meta-model or framework used to describe, theorise and research organised human activities, originating from Soviet cultural-historical psychology in the 1920's.

Exlet. An exception handling process, consisting of a number of exception handling primitives such as Suspend WorkItem, Remove Case, Compensate, and so on, which defines what action should be taken in the event of an exception of a certain type and context.

Process-Aware Information System (PAIS). A software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models (Dumas et al., 2005, 7).

Ripple-Down Rules (RDR). A hierarchical, extensible set of rules of the form "if *condition* then *conclusion*", together with cornerstone case data, conceptually arranged in a binary tree structure.

Service-Oriented Architecture. A software architecture consisting of a number of discrete (usually web-based) services (software components that are accessed or communicate via standard network protocols), that link together as required in order to achieve some task.

Worklet. A (usually) small, self-contained, complete process definition which is designed to be invoked as a substitute for one specific task in a larger, composite process. Each worklet is a complete extended workflow net (EWF-net) compliant with Definition 1 of the YAWL semantics. A set of zero or more worklets may form the *repertoire* of a task.