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. This paper presents the basis of an approach for dynamic flexibility, evolution and exception handling in workflows through the support of flexible work practices, 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*, an extensible repertoire of self-contained sub-processes that can be applied in a variety of situations depending on the context of the particular work instance.

1 Introduction

Workflow systems are used to configure and control structured business processes from which well-defined workflow models and instances can be derived [1, 2]. However, the proprietary 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 [3]; and (ii) deviations from the process model at runtime [4].

These limitations mean a large subset of business processes do not easily map to the rigid modelling structures provided [5], due to the lack of flexibility inherent in a framework that, by definition, imposes rigidity. Process models are 'system-centric', or *straight-jacketed* [6] 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 workflow systems by developing an approach based on natural work practices.

This paper introduces the concept of *'worklets'*, an extensible repertoire of self-contained sub-processes and associated selection and exception handling rules, and grounded in a formal set of work practice principles called *Activity Theory*, to support the modelling, analysis and enactment of business processes. It is organised as follows: Section 2 surveys issues of dynamic evolution and exception handling in workflows,

provides a brief overview of Activity Theory, then introduces the worklet paradigm. Section 3 describes how the worklet approach utilises *Ripple Down Rules* to achieve contextual, dynamic selection at runtime.

2 Achieving Flexibility through Worklets

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 [7]. (A full exploration of Activity Theory can be found in: [8, 9]). In [10], 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 (i.e. work processes) 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 natural 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 natural process evolution.

Each task of a process instance is linked to a extensible 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. Indeed, a top-level or manager worklet is developed that captures the entire workflow at a macro level. From that manager process, worklets are contextually selected and invoked from the repertoire of each task.

In addition, for each 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. Further, worklets to handle these potential events are constructed in *exactly the same way* as those for standard processes. 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

In order to realise the worklet approach, the situated contextual factors relevant to each case instance are required to be quantified and recorded [11] so that the appropriate worklet can be 'intelligently' selected 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 a *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. Process execution states 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 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, the runtime selection of which relies on the relevant context of each case instance, derived from case and historical data. The 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 [12].

The fundamental feature of RDR is that it avoids the difficulties inherent in attempting to compile, *a-priori*, a systematic understanding, organisation and assembly of all knowledge in a particular domain. Instead, it allows for general rules to be defined first with refinements added later as the need arises [13].

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. 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 [14]. 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. This tree traversal provides implied *locality* - a rule on an exception branch is tested for applicability only if its parent (next-general) rule is also applicable.

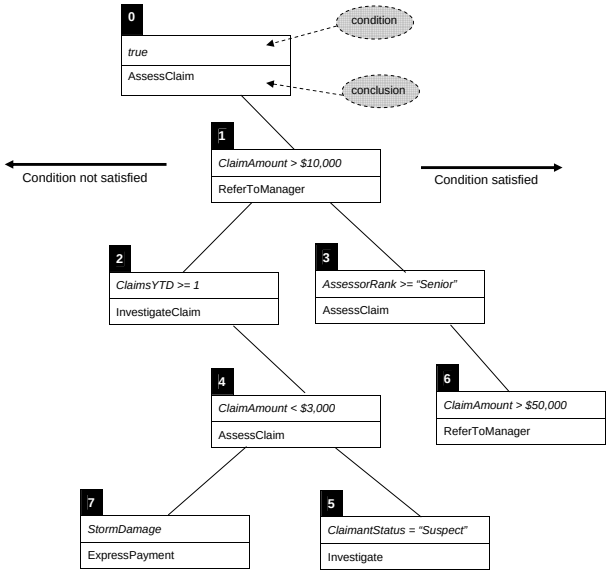

Fig. 1. Conceptual Structure of a Ripple Down Rule (*Assess Claim* Example)

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. In essence, each added exception rule is a refinement of its parent rule.

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 of the returned conclusion 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.

The Selection Process. When a process model is created, it is stored as a template of 'placeholders'. 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 consist of a single node containing a default rule and a conclusion referencing the one worklet defined. 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.

Suppose that, after a while, a new business rule is formulated. In conventional workflow 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, negating the need to explicitly model the choices and repeatedly update the model (with each iteration increasingly camouflaging the original business logic).

In all future case instances, the new worklet defined would be chosen for that placeholder if the condition defined by choosing the attribute differences occur in that instance's case data. Over time, the RDR chain for the placeholder grows as refinements are added to the rule base (fig. 1).

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 active worklets. Each worklet has a second, separate RDR rule base for exceptions, which is interrogated when a message is received. If the default condition in an exception RDR chain is an identifier for the message received, then the worklet will handle that exception by invoking an appropriately selected exception handling worklet (which may modify the current state of its parent worklet when activated (e.g. suspend it), and again on completion, as required). If there is no RDR defined for that exception, it is simply ignored.

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. unmet thresholds, missing data etc.); and (c) external triggers (i.e. user interactions). Exception handling is passed hierarchically through the execution tree to each parent worklet 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.

References

1. W.M.P. van der Aalst, Mathias Weske, and Dolf Grünbauer. Case handling: A new paradigm for business process support. *Data & Knowledge Engineering*, 53(2):129–162, 2005.
2. Gregor Joeris. Defining flexible workflow execution behaviors. In Peter Dadam and Manfred Reichert, editors, *Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications*, volume 24 of *CEUR Workshop Proceedings*, Paderborn, Germany, October 1999.
3. Alex Borgida and Takahiro Murata. 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)*, pages 59–68, San Francisco, CA, February 1999. ACM Press.
4. Fabio Casati. A discussion on approaches to handling exceptions in workflows. In *CSCW Workshop on Adaptive Workflow Systems*, Seattle, USA, November 1998.
5. 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.
6. 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*, pages 42–51, New York, 2001. ACM Press.
7. Bonnie A. Nardi. *Activity Theory and Human-Computer Interaction*, pages 7–16. In Nardi [9], 1996.
8. Y. Engestrom. *Learning by Expanding: An Activity-Theoretical Approach to Developmental Research*. Orienta-Konsultit, Helsinki, 1987.
9. Bonnie A. Nardi, editor. *Context and Consciousness: Activity Theory and Human-Computer Interaction*. MIT Press, Cambridge, Massachusetts, 1996.
10. 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.
11. Debbie Richards. Combining cases and rules to provide contextualised knowledge based systems. In *Modeling and Using Context, Third International and Interdisciplinary Conference, CONTEXT 2001*, volume 2116 of *Lecture Notes in Artifical Intelligence*, pages 465–469, Dundee, UK, July 2001. Springer-Verlag, Berlin.
12. P. Compton and B. Jansen. Knowledge in context: A strategy for expert system maintenance. In J.Siekmann, editor, *Proceedings of the 2nd Australian Joint Artificial Intelligence Conference*, volume 406 of *Lecture Notes in Artificial Intelligence*, pages 292–306, Adelaide, Australia, November 1988. Springer-Verlag.
13. Tobias Scheffer. Algebraic foundation and improved methods of induction of ripple down rules. In *Procceedings of the Pacific Rim Workshop on Knowledge Acquisition*, Sydney, Australia, 1996.
14. 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.