Introspection into an agile workflow engine for long-term processes – tool demonstration –

Mirjam Minor, Daniel Schmalen, Jakob Weidlich
Business Information Systems II
University of Trier
54286 Trier, Germany
{minor,schmalen,weid4701}@uni-trier.de

Andreas Koldehoff
Silicon Image GmbH
Garbsener Landstr. 10
30419 Hannover, Germany
andreas.koldehoff@siliconimage.com

Abstract

This demo paper is on an introspection tool for an agile workflow engine that executes long-term workflows. Agile workflows are workflows whose control flow structure can be adapted at run-time. We will present the architecture of the overall agile workflow management system, briefly introduce the agile workflow modelling language, and describe the tool for the introspection into the representation and execution of the workflows.

1 Introduction

The implementation of a workflow management system that can execute agile workflows is a challenging task as the workflows can dynamically evolve while being executed. The implementation must ensure that the workflow management system can handle structural adaptations of the ongoing workflows at unforeseen, future points. This includes that any workflow describes a consistent, executable control flow still after the modifications, that the already completed parts of the workflows are recorded correctly, and that the system is able to deal with several users modifying and executing workflows in parallel. In contrast to agile workflow technology, variant and exception handling approaches like opaque tokens in Abstract BPEL [9], pockets of flexibility [14], worklets [1], and experience-based exception handling [7, 6] allow structural adaptations only at previously (at build-time) specified points of the control flow. This requires different solutions to the above mentioned problems. Emergent workflows [3, 2, 8, 5] denote the other extreme of flexible workflows. These declarative approaches learn workflows from sets of loosely coupled tasks. Agile workflows lie in between those very structural and the declarative approaches on the other hand as they rely on structured processes but allow the incremental modelling as well as structural ad-hoc changes. In fact, few agile workflow management systems (WFMS) have been implemented, for instance those described in [13, 15, 16, 4]. An introspection tool that gives insight into the system-internal states of all ongoing workflows is a valuable aid for both, system developers and administrative process-owners. However, the structural adaptations of the workflows should not be made with the introspection tool but by means of a separate, graphical modelling tool. The introspection is for debugging and technical monitoring issues.

This demonstration paper on an introspection tool for the execution of agile workflows contributes to the development of the CAKE system (CAKE stands for collaborative, agile knowledge engine) whose methods have been introduced in previous work [11, 12]. Roughly spoken, CAKE is for the management of agile, long-term workflows, i.e. workflows that take several weeks or months of time. CAKE supports late-planning and ad-hoc changes of the ongoing workflow instances that have been derived from the workflow definitions (templates). A suspension mechanism allows the workflow designers to modify parts of a workflow while the remainder of the workflow can continue to be executed. Change reuse is supported by means of a case-based reasoning approach.

The remainder of this paper is organized as follows: In Section 2, we describe the role of the introspection tool within the overall CAKE architecture. Section 3 gives a brief introduction into the modelling language that is the basis for our agile workflow technology. In Section 4, we present the novel introspection concept for agile workflows. Section 5 concludes the paper with a summary.

2 Overall architecture

The core component of the CAKE system is the agile WFMS. Its architecture is depicted in Figure 1. The WFMS consists of three parts: the user interfaces, the workflow en-

actment service with the underlying data access layer, and the test engine.

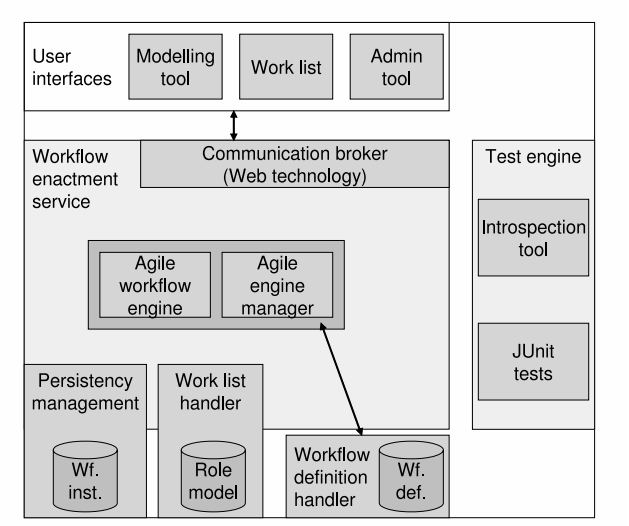

Figure 1. Architecture of the WFMS.

The user interfaces are the following:

- The modelling tool provides a graphical user interface to create and adapt agile workflows.
- The work list shows the tasks that have been assigned to a particular user and notifies the workflow enactment service when a task has been finished.
- The admin tool is for administrative purposes like restarting the workflow enactment service.

The workflow enactment service consists of the following sub-components:

- The communication broker uses Web technology for bi-directional message transfer.
- The core of the workflow enactment service consists of an agile workflow engine and an engine manager. They control the execution and adaptation of the agile workflows.
- The persistency management cares for the consistent and persistent storage of the workflow instances at runtime.
- The work list handler manages a role model.
- The workflow definition handler provides workflow templates.

The test engine for the left hand side of the architecture contains the following sub-components:

- The introspection tool monitors the internal execution and control flow of the agile workflow engine.
- The JUnit tests are for further testing activities.

The introspection tool as a part of the test engine plays an important role for the further development of the other system components that exist in prototypical implementations at the moment. The developers may just have a look at the current structure and execution state of a workflow instance. Furthermore, they can trigger off the further execution of an instance by taking over the role of a work list. And they can even modify the structure of a workflow instance by taking over the role of a simplified, non-graphical modelling tool. The introspection tool may benefit the whole prototypical operation phase in future. Administrative process-owners may employ it for technical monitoring purposes, for instance for solving conflicts that may occur unexpectedly due to a crashed work list in a straightforward way. In this case, the introspection tool is taking over the role of a simplified admin tool.

3 Modelling language for agile workflows

We have specified a control-flow-oriented workflow modeling language for agile workflows. The language has the five basic control flow elements sequence, AND-split, AND-join, XOR-split, and XOR-join, and also loops. We regard loops as structured cycles with one entry point to the loop (the control flow element LOOP-join) and one exit point from the loop (the control flow element LOOP-split). A diamond with an 'L', one incoming and several outgoing arrows with conditions in squared brackets stands for the LOOP-split; a diamond with an 'L]', several incoming and one outgoing arrows stand for the LOOP-join.

Figure 2 shows a clipping of a workflow instance from the chip design domain. It describes the workflow for the implementation of a chip module in hardware description language (HDL).

For adaptability reasons, we have created two more control flow elements: breakpoints and placeholder tasks for sub-workflows. Breakpoints are necessary for the control of modifications in a workflow instance concurrently to the execution. Setting a breakpoint prevents the workflow engine from overrunning tasks that are about to be modified. Placeholder tasks for sub-workflows stand for a reference to another workflow instance that is enacted when the control flow reaches the placeholder. For further details on the modeling language, we refer to the literature [12].

The agility of the instances is restricted by three constraints: The control flow elements form the following building blocks in the process models: sequence-blocks, AND-blocks, XOR-blocks and LOOP-blocks. (A) Building blocks cannot be interleaved but they can be nested. (B)

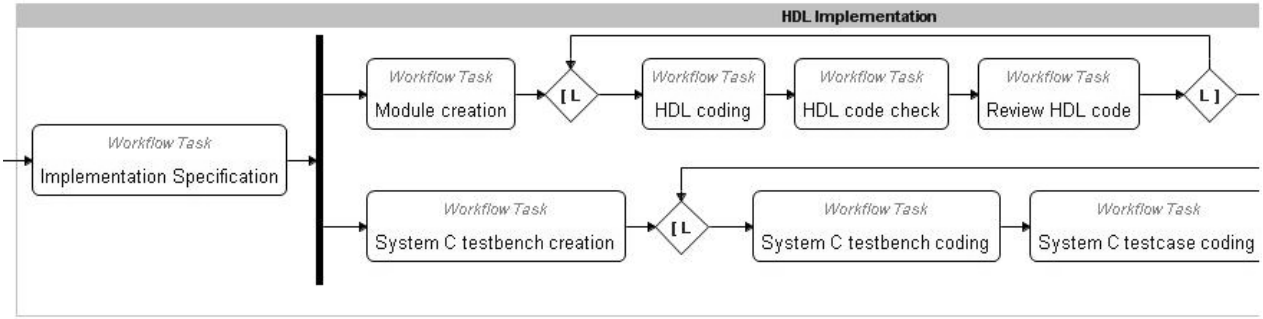

Figure 2. Sample workflow clipping at user level.

Before a structural adaptation can take place the concerned area of the workflow instance must be suspended from execution by means of a breakpoint. (C) Breakpoints are not allowed for 'the past', i.e. in areas of the workflow that have already completed their execution. A workflow instance is well-formed if it complies with the constraints (A) to (C).

4 Introspection into workflow representation and execution

The workflow modeling language facilitates the agility within workflows. This leads to new requirements for the workflow execution that can not be met by traditional workflow enactment services. The two additional control flow elements introduced for breakpoints and for sub-workflows need to be handled. Furthermore, the loop blocks require a special treatment since an adaptation of an ongoing loop may lead to different iterations of the same loop. In the following, the concept of states, a tree structure for the processing of the workflows, and the concept of master copies for the loops are introduced.

The state of processing are stored for all building blocks and tasks of a workflow instance. The following states are employed for processing: READY – the default state for new workflow elements, ACTIVE – currently being executed, COMPLETED – has been executed successfully, FAILED – has been executed unsuccessfully, SKIPPED – has been left out manually, OMITTED – lies in an inactive branch of an XOR, BLOCKED – from a sub-workflow placeholder whose sub-workflow has been suspended, and SUSPENDED – is within the scope of a breakpoint. BLOCKED is a special case of SUSPENDED as the workflow area behind a blocked placeholder tasks is suspended for execution but not accessible for modifications as far as the user has not set an additional breakpoint at this workflow's level.

The building block concept allows representing the workflow instances by means of a tree-oriented data structure. The tree consists of different node types for the tasks, the four building blocks, the placeholder tasks for sub-workflows, the milestones, and the breakpoints. The execution logics of the workflow elements including the states of processing are annotated to the particular node types. Due to this encapsulation, the tree concept is scalable in case the process modeling language is extended, for instance, with new control flow elements for OR-blocks.

Figure 3 shows a snap-shot of the introspection tool for the agile workflow engine that gives insight into the internal representation and execution of the sample workflow instance snippet from Figure 2. The task "Implementation specification" is currently ACTIVE and can be set to COMPLETED by hand. The right hand side of the figure shows a clipping of the representation of the workflow instance in XML (see [10] for a description).

Loops are prepared for agility with the new concept of master copies. The workflow elements that belong to a loop-block are copied during their processing. The master copies form a new sub-tree that is gradually inserted as a sibling of the original sub-tree for the loop-block. In the second and following iterations, the procedure of copying is repeated and leads to the creation of further sub-trees. The current master copies are accessible for the setting of breakpoints and for structural modifications. Modifications are valid for all future iterations but do not affect the past. Nested loops can be modified with restrictions only; due to space limitations, this is not able to be elucidated here in detail.

Figure 4 depicts a snap-shot from the sample workflow instance of Figure 3 where the execution has continued and reached the second iteration of the first loop ("HDL Loop") meanwhile. A second and a partial third iteration of the loop have been inserted into the tree representation. As the task "HDL coding" is currently active in the second iteration, the tasks "HDL code check" and "Review HDL code" in the second iteration and the task "HDL coding" in the third iteration are master copies.

The agile workflow engine implements an event-based execution of the control flow. A system-external event like

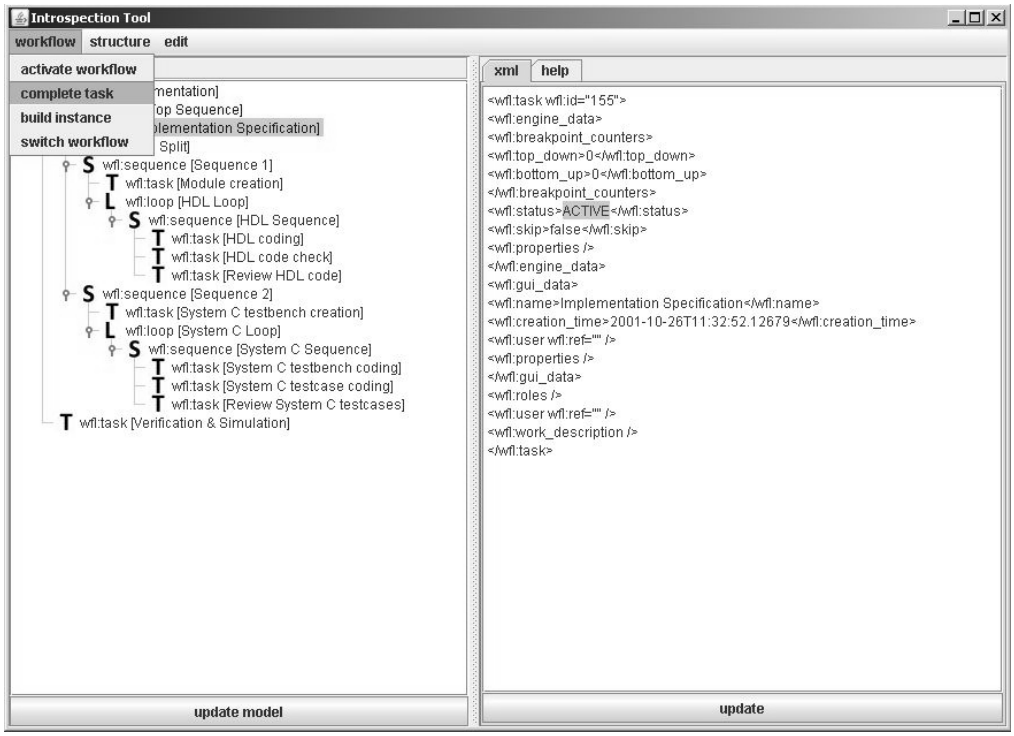

Figure 3. Completing a task manually at system level.

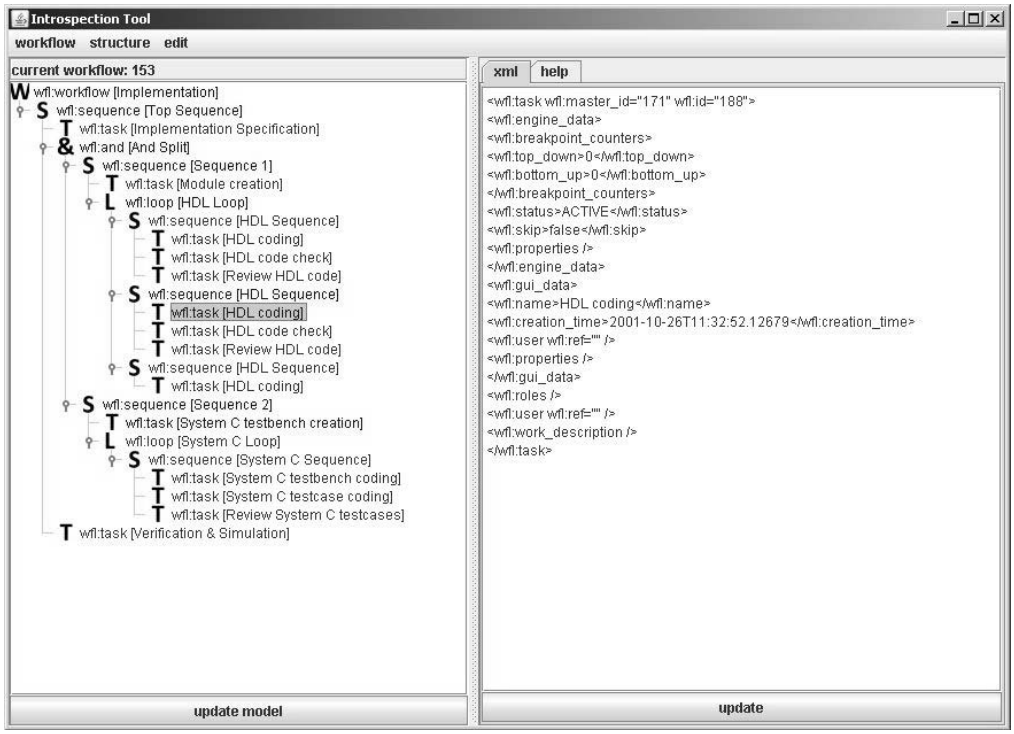

Figure 4. The internal master copies within an ongoing loop.

the completion of a task on a work list triggers a chain of system-internal events. Internal events are propagated through the workflow tree either towards the root or towards a leaf node. A node of the tree interprets incoming events in accordance with the node's execution logics. An 'ACTIVATE' event, for instance, that comes from the father node of a task node causes an outgoing 'ACTIVATED' event in the root direction. Some of the events are parametrized, for instance, a 'MOVE' event that is fired when an inner LOOP has been completed in order to move the sub-tree of master copies towards the outer loop. The events in a chain belong to the same act, e.g. to the handling of a new breakpoint. The chain is expanded incrementally. All member events of the same chain are partially ordered so that they cannot overtake each other. The introspection tool is a good means for the monitoring the effects of such chains of events.

5 Summary

We have demonstrated an introspection tool for an agile workflow engine that gives insight into the representation and execution state of a workflow instance at system level and simulates a simplified functionality of a work list, a workflow modelling tool, and an admin tool. It allows to continue the execution by manually completing the execution of tasks (work list) and to add or delete workflow elements including breakpoints (modelling tool). It allowsto create, start, and re-start workflow instances (admin tool). Such an introspection tool provides valuable support for the development and maturation of novel agile workflow technology.

6 Acknowledgment

The authors acknowledge the Federal Ministry for Education and Science (BMBF) for funding this work under grant number 01M3075. We acknowledge the assistance we have received from both Stefan Pipereit, Silicon Image and Marko Höpken, Silicon Image as well.

References

[1] M. Adams, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In R. Meersman and Z. Tari, editors, *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, GADA, and ODBASE 2006, Montpellier, France, October 29 - November 3, 2006. Proceedings, Part I*, volume 4275 of *Lecture Notes in Computer Science*, pages 291 – 308. Springer, 2006.

[2] S. Carlsen and H. D. Jørgensen. Emergent workflow: the ais workware demonstrator. In *Proceedings of the CSCW-98 Workshop: Towards Adaptive Workflow Systems, Seattle, WA, USA*, 1998.

[3] P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw. Freeflow: Mediating between representation and action in workflow systems. In *CSCW*, pages 190 – 198, 1996.

[4] M. Heller, A. Schleicher, and B. Westfechtel. Process evolution support in the ahead system. In J. L. Pfaltz, M. Nagl, and B. Böhlen, editors, *Applications of Graph Transformations with Industrial Relevance, Second International Workshop, AGTIVE 2003, Charlottesville, VA, USA, September 27 - October 1, 2003, Revised Selected and Invited Papers*, volume 3062 of *Lecture Notes in Computer Science*, pages 454–460. Springer, 2004.

[5] T. Herrmann. Lernendes workflow. In T. Herrmann, A. W. Scheer, and H. Weber, editors, *Verbesserung von Geschftsprozessen mit flexiblen Workflow-Management-Systemen*, pages 143 – 154. Physica-Verlag, Heidelberg, 2001.

[6] S.-Y. Hwang and J. Tang. Consulting past exceptions to facilitate workflow exception handling. *Decision Support Systems*, 37(1):49 – 69, 2004.

[7] Z. Luo, A. P. Sheth, K. Kochut, and I. B. Arpinar. Exception handling for conflict resolution in cross-organizational workflows. *Distributed and Parallel Databases*, 13(3):271 – 306, 2003.

[8] T. D. Meijler, H. Kessels, C. Vuijst, and R. leComte. Realising run-time adaptable workflow by means of reflection in the baan workflow engine. In *Proceedings of the CSCW-98 Workshop: Towards Adaptive Workflow Systems, Seattle, WA, USA*, 1998.

[9] R. Mietzner, Z. Ma, and F. Leymann. An algorithm for the validation of executable completions of an abstract bpel process. In M. Bichler, T. Hess, H. Krcmar, U. Lechner, F. Matthes, A. Picot, B. Speitkamp, and P. Wolf, editors, *Multikonferenz Wirtschaftsinformatik, MKWI 2008, München, 26.2.2008 - 28.2.2008, Proceedings*, pages 437 – 438. GITO-Verlag, Berlin, 2008.

[10] M. Minor, D. Schmalen, and R. Bergmann. Xml-based representation of agile workflows. In *Multikonferenz Wirtschaftsinformatik 2008 (MKWI 2008)*, pages 439–440. GITO-Verlag, 2008.

[11] M. Minor, D. Schmalen, A. Koldehoff, and R. Bergmann. Structural adaptation of workflows supported by a suspension mechanism stand by case-based reasoning. In *16th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE 2007), 18-20 June 2007, Paris, France*, pages 370–375, 2007.

[12] M. Minor, A. Tartakovski, D. Schmalen, and R. Bergmann. Agile workflow technology and case-based change reuse for long-term processes. *International Journal on Intelligent Information Technologies*, 4(1):80–98, 2008.

[13] M. Reichert and P. Dadam. $\text{Adept}_{\text{flex}}$-supporting dynamic changes of workflows without losing control. *J. Intell. Inf. Syst.*, 10(2):93–129, 1998.

[14] S. W. Sadiq, W. Sadiq, and M. E. Orlowska. Pockets of flexibility in workflow specification. In H. S. Kunii, S. Jajodia, and A. Sølvberg, editors, *Conceptual Modeling - ER 2001, 20th International Conference on Conceptual Modeling, Yokohama, Japan, November 27-30, 2001, Proceedings*, volume 2224 of *Lecture Notes in Computer Science*, pages 513 – 526. Springer, 2001.

[15] B. Weber, W. Wild, and R. Breu. Cbrflow: Enabling adaptive workflow management through conversational case-based reasoning. In P. Funk and P. A. González-Calero, editors, *Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings*, volume 3155 of *Lecture Notes in Computer Science*, pages 434–448. Springer, 2004.

[16] M. Weske. Workflow management systems: Formal foundation, conceptual design, implementation aspects. habil thesis, 2000. Fachbereich Mathematik und Informatik, Universität Münster.