XML-based Representation of Agile Workflows

Mirjam Minor, Daniel Schmalen, Ralph Bergmann

Business Information Systems II
University of Trier
Campus II
D-54286 Trier
[minor|schmalen|bergmann]@uni-trier.de

Abstract Agile workflow technology deals with business processes that require structural changes during run-time. For the difficult task of adapting the workflows, a modelling language that is easy to understand is essential. On the other hand, the workflow engine requires a workflow representation that can be processed, made persistent, and exchanged in a straight-forward way. XML provides a good means for the latter issues. In this paper, we present two representation forms for agile workflows that can be transformed automatically into each other. In the outlook, we discuss the potential of a partial transformation of the workflows into and from BPEL in order to integrate our agile workflow system within an enterprise application integration (EAI) scenario.

1 Introduction

„We could not apply a workflow system that is not adaptable to changes.“ states a chip design expert from Silicon Image GmbH, Hannover (S. Rackow, personal interview, October 25, 2006). Chip designers are used to dealing with the dynamics that result from the evolution of technology and from market changes as well. The increasing dynamics of the workflow is a phenomenon that affects the production processes within the high-tech industry: Software developers have to be flexible when customer requirements are changing. Healthcare professionals must react to side-effects and to other complications during the treatment of patients. What these examples from various domains have in common is that they cause major deviations from the usual business processes at run time. Furthermore, the ongoing processes need refinement after several weeks or months of running. Workflow technology supports business processes [WMC, p. 5]. However, traditional workflow management systems, those described in [LR00], are not able to deal with adaptable processes. Consequently, there is a need for *agile workflow technology* [WW05, p. 412], i.e. a workflow technology that allows the late-modelling and structural adaptation of ongoing workflows. The modelling languages and the processing engines for agile workflows require new concepts that can not be transformed one-to-one into standard Business Process Management (BPM) approaches. On the other hand, the employment of proprietary agile workflow management systems within an Enterprise Application Integration (EAI) approach requires the exchange of information. XML is a good means to balance both, expressiveness and exchangeability, of a description language for agile workflows. However, employing a proprietary XML schema dedicated to agile workflows requires a more sophisticated exchange mechanism on top of XML that filters out the agile parts and transforms the remainder into a BPM standard like EPCs [KNS92] or BPEL [Me06]. This paper addresses the graph-based workflow modelling language at the user level and the underlying XML application at the system level, which both are focussed on very large, agile process graphs. Furthermore, we discuss the partial transformation of the XML-based representation into and from BPEL.

In Section 2, we will give a brief introduction to agile workflow technology. In Section 3, we will describe our graph-based modelling langage which includes new workflow elements that address the adaptability of ongoing workflow instances. In Section 4, we explain the XML representation of the workflows which is interpreted by our flexible workflow management system. Finally, we discuss our approach and the future work on a transformation of the XML-based representation into and from BPEL.

2 Agile Workflows

Agile workflow technology allows the adaptation of ongoing workflows. The changes may apply to the process templates called *workflow definitions* as well as to the *workflow instances* that are derived from the workflow definitions. The agile workflow technology can be classified according to three types of structural process changes that can be accomplished at run time:

- Ad-hoc changes of individual workflow instances [RRD03, WWB04, Ba04]
- Modifications to a workflow definition that is already in use by instances ("schema evolution") [Ca98, We99, RRD03]
- Late-modelling and hierarchical decomposition [vE03, FMS05].

Typical ad-hoc changes are, for instance to re-order some parts of a workflow instance or to insert an additional task. Modifying workflow definitions in the same way leads to the difficult migration task of ongoing instances to the new schema. Late-modelling and hierarchical decomposition mean that a workflow instance can be enacted while it is supposed to be refined later on. Our work fits in the first and third classifications. We support the agility by a suspension mechanism using breakpoints (see Section 3). The ad-hoc changes require a special loop handling that separates future iterations from the presence (see Section 4). The late-modelling and hierarchical decomposition is enabled by hierarchical process models (see Section 3). Fig. 1 depicts a sample workflow definition from the chip design domain. Workflow instances for chip design projects can be derived from this template. During execution of the task 'Project planning', the workflow has to be refined, for instance by replacing the 'Dummy design unit' with the real chip design units, i.e. with the modules that are to be designed. Ad-hoc changes may be applied later on, for instance an additional task 'Update implementation specification' of a particular design unit when a change request concerning the implementation in hardware desciption language (HDL) has occurred (see the sample workflow instance for a design unit in Fig. 2). An XML-based exchange format for such agile workflows is a first step towards the integration of agile workflow technology with legacy workflow systems that do not support adaptability.

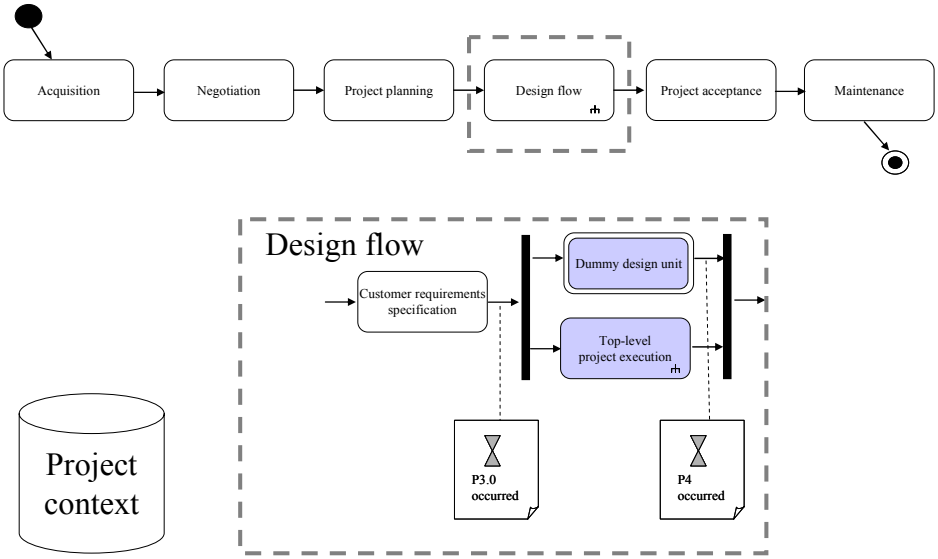

Figure 1: Sample workflow definition for a chip design process.

3 Graph-structured modelling language

We have specified a proprietary process modelling language for the agile workflows by extending the UML activity diagram notation by own symbols that address the adaptability of workflows. The concepts of the language are based on the notion of workflow patterns introduced by van der Aalst et al. [vA03]. Workflow patterns "address business requirements in an imperative workflow style expression" [vA03, p. 4]; broadly speaking, they are useful routing constructs within workflows. In terms of van Aalst et al, our process modelling language has the five basic *control flow elements* (workflow patterns) 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 (see Fig. 2). For adaptability reasons, we have created two more control flow elements:

(1) breakpoints are symbolized by stop signs on connections (see Fig. 2);

(2) placeholder tasks for sub-workflows are depicted as rounded boxes with double borders (see again 'Dummy design unit' in Fig. 1);

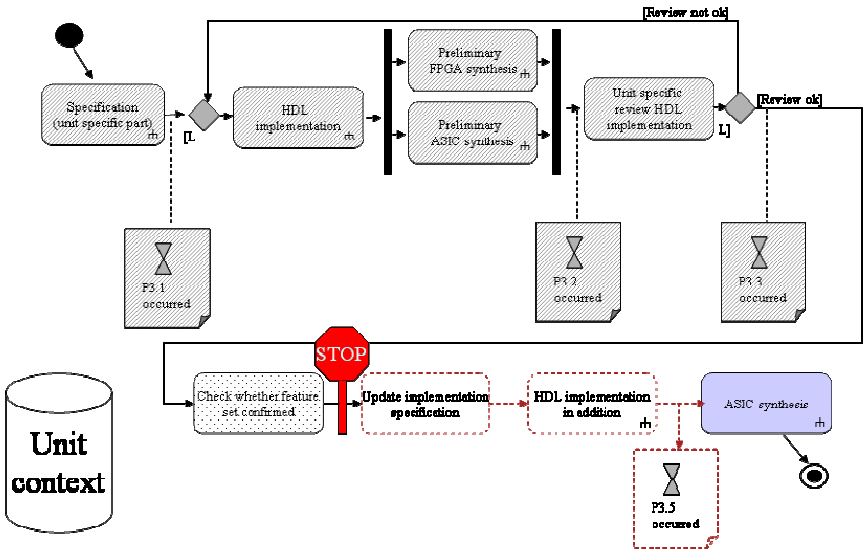

Figure 2: Sample workflow instance for developing a design unit.

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. This new control flow element particularly contributes to the systems capability for late-modelling and hierarchical decomposition.

In addition to the nine control flow elements, our process modelling language consists of the usual workflow elements such as tasks, start symbols, and end symbols as well as two new elements dedicated to modelling and monitoring purposes only:

(3) placeholder tasks for sub-diagrams are marked by a fork symbol (see the placeholder task for 'Design flow' in Fig. 1);

(4) milestones are depicted as comments including a sand glass (see the milestone 'P3.0' in Fig. 1).

Sub-diagrams have only been introduced for clarity reasons. They allow the user to decompose large workflows into parts which are displayed hierarchically. In contrast to sub-workflows, sub-diagrams neither have their own context data nor an own workflow engine. The contents of a sub-diagram are just processed instead of their placeholder. Milestones are a special kind of comments that encapsulate a certain date for a couple of workflow instances, e.g. for the sub-workflows that describe the design of different modules of a chip in parallel. At the moment, neither placeholder tasks for sub-diagrams nor milestones do affect the control flow. Nevertheless, we have included the milestones into the XML representation (see below) in order to facilitate an alternative interpretation of milestones in future.

For reasons of clarity and manageability, we restricted the agility of the process model 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. For example, in Fig. 2, the AND-block with 'Preliminary FPGA synthesis' and 'Preliminary ASIC synthesis' is an inner block of a sequence which belongs to the outer LOOP-block (synthesis is the chip design step that transforms the HDL code into an actual chip layout). (B) 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. In Figure 2, the completion of workflow elements is indicated by the hatching; the dotted task 'Check whether feature set confirmed' is currently active. A process model in our workflow modelling language is *well-formed* if it complies with the constraints (A) to (C).

We have implemented a modelling GUI in Java that makes use of the Eclipse Rich Client Platform (Eclipse RCP) [Da01] and of the Graphical Editing Framework (GEF) [MD+04]. The GUI ensures that constraint (A) is fulfilled as it provides the splits and joins as inseparable pairs. In the current revision of the implementation, the constraints (B) and (C) have to be kept under surveillance by the users.

4 XML representation of agile workflows

The graph-structured representation of agile workflows that we have introduced in the previous section can be processed directly by a workflow engine. However, this is not the best solution as it requires complicated data structures for the control flow information like an adjacency matrix or adjacency lists. A more suitable solution is a tree-oriented representation of the workflow instances based on the building-block property of the modelling language and based on XML technology. The XML representation has the additional benefit that it can be used for persistency and as a basis for information exchange mechanisms.

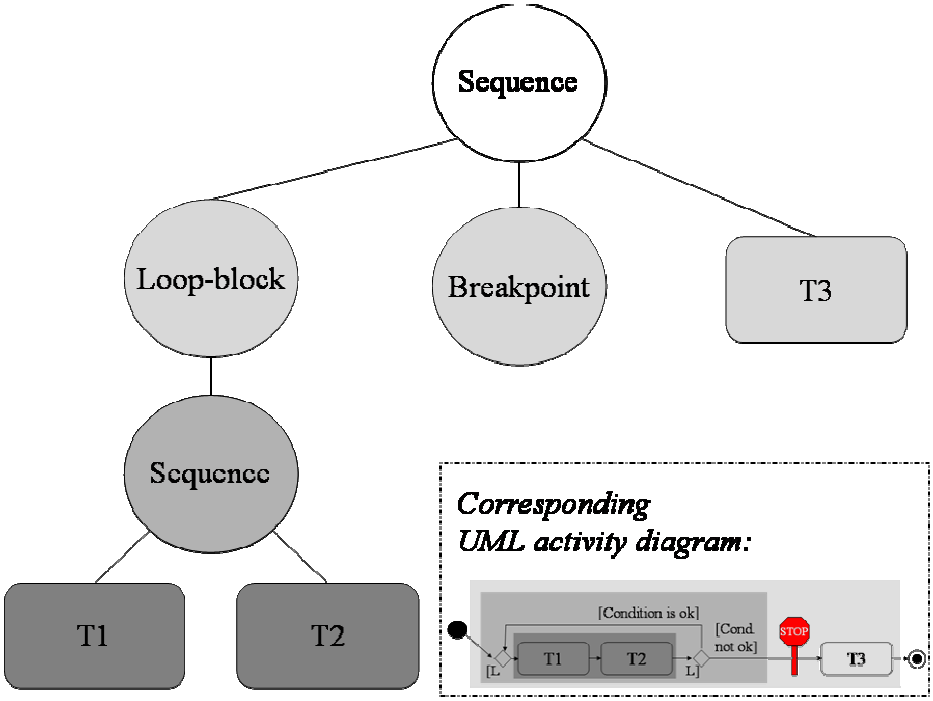

Figure 3: Block-oriented tree representation of a sample workflow.

The modelling GUI transforms each workflow instance automatically into a tree that can be exported into an XML file. The *workflow tree* is consisting of different node types for the tasks, the four building blocks, the placeholder tasks for sub-workflows, the milestones, and the breakpoints. Fig. 3 shows an abstract tree representation of a very simple sample workflow instance. It has three building blocks: an outer sequence (light grey), a loop block (middle grey), and an inner sequence (dark grey). The tasks 'T1' to 'T3' and the breakpoint form the leaf nodes. The execution logics of the workflow elements including the state of processing (READY, ACTIVE, COMPLETED, SUSPENDED etc.) are annotated to the particular node types. Due to this encapsulation, the tree concept is scalable in case the process modelling language will be extended, for instance, with new control flow elements for OR-blocks.

```
1 <xml version="1.0" encoding="UTF-8" ?>
2 - <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3     targetNamespace="cake.ltworkflow.core.workflow"
4     xmlns="cake.ltworkflow.core.workflow" elementFormDefault="qualified"
5     attributeFormDefault= "qualified">
6   <xsd:element name="workflow" type="WorkflowType" />
7   …
8 - <xsd:group name="SequenceGroup">
9   - <xsd:choice>
10        <xsd:element name="sequence" type="SequenceType" />
11    </xsd:choice>
12  </xsd:group>
13        <!-- all workflow tree nodes except for sequence: -->
14 - <xsd:group name="BasicGroup">
15  - <xsd:choice>
16        <xsd:element name="and" type="AndType" />
17        <xsd:element name="xor" type="XORType" />
18        <xsd:element name="loop" type="LoopType" />
19        <xsd:element name="task" type="TaskType" />
20        <xsd:element   name="placeholdertask"   type="PlaceholderTaskType"
/>
21        <xsd:element name="breakpoint" type="BreakpointType" />
22        <xsd:element name="milestone" type="MilestoneType" />
23    </xsd:choice>
24  </xsd:group>
25  …
```

Figure 4: XML schema for the representation of a workflow tree.

The XML schema for the workflow tree defines an own namespace ‘cake.ltworkflow.core.workflow’ (see line 3 in Fig. 4). The acronym ‘cake’ stands for ‘collaborative, agile knowledge engine’ a framework that integrates agile workflow technology with knowledge management techniques like case-based reasoning (see [Mi07]). The XML schema defines elements for the workflow tree nodes: ‘sequence’ for the sequence block nodes (line 10 in Fig. 4), ‘and’, ‘xor’, and ‘loop’ for the further building block nodes (lines 16 - 18), and ‘task’ etc. for the singleton workflow elements (lines 19ff.). The corresponding type definitions (‘SequenceType’, ‘AndType’, etc.) make use of the groups ‘BasicGroup’ and ‘SequenceGroup’ to specify that sequences alter with basic items in the workflow tree. For example, the ‘LoopType’ contains sequences only (compare Fig. 6) while a sequence contains arbitrary ‘BasicGroup’ items.

```
30 ...
31 - <xsd:group name="DataGroup">
32   - <xsd:sequence>
33       <xsd:element name="engine_data" type="EngineDataType" />
34       <xsd:element name="gui_data" type="GUIDataType" />
35     </xsd:sequence>
36   </xsd:group>
37 ...
38 - <xsd:complexType name="EngineDataType">
39       ...
40       <xsd:element name="status" type="StatusType" />
41       ...
42   </xsd:complexType>
43 - <xsd:complexType name="GUIDataType">
44       ...
45       <xsd:element name="name" type="xsd:string" />
46       ...
47   </xsd:complexType>
48   ...
```

Figure 5: Snipplet of the XML schema with special types for engine- and gui-specific data.

The 'DataGroup' (see Fig. 5) is for engine-specific data ('engine_data') and for data that are required for the presentation of a workflow in the graphical user interface ('gui_data'). For instance, the workflow engine is interested in the processing state of the particular workflow elements (line 40 in Fig. 5), while the graphical user interface (gui) needs the names of the workflow elements (line 45) in order to display them to the users instead of the internal identification numbers. All nodes within the workflow tree are containing a 'DataGroup'.

Fig. 6 shows the type definition of a loop node in the workflow tree. The 'DataGroup' is followed by a 'condition' element that describes the exit condition of the loop (line 54), and the child nodes ('child_elements') within the workflow tree (lines 55ff.). The 'SequenceGroup' occurs zero times if the loop is empty (minOccurs="0" in line 58). Before the loop is active, it contains usually one sequence. During the processing of the loop, the workflow engine creates copies of all descendant nodes that are completed, and inserts them into a sibling of the sub-tree that roots in the original loop node. The master copies are required in case the user aims at setting a breakpoint and modifying the future iterations of the loop. For a more detailed description of the master copy mechanism, we refer to the literature [Mi07]. The further building block types are structured similarly to the loop type, except for the master copies. The execution logics of the workflow engine decides whether subsequent child elements have to be interpreted sequentially ('sequence'), in parallel ('and'), or as alternatives ('xor').

```
50 ...
51 - <xsd:complexType name="LoopType">
52  - <xsd:sequence>
53       <xsd:group ref="DataGroup" />
54       <xsd:element name="condition" type="LoopConditionType" />
55     - <xsd:element name="child_elements">
56         - <xsd:complexType>
57           - <xsd:sequece>
58               <xsd:group ref="SequenceGroup" minOccurs="0"
                                              maxOccurs= "unbounded" />
59             </xsd:sequence>
60           </xsd:complexType>
61         </xsd:element>
62    </xsd:sequence>
62   </xsd:complexType>
64    ...
```

Figure 6: Snipplet of the XML schema with the loop type.

The task type (see Fig. 7) contains a role element (line 74) in addition to the 'DataGroup' (line 73). The role specifies who is supposed to execute the task. Alternatively to an abstract role like a 'synthesis expert', a concrete user name may be specified (line 75). The further singleton workflow elements are structured quite self-explanatory: A placeholder task contains a reference to another workflow instance, i.e. the sub-workflow the placeholder stands for. The breakpoint type consists of a 'DataGroup' and some internal properties only, and the milestone type contains additionally a reference to a milestone description. Due to limitations of space, we omit the according snipplets of the XML schema.

The transformation of the graph-based representation into and from XML is implemented as a part of the GUI software. The import and export of the XML files works while the XML RPC-based communication mechanism with the workflow enactment service is still under development.

5 Summary and future work

We have presented two representation forms for agile workflows that can be transformed mutually without loss as one can see easily. A graph-based modelling language is for users (user level) while it's tree-based XML representation is for the workflow engine (system level). The XML schema provides a first step towards the exchange of workflow data with conventional workflow systems and other systems that deal with process information.

```
70 ...
71 - <xsd:complexType name="TaskType">
72    <xsd:sequence>
73       <xsd:group ref="DataGroup" />
74       <xsd:element name="roles" type="RolesType" />
75       <xsd:element name="user" type="ReferenceType" minOccurs="0"
/>
76        ...
77    </xsd:sequence>
78  </xsd:complexType>
79 ...
```

Figure 7: Snipplet of the XML schema with the task type.

Our future work includes the transformation of the XML representation into and from BPEL in order to integrate our agile workflow approach with an EAI scenario. This would provide us with a wide range of new capabilities, for instance, to import ongoing processes from non-agile BPEL engines in order to adapt them in a controlled way by means of our agile mechanisms (suspension, loop handling), or to have access to the visualisation functionality of other systems. The mapping of the non-agile parts of the workflows can be done by means of BPEL activities while the mapping of agile parts, for instance of structurally different iterations of a loop that have been executed, needs further research. Our first ideas for mapping the non-agile parts are: 'sequence' nodes become '<sequence>' activities, 'and' nodes become '<flow>' activities, 'xor' nodes become '<if>' and '<else>' activities ('<elseif>' is required only when an 'xor' has more than two alternative branches), 'loop' nodes become '<repeatUntil>' activities. The transformation of task nodes has two alternative solutions, as BPEL deals with Web services rather than human tasks: BPEL4People [Kl05] provides an additional 'People activity' that "is a basic activity, which is not implemented by a piece of software, but realized by an action performed by a human being" [Kl05, p. 12]. Unfortunately, "straight BPEL 2.0 engines won't be able to implement it" [Si06]. Silver proposes an alternative solution, namely to provide a task management service ("an out-of-the-box Web service that manages human tasks" [Si06]). Both solutions would be possible for our 'task' nodes. For a mapping of the placeholder tasks into BPEL, we aim to investigate BPEL-SPE [Si06] as it is an extension of BPEL for sub-processes. The milestones and the breakpoints can not be handled by standard BPM engines. Consequently, they can not be expressed by means of BPEL activities. The same holds for the gui- and engine-specific data as well as for the roles. It is an open issue to define the detailed transformation mechanisms and to decide how big the information loss from the agile workflow to the non-agile representation should be. A pseudo code similar to that introduced in [MLZ06] would be a helpful means for this difficult task and for the opposite direction of the transformation. Our first analysis has shown that a mapping of our agile workflows into and from BPEL in principle is not lossless but feasible.

References

[Ba04] Bassil, S.; Rudolf, K.; Keller, R. K.; Kropf, P.: A workflow-oriented system architecture for the management of container transportation. In (Desel, J., Pernici, B., Weske, M., Eds.): Business Process Management: Second International Conference, BPM 2004, Potsdam, Germany, June 17-18, 2004, Proceedings, LNCS 3080, Springer, Berlin, 2004, pp. 116 – 131.

[Ca98] Casati, F.; Ceri, S.; Pernici, B.; Pozzi, G.: Workflow evolution. Data & Knowledge Engineering, 24, 1998, pp. 211 – 238.

[Da01] Daum, B.: Rich-Client-Entwicklung mit Eclipse 3.1. dpunkt.verlag, Heidelberg, 1. Auflage, 2001.

[FMS05] Freßmann, A.; Maximini, K.; Sauer, T.: Towards collaborative agent-based knowledge support for time-critical and business-critical processes. In (Althoff, K. D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T., Eds.): Professional Knowledge Management: Third Biennial Conference, WM 2005, Kaiserslautern, Germany, April 10-13, 2005, Revised Selected Papers, LNAI 3782, Springer, Berlin, 2005, pp. 420 - 430.

[KNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)", in: Scheer, A.-W. (Hrsg.): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken 1992.

[Kl05] Kloppmann, M., et al.: WS-BPEL Extension for People – BPEL4People. Retrieved from the Web September 28, 2007. http://www-128.ibm.com/developerworks/library/specification/ws-bpel4people/

[LR00] Leymann, F.; Roller, D.: Production workflow - concepts and techniques. Prentice Hall International, Upper Saddle River, NJ, USA, 2000.

[MD+04] Moore, B.; Dean, D.; Gerber, A.; Wagenknecht, G.; Vanderheyden, P.: Eclipse Development Using the Graphical Editing Framework and the Eclipse Modeling Framework. IBM Redbook, 2004: Retrieved from the Web November 30, 2007. http://ibm.com/redbooks

[Me06] Mendling, J.: Business Process Execution Language for Web Service (BPEL). Aktuelles Schlagwort, EMISA Forum, Volume 26, Number 2, pages 5-8, August 2006.

[MLZ06] Mendling, J.; Lassen, K. B.; Zdun, U.: Transformation Strategies between Block-Oriented and Graph-Oriented Process Modelling Languages. In (Lehner, F., Nösekabel, H., Kleinschmidt, P., Eds.): Multikonferenz Wirtschaftsinformatik 2006 (MKWI 2006), Band 2, XML4BPM Track, GITO-Verlag Berlin, 2006, pp. 297 - 312.

[Mi07] Minor, M.; Tartakovski, A.; Schmalen, D.; Bergmann, R.: Agile Workflow Technology and Case-Based Change Reuse for Long-Term Processes. International Journal on Intelligent Information Technologies, to appear.

[RRD03] Reichert, M.; Rinderle, S.; Dadam, P.: ADEPT workflow management system: Flexible support for enterprise-wide business processes (tool presentation). In (van der Aalst, W. M. P., Ed.): International Conference on Business Process Management (BPM '03), Eindhoven, The Netherlands, June 2003. LNAI 2678, Springer, Berlin, 2003, pp. 370 – 379.

[Si06] Silver, B.: BPEL4People Revisited. Retrieved from the Web September 28, 2007. http://www.bpminstitute.org/articles/article/article/bpel4people-revisited.html

[vA03] van der Aalst, W. M. P.; ter Hofstede, A. H. M.; Kiepuszewski, B.; Barros, A. P.: Workflow patterns. Distributed and Parallel Databases, 14, 2003, pp. 5 – 51.

[vE03] van Elst, L., Aschoff, F. R., Bernardi, A., Maus, H., & Schwarz, S. (2003). Weakly-structured workflows for knowledge-intensive tasks: An experimental evaluation. In 12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises, June 2003, Linz, Austria, IEEE Computer Society, Los Alamitos, California, 2003, pp. 340 – 345.

[WWB04] Weber, B.; Wild, W.; Breu, R.: CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In (Funk, P., Gonzalez-Calero, P. A., Eds.): Advances in Case-Based Reasoning, 7th European Conference, ECCBR 2004, Madrid, Spain, August 30 - September 2, 2004, Proceedings. LNCS 3155, Springer, Berlin, 2004, pp. 434 – 448.

[WW05] Weber, B.; Wild, W.: Towards the agile management of business processes. In (Althoff, K. D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T., Eds.): Professional Knowledge Management, Third Biennial Conference, WM 2005, Kaiserslautern, Germany, April 10-13, 2005, Revised Selected Papers, LNCS 3782, Springer, Berlin, 2005, pp. 409 – 419.

[We99] Weske, M.: Workflow management systems: Formal foundation, conceptual design, implementation aspects, Habilitation's thesis, University of Münster, Germany, 1999.

[WMC] Workflow management coalition glossary & terminology. Retrieved from the Web May 23, 2007. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf