Structural adaptation of workflows supported by a suspension mechanism and by case-based reasoning

Mirjam Minor, Daniel Schmalen, Andreas Koldehoff[*], Ralph Bergmann
University of Trier, Department of Business Information Systems II
54286 Trier, Germany
{minor, schmalen, bergmann}@uni-trier.de

[*]*Silicon Image GmbH, Garbsener Landstr. 10*
30419 Hannover, Germany
andreas.koldehoff@siliconimage.com

Abstract

Collaborative, long-term processes with changing requirements occur in several domains. In design and manufacturing, we observed the following particular characteristics: the process models can be huge and contain longsome decisions while the time to market is tight. Furthermore, different types of products have to be considered. In this paper, we present an approach of new, flexible workflow technology that provides a solution for this: The ongoing workflows can be adapted by means of late-planning and ad-hoc changes. A suspension mechanism allows the workflow designers to modify parts of a workflow while the remainder of the workflow can continue to be executed. A case-based reasoning approach supports the reuse of past experience for this.

1. Introduction

Th. Herrmann reports the observation that "many collaborative tasks in companies can be partly seen as recurrent routines but partly use to contain innovation. ... This phenomenon will increase with the dynamics of the market and its requirements to the flexibility of the company and to the individual customer care." [4, p. 145, own translation]. Traditional workflow systems are able to support the recurrent tasks quite well. In order to deal with the flexible, innovative part, the workflows have to be adaptable. Even more, in highly flexible domains like medicine or chip design situations occur where the ongoing workflows need to be changed. For instance, an alternative measure has to be taken when a certain therapy is not successful for a patient or when a certain algorithm does not work for a new chip technology. This may happen unexpectedly or may require a major adaptation of the ongoing process. Both of them can not be handled by traditional workflow systems.

In this paper, we present an approach of new, flexible workflow technology that has been driven by our URANOS project in cooperation with the chip design industry. We focus on issues of the workflows' representation, execution, suspension, and structural adaptation of ongoing workflows. A case-based reasoning (CBR) approach supports the users in reusing past experience with the adaptation of workflows for adapting current workflows.

In the literature, there are already approaches of flexible workflow technology that support the structural adaptation of ongoing workflows. Late-planning and hierarchical decomposition [10, 3] are one way to modify ongoing workflows. Another way is given by ad-hoc changes of the workflow structure [8, 11]. The ad-hoc changes of workflows have been applied successfully for the medical domain, for instance. However, we derived some crucial requirements from an analysis of the chip design domain that are not met by the existing approaches:

- Huge process models with thousands of tasks require that some parts of a workflow continue with the execution of tasks while other parts are being modified.
- Decisions may take some time with the consequence of delayed workflow modeling activities. As above, the workflow should not pause completely due to tight time constraints.
- The coexistence of different product types (i.e. chip types in our scenario) requires configurable contexts: The context of workflows has to be

compound individually while sharing the vocabulary for this with other workflows.

- Experience from the adaptation of workflows in the past should be reused for the adaptation of an ongoing workflow. Unlike the existing approaches, we cannot guarantee that we have additional information like context information [3, 10] or conversational knowledge [11] for this. Consequently, the retrieval of the past workflows should consider the workflows' structure directly rather than solely operating on additional information.

In order to meet these requirements, we have developed an approach of new, flexible workflow technology that we present as follows: In Section 2, we describe the basic, incremental modelling approach for workflows. Section 3 deals with the control flow part of the workflows, while Section 4 examines the context part. Within Section 3, both, the workflow language and a tree representation of workflows are designed in a way that enables the suspension of workflows. In Section 5, we introduce our authoring support approach by means of case-based reasoning. In Section 6, we discuss our approach and draw a conclusion.

2. Handling flexible workflows

Our workflow approach facilitates structural changes of the ongoing workflows. This allows an incremental modeling approach: Initial workflow instances are derived from a set of templates the so-called workflow definitions. The ongoing workflow instances can be modified by ad-hoc changes and by late-planning.

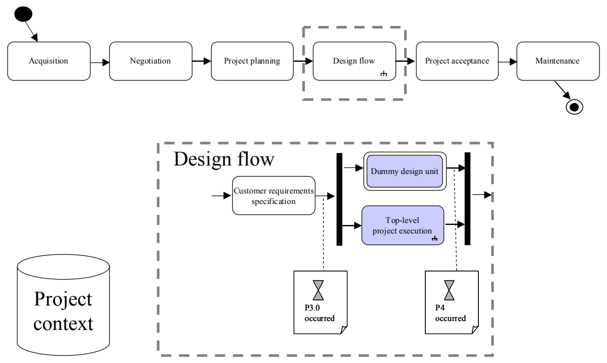
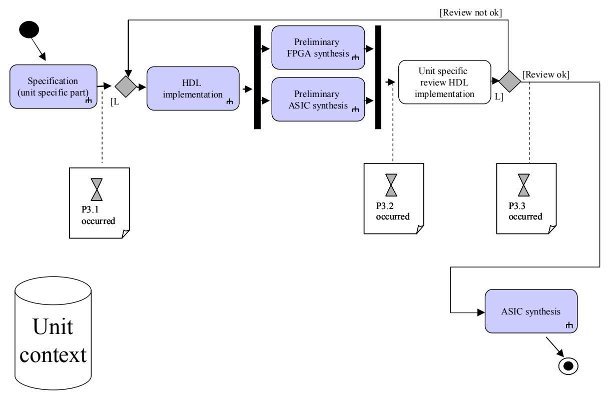


Figure 1. The workflow definition of a design project following SciWay 2.0.

Figure 1 shows an UML activity diagram of a sample workflow definition that we modeled for the chip design domain as template for new projects. Each workflow definition consists of a control flow structure of tasks and of a context model. The context model has the aim to document factors that have a significant impact on the workflows. The target frequency of the chip, for instance, is quite important for the design process: In case it is missed by a chip component, this has an impact on other components that may have to be redesigned. Of course, this affects also the control flow of the according workflow(s). The control flow structure follows the design flow 'SciWay 2.0', i.e. a standardized description of the step by step design process for all digital design projects of our industrial partner sci-worx.



[Review not ok]
Specification (unit specific part)
[L
HDL implementation
Preliminary FPGA synthesis
Preliminary ASIC synthesis
Unit specific review HDL implementation
L]
[Review ok]
P3.1 occurred
P3.2 occurred
P3.3 occurred
Unit context
ASIC synthesis

Figure 2. The workflow definition of a design unit following SciWay 2.0.

3. Control flow structure of workflows

The control flow structure is modeled in a well-defined workflow language that we describe in the following subsection. The execution uses an internal tree-representation that we present afterwards. For both, the modeling and the execution, we have designed special elements and mechanisms that prepare them for the suspension and adaptation of workflows.

Workflow modeling language

The workflow language that we used is based on the notation of workflow patterns introduced by van Aalst et al [9]. Our workflow modelling language consists of the five basic control flow elements (workflow patterns) sequence, AND-split, AND-join, XOR-split, and XOR-join as well as of loops. We regard loops as structured cycles with one entry point to the loop (LOOP-join) and one exit point from the loop (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 stands for the LOOP-join (see Figure 2). For reason of adaptability, we have extended this model by three own workflow elements:

- placeholder tasks for sub-workflows are depicted as rounded boxes with double borders (see *Dummy design unit* in Figure 1).
- placeholder tasks for sub-diagrams are marked by a fork symbol (see Top-level project execution in Figure 1);
- breakpoints are symbolized by stop signs (see Figure 6).

Sub-diagrams have only been invented for reasons of clarity. This hierarchical concept is only introduced in order to decompose large workflows into parts which can be easier shown in the GUIs. In opposite to sub-workflows, sub-diagrams do not have an own workflow engine nor an own context. Breakpoints are necessary for the control of modifications in a workflow instance. Setting a breakpoint prevents the workflow engine from overrunning tasks that are about to be modified.

Tree representation

The control flow elements form building blocks. For instance, the sequence of tasks between the LOOP-join and the LOOP-split in *HDL implementation in addition* in Figure 6 belong to a LOOP-block. Building blocks cannot be interleaved but they can be nested. For example, in Figure 2, the AND-block is an inner block of a sequence which belongs to the outer LOOP-block. The building block concept allows us to represent the workflow instances by means of a tree-oriented data structure. This structure can be processed in a quite straightforward implementation.

In the tree representation, the nodes represent the particular workflow elements (tasks, placeholder tasks, control flow elements or breakpoints) by different types of nodes. Each type of node encapsulates the execution logics of its workflow element, for instance the parallel execution of the children of an AND-node. We explain this further by means of a short example: Assume that we have derived a workflow instance called *Design unit XY* from the workflow definition in Figure 2 and, to simplify matters, replaced the sub-diagrams by plain tasks, e.g. the sub-diagram *ASIC synthesis* by the task *ASIC synthesis*. Additionally, we set a breakpoint between the LOOP-split and the *ASIC synthesis* task. This sample workflow instance is represented by the tree in Figure 3 as follows: The root node of the tree is labeled by the name of the (sub-) workflow (see the legend of Figure 3). Tasks are represented by leaves (see 1 to 6 in Figure 3). A sequence node (*S*) contains a set of elements that are executed in a serial way from the left to the right. Other control flow elements (*L* for loops, *A* for AND's, *X* for XOR's) have solely sequences as children that are processed according to the semantics of the parent node. Breakpoints (*B*) are represented as leave nodes of sequence nodes. The sample in Figure 3 contains a break point between the loop block and task six. The set of node types within the tree structure is easily extendible, for instance by a new type of node (O) for OR-blocks.

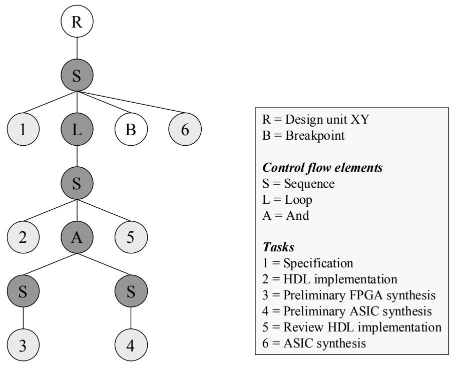


Figure 3. The workflow instance of the design unit XY.

For execution purposes, every node is enriched with engine relevant data such as the state of processing. Additionally, the task nodes contain particular task data, e.g. the roles of potential executors. The execution of the tree is realized with an event concept. Whenever the state of a workflow element changes the particular node fires an event to the children or parents according to its semantics. The receivers decide how to handle it, e.g. to propagate it or to fire further events to its children or parents. For instance, when the task 2 (*HDL implementation*) in Figure 3 is being completed, it fires an event to the parent S. This node generates a new triggering event and fires it to the AND node. The AND node triggers 3 (*Preliminary FPGA synthesis*) and 4 (*Preliminary ASIC synthesis*) via its children sequence nodes. Some events are able to cross the borders of workflow instances via the sub-workflow placeholder tasks. For example, this is necessary in case the error handling of a sub-workflow has failed.

In order to prepare the workflow instances for ad-hoc changes, we have invented the new concept of *master copies* for the handling of loops. With this concept, we are able to distinguish the past and the future parts of the workflow unambiguously. It is a crucial prerequisite for ad-hoc changes within a loop-block. We do not support modifications of particular iterations; all changes are valid for every future iteration. The master copies are created incrementally; each workflow element that has been activated or marked as omitted in an inactive branch of an XOR-block will be copied to the master. Figure 4 shows a brief example of a workflow instance that contains a loop-block with a sequence of two tasks.

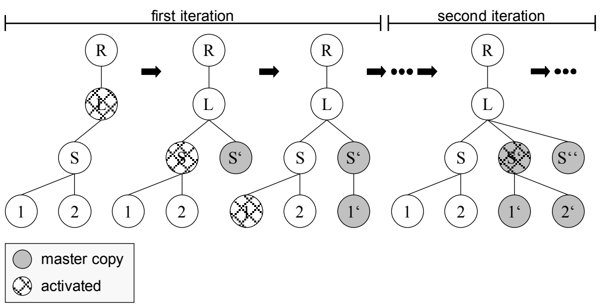


Figure 4. A workflow instance with a loop.

At the beginning of the first iteration, the loop node L is activated by R. Second, S is started and a copy of S is linked to L as first element of the master. S controls the serial execution of the two tasks 1 and 2. For each task that is activated a copy of the task is attached to the corresponding location in the master (see step 3 in Figure 4). As soon as all children of the sequence are completed, L receives an event that the first iteration has been finished. L has to decide whether a second iteration will be started. In case it is, the sequence S' is executed, a further master S'' is created, and the above described mechanism is repeated. In the other case, L is marked as completed and the parent of the loop receives a notification.

4. Configurable workflow contexts

Besides of the control flow structure, a workflow has a context model (see Figure 1). Discussions with chip designers led us to the decision to develop a *configurable context model*. For each type of product, the context model can be tailored to the appropriate set of context factors. For instance, security aspects play a major role in the automotive area while factors from the application context like the number of pixels of a camera are more important for consumer products. Our analysis of change request documents from the chip domain has yielded several interdependencies between context factors. For instance, the output pins of a chip segment have a direct impact on the input pins of a successor chip. We have chosen to organize the context in an ontology-based model in order to *describe interdependencies*.

Figure 5 depicts the top-level ontology for our context model. We distinguish two areas for the definition and the application of context factors:

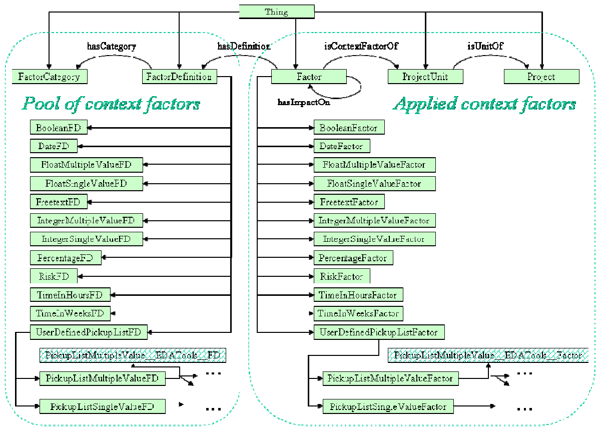


Figure 5: The top-level ontology of the context model.

1. *FactorCategory*, *FactorDefinition* and all its descendant classes (*BooleanFD*, *DateFD*, etc.) belong to the pool of context factors. The pool contains the definitions of the factors including the value types and default values.
2. *Project*, *ProjectUnit* and *Factor* with its descendants (*BooleanFactor*, *DateFactor*, etc.) are for the assignment of factors to project units and their unit-specific values.

We use Protégé conform owl as a context exchange format. The users are able to exchange either the pool of factors only or in combination with the actual application of this pool to projects. We have selected owl because it extends RDF and XML by expressions in description logics (DL) that we use for some properties and restrictions of the relations, e.g. for the property that *hasDefinition* is functional and for the existence restriction concerning *hasDefinition* in *Factor*. These two sample expressions mean that *hasDefinition* must be specified for each instance of *Factor* with exactly one instance of *FactorDefinition*.

The context model can be configured by means of our context acquisition tool. We hide the complexity of the owl format from the users. For further information on the implementation of the context acquisition tool, we refer to the literature [6].

5. Authoring support for the adaptation of workflows

The experience that is contained in an ongoing workflow instance and the changes applied to it can be captured within cases according to the CBR approach. The cases are stored within a case base that can be queried by the user. A query is a description of the current problem situation, i.e. a current workflow instance that is to be modified. The result of the retrieval is a list of the best matching cases from the case base. The past solution from a retrieved case can be reused in order to adapt the current workflow. At the moment, the user has to transfer the solution for adapting the current workflow manually, but in future, we will investigate to what degree this can be supported automatically by proposing potential adaptations.

A case consists of a pair of subsequent revisions of a workflow instance [X, X']. The previous revision X is the problem part of the case; X' - the revision of the workflow instance that has already been modified - is the solution part of the case.

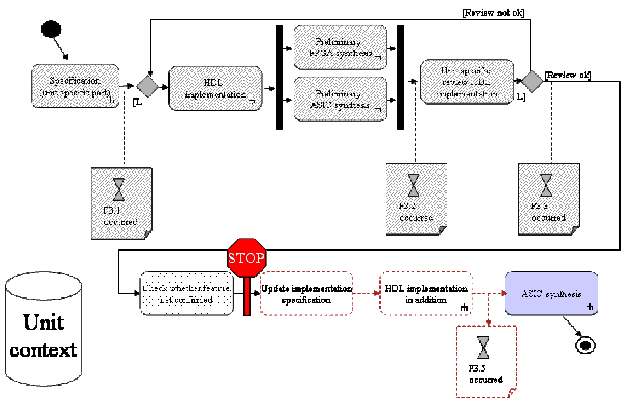

[Review not ok]
Specification (unit specific part)
HDL implementation
Preliminary FPGA synthesis
Preliminary ASIC synthesis
Unit specific review HDL implementation
[Review ok]
[L
L]
P3.1 occurred
P3.2 occurred
P3.3 occurred
STOP
Unit context
Check whether feature set confirmed
Update implementation specification
HDL implementation in addition
ASIC synthesis
P3.5 occurred

Figure 6. Revision of a sub-workflow instance.

Figure 6 shows a sample revision of a sub-workflow instance that has been derived from the workflow definition in Figure 2. It has developed from the initial workflow instance in two adaptation steps: First, it has been extended by the additional task *Check whether feature set is confirmed*. Second, *Update implementation specification* and the sub-diagram with the placeholder task *HDL implementation in addition* have been inserted later on. The context model may have been changed as well, for instance, by modifying the value of a context factor. The two adaptation steps of the workflows' structure and the context model form two subsequent revisions X and X' of the workflow instance. They can be stored as a sample case in our case base where X is the problem part and X' the solution part of the case.

In order to determine the best matching cases to a query workflow instance, we use a similarity function that compares the query pair wise with the particular cases of the case base. The resulting similarity values induce an order of cases. The best matching cases are the retrieval result. The similarity function requires an appropriate representation of the cases. Luo et al. [5] have developed a building block similarity for traditional workflows that would fit quite well to the tree representation that we have used for the execution of workflows. Unfortunately, this method is not suitable for changes of the order of workflow elements what is typical for our workflows. Minor changes, for instance, moving a task to a different block lead to major restructuring activities within the building block tree and consequently seems to impact the similarity values to a too high degree. Due to this, we have developed an alternative representation for retrieval purposes (see below). The cases can be transformed automatically to this retrieval-specific representation.

The retrieval-specific representation of a workflow instance in a particular revision has two parts: one for the control flow structure of tasks and another one for the context model. The context is represented by a structural CBR approach with attribute-value pairs in a straightforward way. The representation of workflows' structure makes use of the fact that the instances are derived from a particular workflow definition. As the instances usually differ only slightly from their templates, they can be described by means of the difference to their workflow definition. A workflow definition is represented as a set of elements, such as tasks and control flow elements, as well as a successor-predecessor-relation on this set. The difference between an ongoing instance and its workflow definition covers the following issues:

1. the structural modifications of tasks and control flow elements
2. the state of processing

Both can be encoded by sets for added and deleted workflow elements with respect to the original template. Hereby, completed tasks as well as passed control flow elements are regarded as deleted.

The similarity of the workflows' structure is computed based on graph edit distances [3] that make use of the sequences of add and delete operations [7]. The similarity of context models is be computed according to the local-global-principle of the structural CBR approach [1]. Both of the similarity values for context and structure are aggregated to an overall similarity value.

6. Discussion and conclusion

We have presented an approach of new, flexible workflow technology that has been motivated empirically by our analysis of the chip design domain but is not restricted to this domain. It is applicable for further domains with collaborative, long-term processes under changing requirements like software engineering or flexible manufacturing.

The state of the Java implementation of our approach is the following: the prototypical core of the workflow engine has been implemented; the context acquisition tool is already under evaluation at our industrial partner sci-worx; the further GUI's and the authoring support component are still under development. Our prototype will be integrated with the CAKE system [3] as an additional component.

The benefits of our system are the following: The system is able to deal with huge process models with thousands of tasks as it supports late-planning and ad-hoc changes of workflows. The adaptations are embedded in a sophisticated suspension mechanism that supports the users to find a good balance between tight deadlines on the one hand and changing requirements as well as delayed decisions on the other hand. The configurable context model extends the workflows' structure in order to document the experience with the development of different product types. A case-based retrieval mechanism supports the reuse of experience for the adaptation of ongoing workflows.

7. Acknowledgements

The authors acknowledge the Federal Ministry for Education and Science (BMBF) for funding URANOS under grant number 01M3075. We acknowledge the assistance we have received from both Stefan Pipereit, Silicon Image and Marko Höpken, Silicon Image as well as of further chip design experts from AMD, Infineon, and Cadence.

8. References

[1] R. Bergmann. *Experience Management: Foundations, Development Methodology, and Internet-Based Applications*. LNAI 2432. Springer-Verlag, Berlin, 2002.

[2] H. Bunke, B.T. Messmer. Similarity measures for structured representations. In M. M. Richter et al., editors, *Preproceedings EWCBR-93, First European Workshop on Case-Based Reasoning*, pages 26–31. University of Kaiserslautern, 1–5 Nov 1993.

[3] A. Freßmann, R. Maximini, and T. Sauer. Towards collaborative agent-based knowledge support for time-critical and business-critical processes. In K.-D. Althoff et al., editors, *Professional Knowledge Management: Third Biennial Conference, WM 2005*, LNAI 3782, pages 420 – 430, Springer-Verlag Berlin Heidelberg 2005.

[4] Th. Herrmann: Lernendes Workflow. In Th. Herrmann, A.W. Scheer, H. Weber, editors.: Verbesserung von Geschäftsprozessen mit flexiblen Workflow-Management-Systemen, pages 143 – 154, Physica-Verlag, Heidelberg, 2001.

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

[6] M. Minor, D. Schmalen, A. Koldehoff, R. Bergmann: Configurable Contexts for Experience Management. In Gronau, N., ed.: *4th Conference on Professional Knowledge Management - Experiences and Visions*. Vol. 2., Potsdam, Univ. of Potsdam, GITO-Verlag Berlin, 119 – 126, 2007.

[7] M. Minor, A. Tartakovski, R. Bergmann. Representation and Structure-based Similarity Assessment for Agile Workflows. *10th International Conference on Case-Based Reasoning*, accepted for publication.

[8] M. Reichert, S. Rinderle, and P. Dadam. Adept workflow management system: Flexible support for enterprise-wide business processes (tool presentation). In W. M. P. van der Aalst et al., editor, *Proc. International Conf. on Business Process Management (BPM '03),* LNCS 2678, pages 370 – 379. Springer Verlag, 2003.

[9] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5 – 51, 2003.

[10] L. van Elst, F.-R. Aschoff, A. Bernardi, H. Maus, and S. Schwarz. Weakly-structured workflows for knowledge-intensive tasks: An experimental evaluation. In *12th IEEE International Workshops on Enabling Technologies (WETICE 2003), Infrastructure for Collaborative Enterprises, 9-11 June 2003, Linz, Austria*, pages 340 – 345. IEEE Computer Society, 2003.

[11] B. Weber, S. Rinderle, W. Wild, and M. Reichert. CCBR-Driven Business Process Evolution. In H. Muñoz-Avila and F. Ricci, editors, *Case-Based Reasoning, Research and Development, 6th International Conference, on Case-Based Reasoning, ICCBR 2005, Chicago, IL, USA, August 23-26, 2005, Proceedings*, LNAI 3620, pages 610 – 624. Springer, 2005.