Proposal for Editing Workflows of a Distributed Software Development Environment

Kolja Markwardt, Daniel Moldt and Jan Ortmann

University of Hamburg, Department of Informatics,
Vogt-Kölln-Str. 30, D-22527 Hamburg
http://www.informatik.uni-hamburg.de/TGI

Abstract In distributed software development projects the different parties involved can be coordinated by the use of flexible workflow management systems (WfMS). Often the process cannot be defined completely in the beginning of a project or has to be adapted later on when conditions change.
In this paper the handling of workflow change in the agent-oriented POTATO system for distributed development will be presented. This includes the interaction of the different agents and their protocols as well as the mechanisms for ensuring the soundness of workflows even if they are changed during their execution.

1 Introduction

For distributed software development, the definition and enactment of processes with workflow management systems (WfMS) is an important means of structuring the interaction between the different participants. Often it can be important to modify these processes during the course of the project. This can be to reflect changes in the general conditions which require changes in the workflow. In other cases, it is not possible to completely specify the required process in the beginning of a project, so that some parts of the workflow can only be defined at a later stage. In ad-hoc workflows defined for one instantiation only, this is obviously more common than in production workflows, in both cases modifications can be necessary, though.

In these cases the workflow process definitions or even the running workflow instances need to be modified. Often it is difficult to decide whether a certain modification can be safely enacted on a workflow. Therefore instead of changing a running workflow changes are often only applied to a new instance, or special monitoring is required. Since workflows in the POTATO system (Process-Oriented Tool Agents for Team Organization) are specified with Petri nets, net-based methods can be used to ensure soundness of these modifications. In section 2 the POTATO system will be described with a focus on the process infrastructure used to enact workflows. Section 3 describes the methods for modification of workflows within the system. In section 2.4 the algorithms for checking the modified workflows for soundness are presented.

2 Workflows in Distributed Software Development

This paper describes the editing of workflow definitions in the context of the Potato system for distributed software development. Therefore this section outlines the main properties of this system.

Potato is an agent application built on the (Petri net based) Mulan/Capa agent platform (see [7,2]). This agent platform itself is built using reference nets and uses the reference net editor/simulator Renew[6] as its execution environment, which is implemented in Java.

The structure of Potato is twofold: It has a tool-based organization that allows users of the system to equip their user agent (UA) with different kinds of tool agents (TA), in order to execute the tasks in the system they need to do. On the other hand a process infrastructure allows the execution of workflow processes in the system, connecting and integrating the different users.

2.1 User, Tool and Material Agents

The main goal of the Potato system is to facilitate the work of different people working together to produce software. To achieve this, users can use different tools to manipulate materials, which are over the course of a project transformed into work results. This follows the notions of the tools and materials approach [10], applied to multi-agent systems to address distributed workplaces.

The main idea about the tool agent concept is that each user controls a user agent (UA), which can be enhanced by different tool agents (TA) (see [4,3]). The user agent provides basic functionality like a standard user interface and the possibility to discover and load new tool agents (tools).

Those tool agents can then plug into the user agents UI with their own UI parts, offering their functionality to the user. By choosing the specific set of tool agents, the user can tailor his workspace to his specific needs. A developer for example needs a completely different workplace then a tester or someone writing documentation.

Material agents (MA) are used to represent and encapsulate the materials or work objects that are currently worked on. Materials are manipulated by tools and can be created, deleted and moved between workplaces. Tools and materials populate the workspace of the user.

2.2 Process Infrastructure

A generic agent-based process infrastructure has been created (see [5]) and is used for Potato. The process infrastructure offers the services of definition and execution of workflow processes in the development environment. It models a complete workflow management system (WfMS) using agent technology. This allows to make use of agent-based features, like distribution and mobility, so that the resulting WfMS is much more flexible than a normal stand-alone one. Within Potato it is adapted to fit into the user/tool-agent structure. To organise the cooperation of different people working together on a project, workflow processes can be defined and enacted.

2.3 Workflow and its Soundness

In [8] workflow nets as a special form of Petri net are defined as well as soundness criteria. A test for checking workflow soundness is also given, which can be executed automatically using for example the Woflan tool [9].

As one of the most wanted properties during the execution of a workflow no deadlocks should occur nor should tokens be "lost" in the process. Therefore the notion of soundness has been defined. If a workflow net is started with a token on the start place, no matter which firing sequence occurs, it will always be possible to reach a marking in which only the end place is marked, and this is the only reachable marking in which the end place is marked. To check this property, the short-circuited net is constructed, by adding a transition to the net from the end place to the start place. Iff this net is live and bounded, the original workflow net is sound.[8]

There are different degrees of possible modifications, that can be handled

2.4 Keeping it Sound

differently. If the definition of a workflow is changed with no currently running instances or if the currently running instances are not to be changed, it suffices to check the new workflow definition for soundness, without any concern for the old definition.

The same is the case, if already running instances are concerned, but changes only occurr within subworkflows, which are not yet started. This is often the case if sections of a process are not specified when the execution begins, and only placeholder subworkflows are inserted to be defined later on. As long as the unspecified segments are workflows of their own, simply checking for soundness of the new definition is sufficient here, too.

It gets problematic however, if workflows have to be changed that already have running instances associated and those instances have to be converted to the new version. Since these instances can be in various states of execution, the standard method of checking for workflow soundness [8] cannot be applied directly.

Ensuring the Soundness of Workflow Modification A workflow definition is considered sound, if for every possible firing sequence it is possible to reach a state, in which only the final place is marked with a token. As mentioned above, for a normal workflow this can be ensured by constructing the sort-circuited net and checking it for the live and bounded properties.

In the case of a modified workflow, the structure of the net changes during the execution, therefore some special constructions are necessary to ensure the soundness.

For every place in the original workflow net a corresponding place in the modified net has to be defined, so that the transition to the new workflow can be executed and checked for soundness. To do this a modification net can be constructed as follows.

For a workflow net its modification net consists of all places, transitions and arcs from the original net as well as the modified net (disjointly united), connected by transitions from all places in the original net to the modified net.

With this construction it is possible to convert any instance of the original net to the new definition since all tokens are then accounted for. With the construction as seen in figure 1 it should also be possible to check the absence of deathlock possibilities during the transformation by checking the modification net for soundness. We will not try to formalize or proof this here though.

Example An abstract example for a Wf modification net can be seen in figure 1. The original workflow W_A consists of two parallel branches, one of which consists of two tasks. In W_B the other branch has got two parallel tasks. In t_{trans} the fact is accommodated, that no tokens must be left behind. Therefore even though one of the parallel tasks is deleted in the modified workflow, the corresponding tokens must be disposed of. Similarly, the task that is split up into two tasks in W_B requires two tokens to be generated in the target workflow.

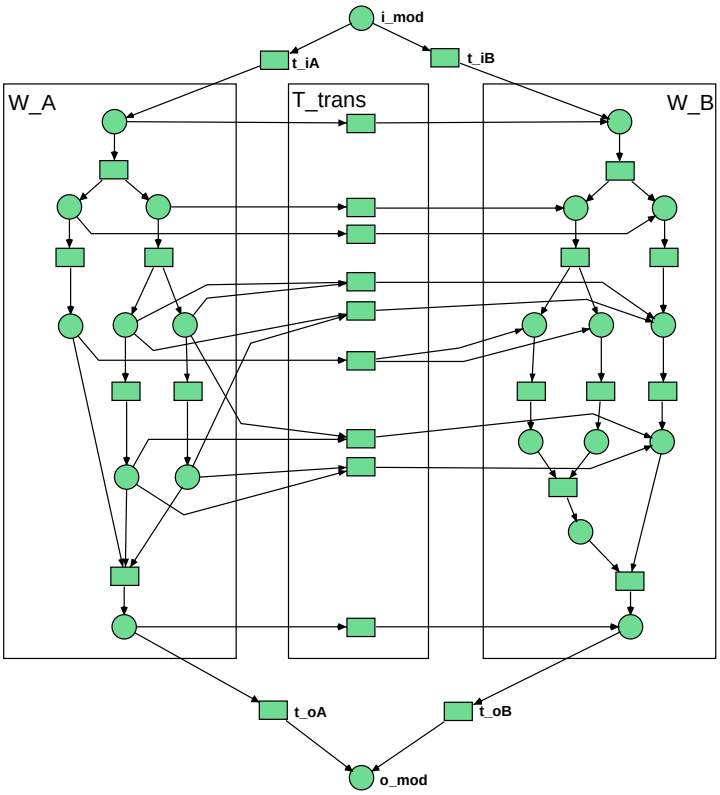

Figure 1. Workflow modification net

3 Modifying Workflows in the Potato System

In this section the mechanisms for modifying workflows are described. Since Potato is agent-oriented, the change process involves a couple of agents.

3.1 Involved Agents

Users with the appropriate authorization can edit workflows. To do so, they use a workflow edit tool agent via their user agent. This tool agent communicates with the workflow definition database agent to get the old workflow definition as a workflow definition material agent. This is then edited and the new version uploaded back to the definition database.

3.2 Adding New Workflows With the WF Edit Tool Agent

To add a new workflow definition, the net editing features of Renew can be used. Additional properties of the workflow can be defined in a special tool, like roles and rules for execution, workflow specific context data, tasks etc.

A material agent is then created and sent to the workflow definition database agent, where it is checked for soundness. This can not find all problems with a workflow, as many problems can occurr in the accompanying definitions of roles, participants, tasks etc. and not in the net structure itself, but at least some of the problems can be avoided this way.

3.3 Modifying Workflows and Their Definitions

If a workflow is to be modified, it must be decided how to handle already running instances of this workflow. If old instances are to be finished according to the old definition, the modification can be handled like a new workflow. The new version can then be added just like a completely new workflow.

If running instances are to be updated to the new definition however, care must be taken to migrate the processes correctly. In section 2.4 a simple algorithm is described to ensure soundness of workflow modifications. It is mandatory however, to specify the migration from the old to the new definition. For every place in the old workflow a place in the new workflow must be specified, so that all tokens can be moved over to the new definition.

In the editing process therefore a special mapping phase has to be added, in which this can be defined. By default it is sensible to assume, that all places existing in both versions of the workflow net are mapped to themselves, but it needs to be checked by the modifying user. If the workflow editing consists of a series of soundness-preserving transformations, each of these transformations could be assigned a default pattern of transformation, which can be adjusted by the user.

Then the new workflow definition along with the migration mapping is sent over to the workflow definition database agent, where it is verified. If verification

is successful, the new version is saved as the new default for this workflow type. All workflow engines currently executing instances of the old definition must then be notified of the change and the migration mappings be applied.

4 Conclusion and Outlook

POTATO integrates the ideas and concepts from [1] and [4,3] to provide a framework. This shall allow for the support of workflows within a group collaborating in a distributed way. Here we proposed the use of user, tool and material agents, which cover specific roles within the application. They allow for easy editing of agent based workflows. Furthermore, we proposed to add formal checks on the workflows resp. their modification at runtime, based on traditonal techniques. The transfer of markings from one running instance of a workflow to another can e.g. be based on the places cuts.
In the long run it is planned to apply these concepts to our own software development process and environment. Therefore, the RENEW-IDE will be enhanced considerably.

References

1. Lawrence Cabac. Multi-agent system: A guiding metaphor for the organization of software development projects. In Petta Paolo, editor, *Proceedings of MATES'07*, volume 4687 of *LNCS*, pages 1–12, Leipzig, Germany, 2007. Springer.
2. Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In *Agent-Oriented Software Engineering III: Revised Papers and Invited Contributions*, number 2585 in Lecture Notes in Computer Science, pages 59–72, Berlin Heidelberg New York, 2003.
3. Kolja Lehmann, Lawrence Cabac, Daniel Moldt, and Heiko Rölke. Towards a distributed tool platform based on mobile agents. In *Proceedings of MATES'05*, volume 3550 of *LNAI*, pages 179–190. Springer, September 2005.
4. Kolja Lehmann and Vanessa Markwardt. Proposal of an agent-based system for distributed software development. In Daniel Moldt, editor, *Third Workshop on Modelling of Objects, Components and Agents (MOCA 2004)*, pages 65–70, Aarhus, Denmark, October 2004.
5. Christine Reese, Jan Ortmann, Daniel Moldt, Sven Offermann, Kolja Lehmann, and Timo Carl. Architecture for distributed agent-based workflows. In B. Henderson-Sellers and M. Winikoff, editors, *Proceedings of AOIS'05*, pages 42–49, 2005.
6. RENEW – the reference net workshop homepage. `http://www.renew.de/`, 2008.
7. Heiko Rölke. *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*, volume 2 of *Agent Technology – Theory and Applications*. Logos Verlag, Berlin, 2004.
8. Wil M. P. van der Aalst. Verification of workflow nets. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, pages 407–426, London, UK, 1997. Springer-Verlag.
9. H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst. Diagnosing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
10. Heinz Züllighoven. *Object-Oriented Construction Handbook*. dpunkt Verlag, 2005.