A Process-Oriented Tool-Platform for Distributed Development

Kolja Markwardt, Lawrence Cabac and Christine Reese

University of Hamburg - Department of Informatics, Germany

Abstract. Many software projects today are executed geographically distributed with teams of developers, designers, testers, etc. in different countries all over the globe. This requires a development environment that allows easy and flexible adaptation to different kinds of teams and their processes. This paper presents an architecture for a distributed software development environment, that allows users to collaborate on flexible processes. The focus is put on providing distributed tools and organising the processes needed to successfully produce software with these tools.

1 Introduction

Distributed software development is one of the major concerns today. Many different approaches are made for the creation of flexible environments to solve the different challenges encountered here. A distributed software development environment needs to offer users powerful tools to handle their tasks, as well as allow the interaction and coordination of distributed partners working together on a project.

Some examples for distributed software development environments are Jazz [6] by the Eclipse [4] project or Microsoft Visual Studio Team System [10]. In these cases existing IDEs are enhanced with collaboration features to address the needs of distributed development.

We try to start from an inherently distributed system, namely a multi-agent system. On that basis we first build a tool platform, which can be used for any kind of distributed collaborative work. Our goal and case study is then the distributed software development environment.

Other tools focus on certain aspects of distributed development, for example D-Meeting [1] supports distributed meetings and the gathering of meeting results. While synchronous processes are possible in our approach and a tool for synchronous communication has been implemented [14], the focus here lies on asynchronous processes.

This paper describes the POTATO (Process-Oriented Tool Agents for Team Organization) system for distributed software development. It is a tool platform, designed to integrate different kinds of tools into one process-oriented environment. The tools can either be standalone or collaboration tools. The interaction between the users is controlled by a process infrastructure that allows the definition and execution of process-based applications. This tool platform can then be used to build a distributed software development environment. Tool agents are used to build tools that can be used to execute tasks in the system.

Markwardt K., Cabac L. and Reese C. (2009).
A Process-Oriented Tool-Platform for Distributed Development.
In *Proceedings of the 7th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 43-52
DOI: 10.5220/0002219200430052

The underlying technology used for the development are reference nets, a special class of colored Petri nets suitable for the modelling and execution of complex concurrent systems. Within these nets, Java code can be used for inscriptions. Based on the runtime environment RENEW, CAPA is a complete multi-agent system platform implemented in Petri nets, which is then used to build a system of tool and material agents as well as a process infrastructure. These subsystems are then combined into the POTATO-system, which offers these functionalities to application programs.

Section 2 describes the agent platform that is the basis for the system, highlighting how nested agents/platforms can be used to structure the system on an intuitive yet powerful way.

Following in section 3 is the process infrastructure used to create process-oriented applications. Processes are designed as entities in their own right, controlling their own execution. The architecture for tool agents can be seen in section 4. Section 5 outlines the integration of the process infrastructure with the tool agent platform. In the end, section 6 shows some example scenarios to show how the components work together, followed by a conclusion and a view on future work.

2 Agent Systems

For the design of concurrent, distributed systems agents and multi-agent systems provide very useful tools and metaphors. They are especially useful where different parties work together in a heterogenous environment. This is often the case in distributed software development, when different companies work together on a project. This section describes the agent platform CAPA on which the POTATO system is built.

Agent Platforms. The POTATO system is built on the MULAN agent platform [13]. MULAN is implemented using reference nets [7] as a modelling and execution language. Reference nets are a special formalism of colored Petri nets, in which references to nets can be used as tokens within other nets. This allows the building of very complex models. Thanks to the integration of Java commands within the formalism, it is possible to graphically model a system and use this model as actual, executable code. The extension CAPA [3] augments this platform to a complete FIPA-compliant [5] agent platform.

The basic model of MULAN/CAPA defines agents, which reside on agent platforms. They own a knowledge base in which their knowledge is kept. The behavior is determined by a number of agent protocols and decision components within the agent.

Protocols. Even though in theory agents possess autonomy, they need to adhere to agreed-upon communication standards to enable meaningful interaction. In CAPA each agents' behavior is specified by a set of agent protocols. The term protocol in CAPA is used for one agent's part in an interaction, which works together with other agents' protocols to form a conversation.

Modelling an agent system then consists of two orthogonal dimensions, designing the agents that are part of the system (the structure) and designing their protocols (the

behavior). If both parts have their own identity, agent and workflow definitions, it is easier to focus on one at a time and get to a cleaner design. The next section shows how workflow processes are handled in the POTATO system.

3 Processes

To organise the cooperation of different people working together on a project, workflow processes can be defined and enacted. An agent-based process infrastructure has been created to take care of this important aspect of collaborative systems. [11].

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.

Not only the different parts of the WfMS are modelled as agents, the workflow itself is, too. Once a workflow is instantiated, a workflow agent is created, that holds the process definition as well as any specific case data. This agent then takes care of the execution of the contained process, interacting with any other agents which play a part in the process.

Process Infrastructure. The process infrastructure is a subsystem within the POTATO tool platform which offers workflow execution services. This allows the explicit modelling of structure elements of the application (agents) and behavior elements (processes). The elements of a classical workflow management system (WfMS) have been implemented as cooperating agents, thus allowing easier distribution among different platforms as well as a consistent execution environment.

At the center of the WfMS is the execution of workflow process definitions, which are specified as Petri nets. Since the CAPA platform itself is implemented with Reference nets (see section 2), this allows a very smooth integration.

The execution of process definitions is handled by workflow engine agents. They are instantiated by a workflow enactment agent, which has knowledge of the different process definitions within the system and creates new workflows on demand. Usually a workflow engine agent can handle several workflow processes.

The workitem dispatcher agent handles the worklists of all registered workflow participants and updates them whenever changes occur. Participants can select tasks for execution and inform the dispatcher of their completion. Whenever an update of the activation state of a task transition happens the dispatcher sends an update of all relevant worklists to the participants.

Another agent is used for the handling of roles, rules and rights (RRR agent). It decides who can access which functionality in the system, for example executing certain tasks or define new processes and tasks. The RRR agent also performs security checks on operations and when a user logs in into the WfMS.

Security issues will be handled by the security subsystem, which is another multi-agent system due to the complexity of security in distributed systems. The RRR agent strongly cooperates with the security multi-agent system.

Workflow Agents. The workflow agent plays a special role in the process infrastructure, as it is essentially a process definition which gets elevated to the status of an agent. Usually the interaction between agents is regulated by protocols, which are implicitly agreed upon, and whose parts are distributed between the participating agents. With the workflow agent, this implicit understanding is encapsulated into an own entity, putting it into the focus and allowing a single point of access.

Whenever a new process instance is created in the process infrastructure, the workflow enactment agent creates a new workflow agent and initialises it with a process definition (a workflow Petri net). Additional information about the process context, like participants, case data, distribution aspects etc. can be stored as information within the net.

The workflow agent is then connected to a workflow engine agent and makes sure the process is executed correctly. If needed, the workflow can be fragmented [12] and migrated to different platforms and other WfMSs to e.g. ensure the correct execution of an interorganizational workflow.

In this way designing the interactions between agents is much easier. It is no longer necessary to check that all interfaces are correct and every agent protocol fits together with the others, since everything can be designed in one central place and distributed at runtime. Of course the process infrastructure is more complicated then a traditional WfMS due to the higher flexibility.

4 Tool 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 [15], applied to multi-agent systems to address distributed workplaces.

Overview. 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). [8] The user agent provides basic functionality like a standard user interface and the possibility to load new tool agents. 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.

User Agent. The user agent is every user's starting point into the distributed development environment. Its role is therefore a dual one; one part is the UI component which allows the user to access the system, the other part is the connection to the multi-agent

system.

The user interface needs to be flexible enough to allow easy integration of new tools and materials. In the prototype currently worked on, the user interface is implemented as an Eclipse plugin using the Rich Client platform and SWT. [4] Other systems would of course be possible, as long as user and tool agents agree on one standard.

Tool Agents. To access functionality in the system, tool agents are used. This can be an encapsulated legacy application, local handling of a material, cooperation with other users via connected tool agents or access to a remote service provider (see figure 1).

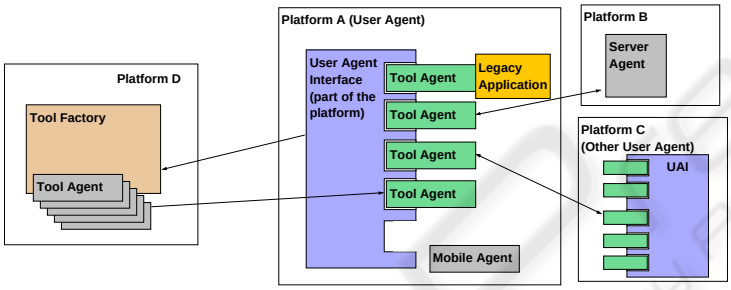

Fig. 1. User and Tool agents.

They are created within the user agent or migrate there in order to offer certain functions to the user. A tool agent is usually explicitly ordered by a user agent. It is also possible to send a tool agent to the user agent to request certain actions from him. This is for example the case with tool agents used within the process infrastructure for task execution. When a user accepts an open task, a tool agent for handling that task can be sent directly to the user.

Tool Factory. Not every user agent platform needs to know all possible tool agents. A repository can be used to store tool definitions and generate new tools on demand. User agents can query this tool factory for available tool types and the tool factory generates new tool instances on demand. If the tool factory is located on a different platform than the user agent, the tools can be migrated over the network to the target platform.

Material Agents. Material agents represent work objects in the system, such as a piece of source code, a documentation item, a system specification etc. Materials are handed over between user agents, who can then work on them, transforming and modifying them using tool agents.

A material agent allows access to its internal data over an interface specific to this material, like an object in object-oriented programming. For creating materials a special repository like the tool factory is conceivable, otherwise certain types of tools can create new materials.

The workflow agent (section 3) is a special, complex material. It encapsulates the workflow process definition and the current execution status, but can also contain other materials needed in the course of the process, like work documents generated and manipulated.

5 Integration

This section describes the integration and interaction between our process infrastructure and the tool agents architecture. These two concepts are essential for the development environment envisaged and need to work together smoothly.

A user of the system has to achieve anything by the use of tool agents. His workspace offers him different tools to use, some of which involve using workflows in one way or another. For most of the interfaces a WfMS has to offer according to the WfMC reference model (see figure 2), tool agents can be applied.

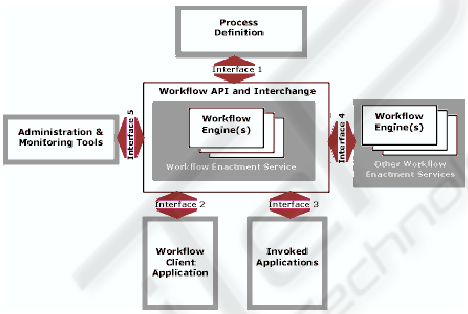

Fig. 2. Workflow Interfaces [2].

Workflow Definition Tools. The first interface is the workflow process definition. Users who are allowed to define their own processes can do so by using a process definition tool. The created process pattern is a material, which is then handed over to the workflow database agent. The workflow enactment agent can access and instantiate it then.

At the current time, no such tools exist yet. The process definitions are usually designed directly in the RENEW tool and added to the workflow database manually.

Workflow Client Application Tools. The workflow client application and invoked application interfaces are used to trigger the actual work to be executed in the course of the workflow execution. Once a user selects a task for execution, the process infrastructure creates a tool agent specifically for this task instance. The tool can be tailored to the task and contains all necessary context information to allow the successful handling.

Invoked applications do not require user interaction. Service agents are used to execute these tasks, but the same interface as for tool agents can be used here as well.

Workflow Engine Interface. The fourth interface of the reference model is used to connect a WfMS with other, remote workflow engines. This interface can be used to construct distributed workflows or execute parts of a process in a remote WfMS. With the notion of workflows as tools (see below), subprocesses can be achieved by simply calling a remote WfMS as a tool.

More complex interaction has to be negotiated directly between the WfMS', though. This is still an open topic for further research.

Workflow as Tool. Tool agents cannot only be called by the WfMS, they can initiate and control a workflow, too. A tool can offer some functionality to the user, which relies on the execution of a (sub-)workflow. So when the functionality is called from within the tool agent, a new workflow agent is created. During the course of its execution, other tool agents might be called.

Yet another tool can offer monitoring functionalities over workflow instances the user owns. Both the starting and monitoring of processes uses the administration and monitoring interface of the WfMS.

6 Scenarios

To understand the purpose of the system, this section describes some usage scenarios. The process for Change Management has been implemented as an example subprocess of distributed development. It shows how the different parts of the system work together. The complete process envisaged for distributed development is outlined afterwards.

6.1 Change Management

An important process in software development and maintenance is the handling of changes in the software. Changes can originate from bugs found during testing, changing requirements or enhancement of the software with new functionality. To successfully execute a change it is important, that all parties involved follow the predefined processes, because only then transparency can be ensured.

A basic change management application has been built based on the POTATO tool platform. [9] The exact processes used in change management are very dependent on the actual context, like involved parties (internal and external), quality of service agreements, additional bug tracking software, company policies etc.

In our scenario, the workflow starts with a change requester (for example a user representative) opening a new change request (CR) via his CR tool. This connects to the Workflow enactment service and requests a new workflow of the type "Change Request handling" to be started. The workflow enactment agent looks up the process definition, adds the requester to the list of participants and starts a new workflow agent. The beginning of the workflow is shown in figure 3.

Now the CR could be assessed, categorized and edited, cost estimates be made and negotiated, before the CR is actually worked upon. In this example all CRs are immediately executed, as might be the case in a small internal CR cycle. So the next task in line is the implementation of the required change.

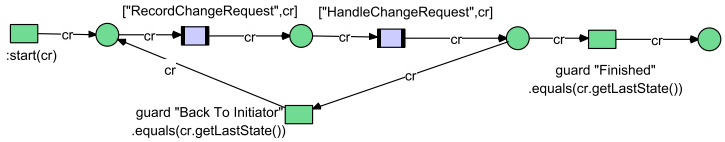

Fig. 3. Change Management Workflow (detail).

At some point the process is finished and the CR is closed. The tool which started the workflow is notified of this and can provide some feedback about this. Of course a real Change Management workflow would be more complex and require different tools.

6.2 Distributed Development

The eventual goal of the POTATO system is to support the complete process of distributed software development. The aspect of Change Management has just been shown, other processes are yet to implement. The overall structure is also more complex than was necessary in the last example.

Different companies working together at different places will all have their own agent platforms. These platforms are interconnected and share some resources, but certain information will be kept private. For example a process spanning two companies can be defined as public information, but the detailed subprocesses each company uses internally is private information.

To implement a distributed project, either a standard process is selected or a new process definition is tailored for the specific project. Since most distributed projects are very complex, this will usually be necessary in order to address all the specific circumstances of the project. This global process is then enacted on either a shared platform for the project or one of the participants' platform.

The different activities are then assigned to the relevant parties and handled accordingly. For example a service provider will be assigned the task of writing documentation. This task could be handled by some kind of tool agent, it is more likely however, that it spawns a whole new subprocess on the side of the service provider.

In a complex network of collaborating companies, changing teams can be put together to work on different projects, each time designing new processes and creating new temporary collaboration platforms.

7 Conclusions and Outlook

In this paper the process-oriented tool platform POTATO has been described. It has been stated, how such a tool can be used to support distributed collaborative software development processes. The different constituent parts have been described as well as their interaction.

The process infrastructure as well as the tool agent architecture have been described as the constituent parts of the system. The integration of these two parts was described and motivated with two scenarios.

At the current time, a prototype of this system exists, the test application for change management processes in the context of distributed development has been built [9]. To facilitate other processes of distributed software development, the different processes used in this domain have to be described in detail and implemented as workflow definitions. In parallel tool and material agents have to be implemented to enact the processes.

References

1. Naoufel Boulila, Allen H. Dutoit, and Bernd Bruegge. D-meeting: an object-oriented framework for supporting distributed modelling of software. In *International Workshop on Global Software Development, International Conference on Software Engineering*, 5 2003.
2. Workflow Management Coalition. Wfmc workflow reference model, 1995.
3. Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Concurrent architecture for a multi-agent platform. In *Agent-Oriented Software Engineering III: Third International Workshop, AOSE 2002, Bologna, Italy, July 15, 2002. Revised Papers and Invited Contributions*, number 2585 in Lecture Notes in Computer Science, pages 59–72, Berlin Heidelberg New York, 2003.
4. eclipse project. $www.eclipse.org$, 2008.
5. Foundation for intelligent physical agents. URL $http : www.fipa.org$, 2008.
6. Jazz project. $http : www.jazz.net$, 2008.
7. Olaf Kummer. *Referenznetze*. Logos Verlag, Berlin, 2002.
8. 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.
9. Kolja Markwardt, Daniel Moldt, Sven Offermann, and Christine Reese. Using multi-agent systems for change management processes in the context of distributed software development processes. In Shazia Sadiq, Manfred Reichert, and Karsten Schulz, editors, *The 1st International Workshop on Technologies for Collaborative Business Process Management (TCoB 2006)*, pages 56–66, 2006.
10. Microsoft visual studio team system. $http : msdn.microsoft.com/vsts2008/products$, 2008.
11. Christine Reese, Jan Ortmann, Daniel Moldt, Sven Offermann, Kolja Lehmann, and Timo Carl. Architecture for distributed agent-based workflows. In Brian Henderson-Sellers and Michael Winikoff, editors, *Proceedings of the Seventh International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2005), Utrecht, Niederlande, as part of AAMAS 2005 (Autonomous Agents and Multi Agent Systems), July 2005*, pages 42–49, 2005.
12. Christine Reese, Jan Ortmann, Daniel Moldt, Sven Offermann, Kolja Lehmann, and Timo Carl. Fragmented workflows supported by an agent based architecture. In Manuel Kolp, Paolo Bresciani, Brian Henderson-Sellers, and Michael Winikoff, editors, *Agent-Oriented Information Systems III 7th International Bi-Conference Workshop, AOIS 2005, Utrecht, Netherlands, July 26, 2005, and Klagenfurt, Austria, October 27, 2005, Revised Selected Papers*, volume 3529 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, 2006.
13. Heiko Rölke. Modellierung und implementation eines multi-agenten-systems auf der basis von referenznetzen. Diplomarbeit, Universität Hamburg, 1999.
14. Steven Willmott et al. Netdemo: opennet networked agents demonstration. In Michael Pechoucek, Donald Steiner, and Simon Thompson, editors, *AAMAS 2005. Proceedings (Industry Track)*, pages 129–130, 2005. 2 individual demos: (1) CAPA: The CAPA Mobile Chat Agent & Web Services Gateway Agent and (2) Settler: AgentBased Settler Game.
15. Heinz Züllighoven. *Object-Oriented Construction Handbook*. dpunkt Verlag, 2005.