Facilitating Collaboration in a Distributed Software Development Environment Using P2P Architecture

Maryam Purvis, Martin Purvis, and Bastin Tony Roy Savarimuthu

Department of Information Science, University of Otago
P O Box 56, Dunedin, New Zealand
{tehrany,mpurvis,tonyr}@infoscience.otago.ac.nz

Abstract. This paper describes efforts to facilitate collaborative work in a distributed environment by providing infrastructure that facilitates the understanding of inter-connected processes involved and how they interact. In this work we describe how our agent-based framework supports these. This distributed work environment makes use of both P2P and client-server architectures. Using an example of developing an open source software system, we explain how a collaborative work environment can be achieved. In particular, we address how the support for coordination, collaboration and communication are provided using our framework.

1 Introduction

Distributed software teams are becoming more common in today's software projects, because the teams are based on skill pools that are available in the global community rather than being constrained with local resources. Distributed software development [1,2] involves collaboration of people from distributed geographical locations. This presents challenges in day-today activities in areas, such as co-ordination, collaboration and communication [2,3]. Co-ordination and collaboration can be facilitated by the provision of flexible communication mechanisms. In the context of collaborative work, an important factor that impacts the success of the final outcome is how effectively any issues associated with the shared objective are communicated and resolved. Such communication can be direct, such as face-to-face interactions, telephone conversations, interactions by means of chat tools, email, etc; or they can be indirect through common artifacts associated with the final outcome. In the context of developing an open source software system, the artifacts associated with the final product comprise documents, process models, source code etc. A mechanism is needed that ensures these constantly evolving artifacts are easily accessible to the collaborating partners. So there is a need for a system that provides infrastructural support for the smooth functioning of a collaborative work environment.

S. Joseph et al. (Eds.): AP2PC 2006, LNAI 4461, pp. 167–174, 2008.

We will assume that in a context of an open source software development, a distributed team working on a particular project is composed of a few sub-systems. For example in the development of an operating system, the sub-systems can be developing the kernel, I/O and file system, mail system, networking, a set of tools etc. A number of interested people work towards the development of each sub-system. In this environment the following elements can be useful:

- A model that represents the functional and behavioral aspects of the project
- A model that represents the sub-system level activities
- A model of the communication protocol (Interaction Protocol) between various collaborators

In this paper we describe how these capabilities are incorporated into the collaborative work environment. Using various scenarios we also explain how these features are utilized. To achieve a collaborative work environment and provide communication mechanism between interacting collaborators we use the agent based system OPAL [4]. Using this system we can model each collaborator as a software agent. The Coloured Petri Net [5] formalism is used to model the activities of the collaborators as well as the communication protocols. These models are presented in more detail in Section 3.2.

2 Background

To develop the infrastructure needed for collaborative work environments we have used Coloured Petri nets to represent process models and software agents as the building block for providing P2P support. We use Coloured Petri nets (CPN) as a formalism to model workflows in our system. The mathematical foundation behind the Coloured Petri nets makes it a useful tool for modeling distributed systems. A detailed description of CPNs can be found in [5].

We have used software agents to build our system. Some of the commonly accepted characteristics of an "agent" (listed by Bradshaw [6]) are reactivity, collaborative behaviour, communication ability, adaptivity and mobility. An important benefit is that multiagent systems facilitate distributed and open architecture. Such a system can be adaptable and is robust under conditions of local failures and changing environmental conditions.

The next section describes an open source software development scenario and explains how the P2P architecture is used.

3 Collaborative Work in Open Source Software Development

3.1 An Overview of Collaborative Work

In this section we describe the collaborative work associated with an open source software development environment. Figure 1 shows how several collaborators residing in one location (e.g Dunedin), can communicate with other collaborators

in another location (e.g Wellington). Collaborators A, B, C, and D may be involved in the development of one sub-system (such as a kernel sub-system), while collaborators B and E are working on another sub-system (such as a networking sub-system). For each of these sub-systems there exists a server to which the sub-system members may commit their internally developed local artifacts. The sub-system servers periodically update their stable releases to the project server.

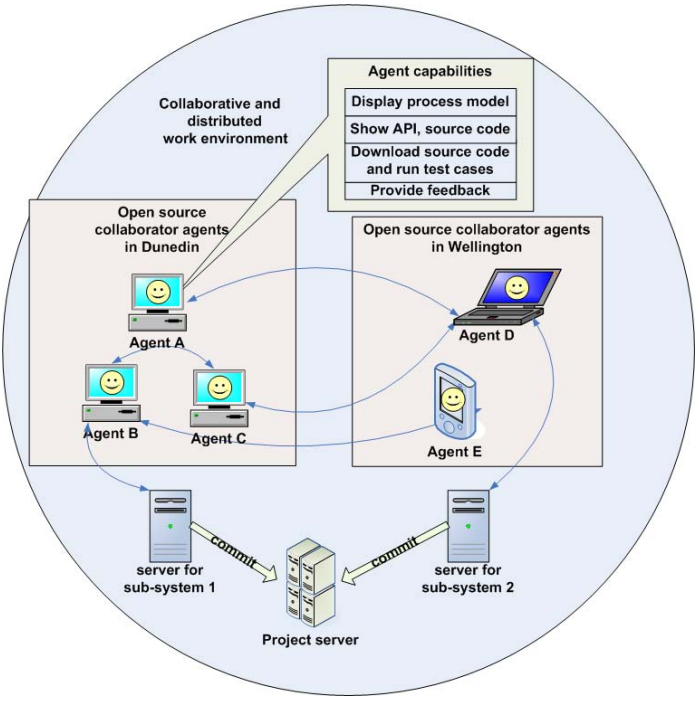

Fig. 1. An agent based collaborative software development environment

There are both inter-group and intra-group communications in the collaborative work environment. However, the inter-group communications may be more frequent, due to a possibly higher level of dependencies between the various components involved. Due to frequent changes in modules during development and the need to integrate the related modules, it is possible that members of a group will access a particular module even when it is not quite suitable for final release. For example, one member may want to obtain the API of a module, or the supporting document such as the specification, associated test cases and so on. In these circumstances the members can obtain a pre-release module for preliminary testing from the module developer directly using P2P communication. The members can thus publish pre-release modules that can be accessed by another module for integration and testing purpose. If there are any conflicts

in terms of the expected interface and the current interface, it can be sent as a comment or feedback to the developer of that particular module.

On comparing with the work done by other researchers [1,2,3], our approach provides a formal and uniform way of communication mechanism between the peers by using Interaction Protocols. In addition, the collaborative agent has a built-in knowledge of the system interfaces and dependencies. This knowledge can be used in informing the collaborators to take certain actions when required in the context of software development (explained in scenario 5).

The functionalities provided by each agent are indicated inside the callout box at the top of Figure 1. The agents can perform various software engineering activities such as displaying process models, showing API and source code, downloading source code and test cases. The agents can then provide notifications on updates and feedback on the artifacts developed.

3.2 Scenario Description

In the following scenarios we demonstrate how coordination is achieved by interconnecting the overall process models with the sub-system process model in scenario 1. Similarly in scenario 2, we describe how agent interaction protocol and the model associated with each of the transitions are linked.

Group collaboration is described in scenarios 3, 4 and 5 where the participating agents can make the project artifacts available to each other and make certain requests. Coordination and collaboration are realized through agent-based peer to peer mechanism provided by our agent-based framework.

Scenario 1: Sharing a common understanding of the overall process model of the project. All collaborative partners should share a common understanding of the project that they are working on. To facilitate this common understanding we use Coloured Petri nets to represent the overall structure and behaviour of the project. The project moderator develops the process model (through discussion with related resources).

The project manager sends an XML-based process model via agent based communication modes to all the participants. The participant agents can then display the process model. The collaborators can modify the process models and send the result to the moderator agent. The moderator agent collates various process models and sends the models again to all the participants for choosing the suitable process model (perhaps by consensus).

For example the model shown in Figure 2 describes the overall project structure and the dependencies between various components. This model shows that the project is partitioned into three sub-systems, s1, s2 and s3. It can be observed that s1 and s2 can be performed concurrently. The diagram also shows that s3 depends on s1 and s2. Each sub-system in turn is represented using a CPN model which results in a hierarchy of process models that describe the overall model of the system.

Scenario 2: The process associated with the communication between agents. The generic process model describing how agents communicate with

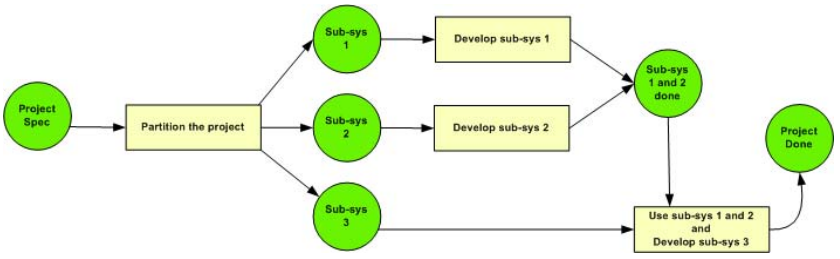

Fig. 2. Model associated with message handling (communication) in each agent

each other is given in Figure 3. Note that the interaction between collaborating members of the software development team are represented in the model by interactions between software agents, representing those team members. Each model can be executed by a collaborator agent [7] which makes use of JFern [8], a Petri net engine. The collaborator agent performs the following operations:

- Receive and parse the requests coming from other collaborator agents
- Send results to other collaborator agents

There exists a message dispatcher in the agent based framework that dispatches messages that reach the "out" node shown in Figure 3. All messages coming to a particular agent will be accumulated in the "in" node of that agent and out-going message will be placed in the "out" node of that agent [9].

When an agent recognizes a message in its "in" node, it evaluates which transition should be invoked based on information received. Each of these top-level transitions may be considered as an abstraction for a more detailed sub-model Petri net that represents a refinement of the top-level abstraction. For example the processRequest transition expands to a sub-model where the decision regarding which type of request is handled (such as show API or download source code activity) can be invoked. Once the activity is performed, the control is returned to the parent process model's transition.

Scenario 3: Group configuration. In our agent based framework, for each project, a moderator agent is created. Our framework uses OPAL's JXTA implementation [10] to facilitate peer to peer communication which allows for agent discovery, joining and leaving. Collaborator agents can join a given project by searching the projects listed in the directory service of the system. In doing so, the collaborator agents interact with the moderator (such as finding details about the overall process model). Similarly, a sub-system can be formed when one of the collaborator agents itself, chooses to become a moderator. The newly joining agents can then decide to join this specific group to implement a particular sub-system. It is also possible that one collaborator agent can be a part of two sub-systems.

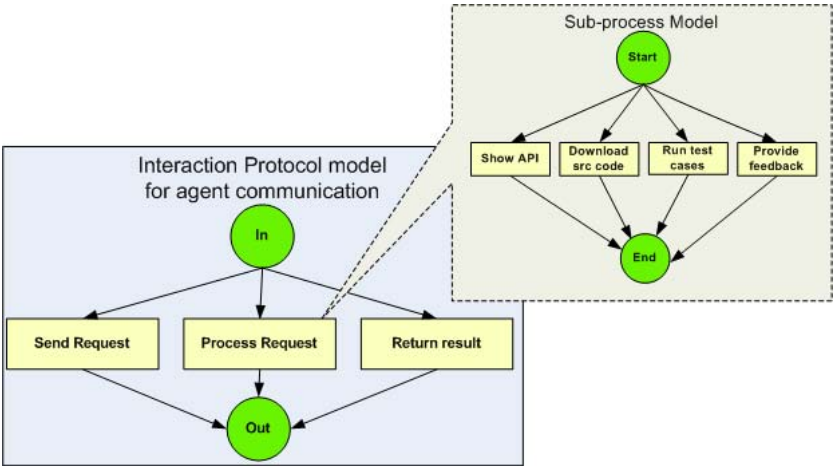

Fig. 3. Model associated with message handling (communication) in each agent

When changes are made to the artifacts produced by each sub-system, its members are notified. Also, the changes will be published to other sub-system members that have subscribed to receive these changes.

Scenario 4: Making artifacts available to the collaborators. Development team members working on each sub-system can publish their requirement specifications, API's, source code, test cases, test results etc. using a Web Service. The collaborator agents are notified of any changes made to these services. When need arises an agent can connect to a Web Service and retrieve required information. For example agent A, who is interested in updates from agent B and C, receives the notification of updates from B and C. When needed, agent A can make use of that information.

Scenario 5: Details associated with accomplishing different types of requests. Recall that Figure 1 shows that each collaborator agent can perform various services such as display process model, show API, download source code, run test cases etc. Here we describe how a collaborator agent can run test cases in a distributed environment. Assume that there are three agents belonging to three different sub-groups. Assume that agent A has to test the modules developed by B and C (as A's module interfaces both B and C). Agent A has the basic knowledge of their dependencies and can only test when both B and C have notified that their code is ready to be tested. When both the notifications are received, Agent A requests and receives the API documentation from both B and C. Agent A tests the modules developed by B and C which are exposed as Web Services and sends the results (bug report) to B and C. B and C can resolve the issues raised by A. Here we are assuming that B and C have not made any changes to their interfaces. If B has changed the interface for the module that is being developed, then A should modify the test cases and B should incorporate the changes in the Web Service that is exposed for A to test.

3.3 Infrastructural Components

In our framework the communication between agents takes place by using the infrastructure provided by the OPAL framework. Each collaborative worker in our system is represented by an agent. Each of these agents is made up of micro-agents [7]. Each micro-agent can perform certain roles. These roles could be displaying the user interface (UI micro-agent), providing communication (communications micro-agent) and process information (process micro-agent). The agents send each other messages, the contents of which are usually text-based. In our system we also use agents to execute process models.

This approach is open and scalable, since new participants may easily join the collaboration environment by registering themselves with the project moderator. The newly joined participants can interact with other team members as long as they use the agent-based infrastructure.

4 Conclusions and Future Work

In this paper we have described how an agent-based system can be used to facilitate a collaborative P2P work environment. Using different scenarios, we have demonstrated how agents can be used to coordinate, collaborate, and communicate with each other in the context of a distributed software development environment, such as an open source project. This paper reports work in progress. We acknowledge that not all possible scenarios in distributed work environment have been accommodated. In the future we plan to port the system, so that it can make use of PDAs while keeping in mind the limited capabilities of smaller devices [10].

References

1. Gutwin, C., Penner, R., Schneider, K.: Group awareness in distributed software development. In: CSCW 2004: Proceedings of the 2004 ACM conference on Computer supported cooperative work, pp. 72–81. ACM Press, New York (2004)
2. Froehlich, J., Dourish, P.: Unifying artifacts and activities in a visual tool for distributed software development teams. In: ICSE, pp. 387–396 (2004)
3. Guck, R.: Managing Distributed Software Development (2006), http://www.stickyminds.com/
4. Purvis, M.K., Cranefield, S., Nowostawski, M., Carter, D.: Opal: A Multi-Level Infrastructure for Agent-Oriented Software Development. In: The information science discussion paper series no 2002/01, Department of Information Science, University of Otago, Dunedin, New Zealand (2002)
5. Jensen, K.: Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Volume 1: Basic Concepts. In: EATCS Monographs on Theoretical Computer Science, Springer, Heidelberg (1992)
6. Bradshaw, J.: An Introduction to Software Agents. In: Bradshaw, J. (ed.) Software Agents, pp. 3–46. MIT Press, Cambridge (1997)

7. Nowostawski, Mariusz Purvis, M.K., Cranefield, S.: KEA - Multi-level Agent Infrastructure. In: Dunin-Keplicz, B., Nawarecki, E. (eds.) CEEMAS 2001. LNCS (LNAI), vol. 2296, pp. 355–362. Springer, Heidelberg (2002)
8. Nowostawski, M.: JFern - Java-based Petri Net framework (2003), http://sourceforge.net/projects/jfern/
9. Purvis, M., Purvis, M., Haidar, A., Savarimuthu, B.T.R.: A distributed workflow system with autonomous components. In: Barley, M.W., Kasabov, N. (eds.) PRIMA 2004. LNCS (LNAI), vol. 3371, pp. 193–205. Springer, Heidelberg (2005)
10. Purvis, M., Garside, N., Cranefield, S., Nowostawski, M., Oliveira, M.: Multi-agent System Technology for P2P Applications on Small Portable Devices. In: Moro, G., Bergamaschi, S., Aberer, K. (eds.) AP2PC 2004. LNCS (LNAI), vol. 3601, Springer, Heidelberg (2005)