An Interaction Support Mechanism in Software Development

Makoto Matsushita[†]
matusita@ics.es.osaka-u.ac.jp

Hajimu Iida[‡]
iida@itc.aist-nara.ac.jp

Katsuro Inoue[†]
inoue@ics.es.osaka-u.ac.jp

[†]Osaka University
Dept. of Information and Computer Sciences
Faculty of Engineering Science
1-3 Machikaneyama, Toyonaka
Osaka 560 JAPAN

[‡]Nara Institute of Science and Technology
Information Technology Center
8916-5 Takayama, Ikoma
Nara 630-01 JAPAN

Abstract

This paper proposes a new modeling method of interactions in software development process, which focuses on the interactions among the elements of the process, and a new software development environment based on the model.

In this method, interactions in software process is modeled as a set of agents and communication channels. Agent acts interaction between other agents with channels. Channels are classified by their contents and types of interaction.

A prototype of supporting environment for the software development which stands on the model is also developed. The environment consists of proxy program for agent and integrated communication server, which provides mechanism for interaction, execution of process, and user navigation.

1 Introduction

Recently software is being developed in more and more distributed ways, spreaded to the network[1]. In this circumstance, understanding a progress of development and establishing communications among development workers are very important issues, which were not problematic in the traditional centralized software development environment.

Researches on the software process has emerged to solve such problems. They clarifies the development procedures, and identifies the relation between the procedures[3, 11, 2]. The software development procedures and associated development environments are reorganized based on the software processes. There are various fields in research areas of software process, including process programming[12] which precisely models and describes software development procedures, and process-centered software development environment[4, 5, 13] which supports worker's tasks based on the process definitions.

In a software process, there are lots of interactions, including conversation between the users, using application by the users and delivering products from a developer to a user. Software development will progress while these interactions are made. To support interactions among various tasks in software development, however, the process models already proposed and their process-centered development environments are not adequate, since the models do not consider topics of interaction, but only consider types or forms of the interactions.

In this paper, we propose a new process model focusing on the interaction with topics. Also, we will design an environment supporting the interaction[9]. In this model, the interactions in a software development are modeled as a set of agents and communication channels with the topic of interaction.

In Section 2, we discuss the interaction in the software process. In Section 3, we introduce a new modeling method of the interactions in a software process. In Section 4, we describe a development environment based on the model. Finally, in Section 5, we present our conclusions.

2 Interactions in software process

This section discusses software developments on software processes and interactions.

2.1 Background: Process support for software development

There are many attempts of describing elements of software development procedures as software processes. Once the current development procedure is described as a software process, defining organization's standard process or applying the process to the development environment would be easily established. The process descriptions can be also used to measure development progress in the quantitative and visible ways.

Formally described software processes would make it possible to execute or enact automatically the description. Also, the formalized process could be the bases of the simulation of software development environment.

In general, it is unrealistic to employ fully automated execution of the software process. Thus, we will use partially automated execution method of the process.

2.2 Interactions in software development

In software development, there are lots of types of interactions, including "product send/receive", "notification of start/end of a job", "query or reply", "conversation with co-developers", and so on. The contents of interaction widely vary.

Consider following scenario of software development environment (Figure 1).

> An application is being developed at a single site. It is composed of common libraries and other parts using the common libraries.
>
> There are three common libraries (*libR, libS, and libT*) to develop. There are three developers (*A, B, and C*). Each library is developed by two or more developers of the three. There is a "responsible developer" for each library. The responsible developer has to remember what is done in the library development.
>
> There are four users (*X, Y, Z and W*), and each user uses one or more libraries. The users may want to know the announcement of new releases of the libraries. They also ask some questions to the library developers, and exchange messages with other library users.

The following interactions may exist under this scenario.

- A user may try to get a new library.
  If a user wants to use a library which is not used before, he or she must notify to the developer of the library.

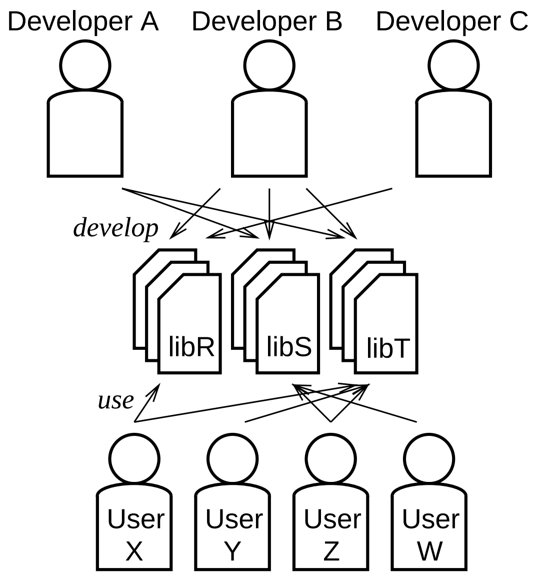

Figure 1. Library Development Example

- If bugs are found, they should be reported to the developers.
  The user who found the bug in the library should notify to the developers of the library, and he or she will receive the result of the bug fix.

- Users may send questions.
  The user's questions on the functions or usages of the library are sent to the developer of the library.

This example assumes a small development team; however, we can see many types of interactions, including file transfer and conversations.

This paper proposes "Process-Centered Interaction Model", for supporting message- or file-based interactions in software development environment, such as "product send/receive", "notification of start/end of a job", "query and reply", "conversastion with co-developers", and so on. This model is based on software process description for interaction template. In the model, we employ "WorkAgent" which represents the subject of the interaction. The interaction is established as a "Channel".

2.3 Related work

Process-centered development environments already proposed and implemented mostly treat the interactions as parameter passing or message passing. Under these environments, we generally need, associated with the interaction information such as “who is the partner of the interaction”. Otherwise, it is difficult to be used as a framework of supporting interaction, since abstraction level of interaction in already proposed process-centered environments is too high.

On the other hand, if we consider to use some existing communication support tool for our purpose, following problems cannot be disregarded:

- With email based tools, there is difficulty of ensuring “to deliver to a person who should get the email”. Moreover, email is not suitable for immediate conversations.
- With conversation tools such as talk/phone, it is good for conversation of course. For sending large amount of information such as file, however, this kind of tool is not suitable. Controlling and tracing communication are also difficult.
- With groupware base tool such as [7, 6, 8, 10], it would be useful as a distributed database. However, if it is an email-based environment, it is not suitable for conversations. If it is a voice/video-based communication tool, it would be hard to trace contents of interaction.

3 Process-centered interaction model

In this section, we introduce the 'Process-Centered Interaction Model'. The overview of the model is presented at first. Then we'll see the details of “WorkAgent” and “Channel”, that play important roles in the model. We also show an example of description of interactions schema and representation of interaction based on the schema.

3.1 Required features

At first, we specify required feature of the model to describe/support interactions in software development processes as follows:

- Process-Centered:
  The largest assumption for our model is that the way of the work (process model) determines the optimal way of the interaction, that is, the structure of the process must be directly reflected to the structure of interaction.
- Actor sensitive:
  The model must be sensitive to the subject of interaction (performers of interaction), because we have to determine the concerning members of every interaction.
- Information sensitive:
  The model must be sensitive also to the content of information treated during the interaction, because these kinds of information are quite important *product* of the process to make effective management of the process.
- Simplicity:
  We assume this model as one component of the generic process model which totally supports the whole activities of software development. Therefore, interaction model should have fine modularity and simplicity to establish the integration with other components.

3.2 Model definition

3.2.1 Overview

At first, we define a software process as a set of *tasks*. Every task is defined as a sequence of activities. An activity is an atomic action in the software process, and every task may communicate with other tasks by sending messages.

To represent required feature listed in section 3.1, we employ the object-oriented model as a base model. We introduce two classes “WorkAgent” and “Channel” to be used as components of interaction model construction.

WorkAgent is an entity which produces events treated as interactions, and its instance may be a certain activity to be executed by a human, a program tool, or some external systems.

Channel is a virtual path of interactions classified based on their contents and types. Its instance may be generated to represent some topics in the software process, such as “library information” or “testing results”.

WorkAgent acts interaction between other WorkAgents with Channels, achieved by *connecting to* Channels. WorkAgent can send/receive information to/from Channels.

Formally, we define our process-centered interaction model as follows:

```
Interaction ::= [set of WorkAgents, set of Channels,
                 ConnectionInfo]
WorkAgent ::= finite state machine
Channel ::= Communication channel for WorkAgent
ConnectionInfo ::= Mapping function from Channel
                     to set of WorkAgents
```

In following sections, we explain the details of WorkAgent and Channel.

3.2.2 WorkAgent

From the aspect of interaction modeling, every task can be regarded as a producer of interactions. Therefore, we model a task as an object called WorkAgent. In other words, WorkAgent produces a series of messages during its execution.

The definition of WorkAgent is as follows:

```
WorkAgent ::= [Name, a set of Attribute, a set of Taskbody]
Name ::= string
Attribute ::= [Attrname, Attrvalue]
Attrname ::= string
(value of Attrname is defined by each process)
Attrvalue ::= scalar value
(range of Attrvalue is defined by each process)
Taskbody ::= [Precondition, Activity]
Precondition ::= A boolean function (domain is input data)
Activity ::= A sequence of procedure for WorkAgent
(Human work, Tool invocation, Message passing, etc)
```

WorkAgent has several properties such as Name, Attributes, and Taskbody.

Name is used to identify WorkAgent. Attributes are set of enumerate-type variables. Attributes are used to specify the characteristic of the role of WorkAgent. Therefore, every enumerate ranges are defined according to every process to be modeled. Taskbody is a sequence of atomic activities, and it is guarded by a Precondition to be satisfied before execution. Atomic activities are determined according to every process and WorkAgent.

3.2.3 Channel

The interaction is defined as a series of messages which are generated by WorkAgents in the software process. These interactions can be categorized based on the type of information and concerning WorkAgents.

Therefore, there are some relations among tasks to specify such category of the messages they exchange. Since several attributes associate to these relations, we model it as a class called Channel.

The definition of Channel is as follows:

```
Channel ::= [Name, a set of Attribute, Closure, a set of Infotype]
Name ::= string
Attribute ::= [Attrname, Attrvalue]
Attrname ::= string
(value of Attrname is defined by each process)
Attrvalue ::= scalar value
(range of Attrvalue is defined by each process)
Closure ::= [Inclosure, Exclosure]
Inclosure ::= a pair of Attributes of WorkAgent
(Definition of Exclosure is the same as above)
Infotype ::= [Typename, a set of FieldDecl]
Typename ::= string
FieldDecl ::= [Fieldid, Fieldtype]
Fieldid ::= string
Fieldtype ::= string, file, bool, etc.
```

Like WorkAgent, Channel has several properties such as Name, Attributes, Closure, Infotype. Name and Attribute have same utilization as WorkAgent's ones.

Closure defines the range of message distribution in the form of WorkAgent type in order to control interactions. Closure consists of Inclosure and Exclosure: Inclosure specifies the WorkAgents which *must be connected to* this channel, and Exclosure specifies the WorkAgents which *must not be connected to* this channel. Infotype defines types of the messages in the Channel. Infotype consists of Typename and a set of FieldDecl: Typename specifies name of the message type, and FieldDecl declares the contents of the messages.

Connecting or disconnecting action must be made before or after use of a channel by WorkAgents to perform interaction each other through the channel.

3.3 Example of interaction expressed by model

This section describes how interaction schema is expressed by the model, and how WorkAgents and Channels work, by using an example case of library development visited before. Figure 2 is an image of interaction model representation for the library development.

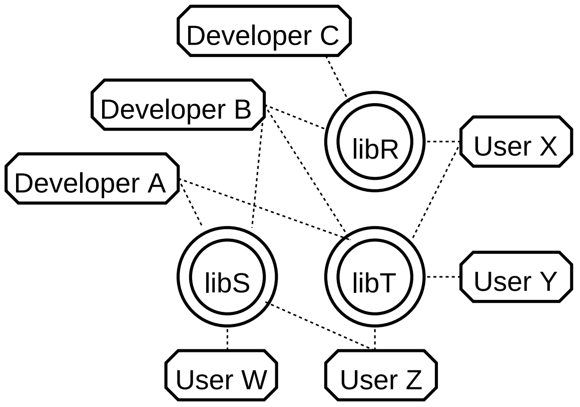

Figure 2. Library Development Example: model mapping

Rectangles which round off the corner represent WorkAgents, and double circles represent Channels. Dotted lines show connectivity among Channel and WorkAgent. There are seven WorkAgents and three Channels in this figure. Now, we show the detailed steps of the description.

3.3.1 Defining WorkAgents and Channels

In this example, it seems that developers and users of libraries interact with each other. In figure 2, developers and users are mapped to WorkAgents. Since the contents of interactions will be related to the libraries which is developed/used, library names are mapped to Channels.

Every WorkAgent has connections to the Channels which may concern, i.e., developers' WorkAgents connect to library Channel of which they develop, users' WorkAgents connect to library Channel of which they use.

3.3.2 Describeing WorkAgents

Figure 3 shows a sample description of WorkAgent for Developer *A*.

```
define workagent Developer˙A
begin
   attribute begin
      role: develop;
      primary: S;
   end
   vars begin
      bugbuffer: array of bug-report;
      qbuffer: array of question;
      res: bool;
   end
   initaction begin
      join(libS);
      join(libT);
   end
   primitive add consult, debugging, makeans;
   action begin
      channel libS begin
         bug-report:
            push(bugbuffer, info);
            res = consult(info);
            if (res == True) –
               send(libS, debugging(info));
            ˝
         question:
            push(qbuffer, info);
            send(libS, makeans(info));
      end
      channel libT begin
         bug-report:
            res = consult(info);
            if (res == True) –
               send(libS, debugging(info));
            ˝
         question:
            send(libS, makeans(info));
      end
   end
end
```

Figure 3. Description of WorkAgent

As attribute values of Developer *A*, two attributes "role" and "primary" are defined. "Role" means "role of this WorkAgent in a process", and "developer" or "user" is valid for its value. "Primary" means "most important library for this agent", i.e., responsible library for developers or most concerned library for users, and a name of the library is valid.

In order to specify several factors establishing the control of Agent's behavior, sections for "control variable definition", "initial action definition", and "external (primitive) activity declaration" are introduced to our syntax.

Variable definition is for defining variable which is used in the description to keep values of information, interaction, etc. Initialization section describes which actions are needed when this WorkAgent begins to work. In this example, joining two channels are specified. Primitive tasks associated to the Agent is also declared, however, its specific actions are not described since it is out of our model's scope.

3.3.3 Describeing Channels

Figure 4 shows a sample description of Channel *libS*.

```
define channel libS
begin
   attribute begin
      libname: S;
   end
   restriction
      in: (develop, S)
   infotype begin
      announce:
        (topic: string, contents: file);
      bug-report:
        (topic: string, contents: file);
      question:
        (topic: string, body: file, keyword: string);
      answer:
        (topic: string, body: string);
   end
end
```

Figure 4. Description of WorkAgent

Same as WorkAgent, attributes of Channel must be defined. In this example, only one attribute "libname" is defined. Attribute "libname" is used for the library name which is focused by this Channel. Closure of this Channel is defined in "restriction" section.

This example shows that this channel has an Inclosure restriction of "developer of libS must be connected to this Channel". There is four Typename in this Channel, announcement, bug-report, question, and answer. Each Typename has several field, i.e., topic string, message file, and keyword strings of message, however, its implementation of information type are not described, since out model is not intended to describe such information.

3.3.4 Behavior of WorkAgents and Channels

We have defined all WorkAgents, Channels, and their connectivity. Now we will see how these description work in the library development with following scenarios.

Scenario 1: Get a new library Suppose user *Y* is going to use new library *libR*. User *Y*'s WorkAgent simply acts "join(*libR*)". Channel connectivity of user *Y*'s WorkAgent is which libraries are used by user *Y*. Of course, if User *Y* decided not to use *libT*, do "leave(*libT*)" to disconnect Channel *libT*.

Scenario 2: Finding a bug Suppose developer *B* find a bug in *libS*. Developer *B* sends a bug-report to Channel *libS*. This bug-report will be sent to all WorkAgents connected with Channel *libS*, i.e., user *Z*, user *W*, and developer *A*. Since developer *A* has responsibility to the development of *libS*, he or she stores this bug-report to buffer, and consults if this is a true bug. If developer *A* recognizes that it is a true bug, he or she will do debugging and sends its result to Channel *libS*.

Not only developer *B*, but also all WorkAgents connected to Channel *libS* can send a bug-report, and it will be accepted to responsible person (WorkAgents) of *libS*. Moreover, developer *A*'s reply will be received by all WorkAgents connected to Channel *libS*.

Scenario 3: Sending a question Like "finding a bug" scenario, any questions should be received to all developers, and all answers should be correctly sent to the users. These questions and answers are shared with WorkAgents connected to the Channel, therefore, we can avoid to send duplicated questions.

4 Development environment based on interaction model

This section describes a supporting environment for the software development. Based on the interaction model described in section 3, at the beginning, an overview of the supporting environment is described. Then we will show an implementation of the environment.

4.1 Overview

The environment based on our model provides the following features.

- Interaction services between WorkAgents
  WorkAgents and Channels defined by the model are actually implemented under a networked environment.
- Execution support for WorkAgent
  If WorkAgent jobs is executable, it is automatically performed by the environment.
- Navigation or tool invocation for user
  If the user is responsible for the job progress of a WorkAgent, the environment invokes the tools needed for the jobs. Also the environment navigates the job schedules by merging information of WorkAgents.

4.2 Environment architecture

Figure 5 illustrates the overview of the supporting environment.

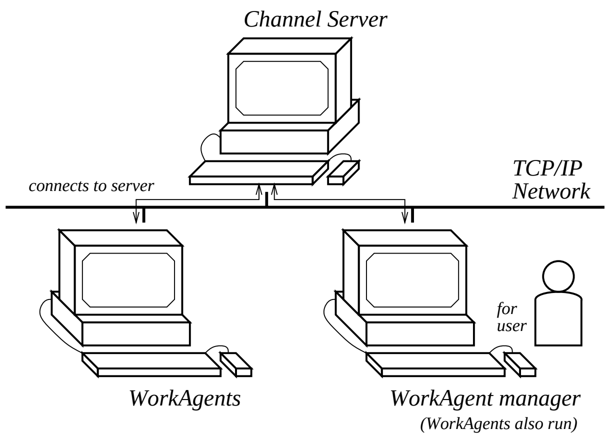

Figure 5. Overview of supporting environment

We assume that the environment is used under a small networking environment. The environment is composed of the following parts.

- WorkAgent program
- Channel server
- TCP/IP network connection
- WorkAgent manager

Each WorkAgent is built with a WorkAgent program. The WorkAgent program reads a WorkAgent definition before it starts. All Channels are controlled by the Channel server. The WorkAgent program is connected to the Channel server by the TCP/IP connection over the network.

There is a program called WorkAgent manager, which controls one or more WorkAgent programs. It also has an interface to the users. The WorkAgent manager shows statistics of the WorkAgent program, and supports execution of the WorkAgent tasks for users, such as tool invocations.

The WorkAgents which are not managed by the WorkAgent manager act themselves. They would work with some product management tool or project management tool, which will invoke another (external) programs, execute pre-defined jobs, and return job results to the Channel server.

The Channel server manages channel connectivity between WorkAgents, sends information to appropriate WorkAgents, and records an interaction history. The Recorded history is used to analyze and check progress of development. An experimental implementation of the WorkAgent program and the Channel server has been already completed.

4.3 Experimental implementation

The experimental implementation of the WorkAgent program runs on BSD UNIX, written in C. This WorkAgent program simply reads definition, connects the Channel server, and performs the interaction; thus porting this program to other environments such as X Window System or to other languages such as Perl and Tcl/Tk is easy.

The current WorkAgent program has a feature of sending/receiving simple data (file and short statement), and has terminal–oriented user interface (user can operate with some commands) since WorkAgent manager to support navigation and tool invocation for users is not yet implemented. If a WorkAgent program receives a file from a Channel, it will automatically store the file to the disk and inform the user of the data arrival. When it receives a short statement, it will display the statement on the screen.

The Channel server is experimentally implemented on BSD UNIX also. This Channel server program consists several parts; socket processing engine, Channel management table, socket management table, and an interface for maintenance.

Following example is actually applied to the experimental system of the WorkAgent program and the Channel server. In this example, we use the same scenario of software development environment which is considered in section 3.

> Suppose developer *B* finds bugs in *libS*. He or she operates his or her WorkAgent program to send a file of bug-report to a Channel *libS*. As soon as WorkAgents for user *Z*, *W*, and developer *A* receive the file, the WorkAgent program stores the received file to a disk, and notifies the receipt to the user.
>
> The WorkAgent program for developer *A* also stores the file to the bug-report buffer. Developer *A* examines the bug-report, decides this is a bug or not, and sends an answer to the Channel *libS* using his or her WorkAgent program. The answer will be received by all WorkAgent program connected to the Channel, and it will be displayed to the users of *libS*.

We found that the scenario works completely on the experimental system.

5 Conclusion

In this paper, we propose "Process-Centered Interaction Model", for supporting interactions in software development environment. The model consists of two components, "WorkAgent" and "Channel", to establish the interaction. In this model, the interactions are treated as event sequences which are produced by WorkAgents and sent through Channels. With this model, we see that the interactions among the users are clearly identified and described.

Also we have designed an experimental environment for the software development which is based on the model. The environment provides interaction service, execution support, navigation and tool invocation to the users. Using the experimental implementation of the environment, the users easily interact each other, i.e., sending/receiving messages or files.

As a further work, validation of our model and environment though experiments, and verification, consistency, and error detection mechanism of our model is needed. Integrity of the whole described model is possibly done by checking Channels required by WorkAgents'. Also, we plan to design an improved communication model with more intelligent interaction.

References

[1] M. Aoyama. Distributed development environment: A new paradigm of software development environments. *Journal of IPSJ*, 33(1):2–13, 1992. in Japanese.

[2] Information Processing Society of Japan. *Proceedings of Software Process Symposium*, 1994.

[3] K. Inoue. Trends of research area about software process. *Japan Society for Software Science and Technology Technical Report*, 95(SP-2-1):1–10, 1995. in Japanese.

[4] G. Kaiser, P. Feiler, and S. Popovich. Intelligent assistance for software development maintenance. *IEEE Software*, pages 40–49, May 1988.

[5] T. Katayama. A hierarchical and functional software process description and its enaction. In *Proceedings of 11th International Conference on Software Engineering*, pages 343–352, 1989.

[6] K. Lai and T. Malone. Object lens: A "spreadsheet " for cooperative work. In *Proceedings of Computer Supported Communication Work '88*, pages 115–124, 1988.

[7] T. Malone, K. Grant, K. Lai, R. Rao, and D. Rosenblitt. Semistructured messages are surprisingly useful for computer-supported coordination. *ACM Transactions on Office Information Systems*, 5(2):115–131, 1987.

[8] T. Malone, K. Lai, and C. Fry. Experiments with oval: A radically tailorable tool for cooperative work. In *Proceedings of Computer Supported Communication Work '92*, pages 289–297, 1992.

[9] M. Matsushita, H. Iida, K. Inoue, and K. Torii. An interaction support mechanism in software development processes. *Information Processing Society of Japan Technical Report*, 95(SE-102):165–170, 1995. in Japanese.

[10] Y. Matsushita. To realize human-oriented groupware. *Journal of Information Processing Society of Japan*, 34(8):984–993, 1993.

[11] K. Ochimizu. Current status of research on software process. *Japan Society for Software Science and Technology Technical Report*, 93(SP-1-1):1–11, 1993. in Japanese.

[12] L. Osterweil. Software processes are software too. In *Proceedings of 9th International Conference on Software Engineering*, pages 2–13, 1987.

[13] S. Sutton, D. Heimbigner, and L. Osterweil. Language constructs for managing change in proces-centered environments. In *Proceedings of 9th International Conference on Software Engineering*, pages 328–338, 1987.