Distributed Process Management System Based on Object-Centered Process Modeling

Makoto Matsushita[†] Makoto Oshita[†]
Hajimu Iida[‡] Katsuro Inoue[†]

[†]Osaka University [‡]Nara Institute of Science and Technology

Abstract. Most of process-centered software engineering environments and those languages focus on *how to produce the product*. However, recent software development tend to require to focus on *what should be made*, because of emergence of various types of software developments; e.g., software reuse, component-based composition, and so on. To achieve this, we propose in this paper a new development environment named *MonoProcess/SME*, which is based on an object-centered software process model *MonoProcess* we also propose. MonoProcess consists of a set of objects which represent artifacts and resources in the software development. An object has attributes and methods, which represent characteristics and operations of the object, respectively. MonoProcess/SME is an software development environment for project management and development support, using the idea of MonoProcess modeling. MonoProcess/SME illustrates software development environment as it is, and provides an environment for software process execution, management, and improvement.

This paper is submitted to WWCA98.
Correspondence: Makoto Matsushita, Division of Software Science, Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University, 1-3 Machikayenama, Toyonaka, Osaka 560, Japan.
Phone: +81-6-850-6571, Facsimile: +81-6-850-6574, Email: matusita@ics.es.osaka-u.ac.jp

1 Introduction

Software process description and its enaction help the software development to proceed effectively and to produce high quality software[16, 19, 21]. However, most of process-centered software engineering environments tend to enforce specific types of development activities to the developer. Also, they require proprietary and exclusive systems/environments which are completely different from existing software development environment [3, 4, 5, 7, 23, 24, 25]. Therefore those systems are not yet widely used in real software development.

Moreover, most of these software process languages[2] focus on the description of "how to produce a product"; i.e., a procedure of software development. However, recent software development methods such as object-oriented programming, software reuse, component-based programming mainly focus on "what should be made"; i.e., artifacts in software development environment. In process-centered software engineering environment, these artifacts-centered idea of software development should be supported, to make more effective support for software development.

In this paper, we propose a process management system, MonoProcess/SME, based on our object-centered software process description model, MonoProcess. We have developed a prototype system of MonoProcess/SME, which contains essential features for representing and enacting ISPW-6 example problem[20]. The goal of MonoProcess is to illustrate software development environment as it is, and provide a framework for software process description, management, and improvement.

MonoProcess consists of a set of objects, which are artifacts and resources in the software development. An object consists of attributes and methods. An attribute represents characteristic of the object. A method is a function applied to the objects. MonoProcess provides a feature of grouping any objects, which enables to combine two or more objects into a single object. It also provides object inheritance to share information between objects. Messages to objects, which would activate the methods or attribute accesses are recorded automatically as the operation history of the object. With these features, the status of the software development environment is easily monitored, and thus MonoProcess/SME provides helpful information to the project manager. MonoProcess/SME assumes that it runs under network-based software development environment, and is also used by the developers to help their work.

The major contributions of MonoProcess and MonoProcess/SME are as follows:

- Object-oriented analysis can be imported to process description.
- Granularity of the object representation is easily managed by using the object grouping and inheritance features. Multi-grained operations are established straightforward.
- Any partial information within an object easily extracted.
- The representation of the objects is flexibly changed corresponding to the change of the project structure.

– MonoProcess/SME is easily installed to existing development environments.

This paper is organized as follows. In section 2, we will describe the definitions and features of MonoProcess, and we compare our model to other software process description languages. In section 3, we will explain an idea of MonoProcess/SME, add-on type process management system. Finally, we conclude our work in section 4.

2 MonoProcess

In this section, we introduce our process model named *MonoProcess*. At first, the definition of MonoProcess object is presented. Then we will see how the software process is presented with this model. We also show the feature of MonoProcess.

2.1 Overview

All artifacts and resources in the software development environment is shown as *an Object*. Software development environment is defined as a set of objects. Object O is defined as $\{Ol, A, M\}$, where Ol is an object label, A is a set of attributes, and M is a set of methods. Ol is a unique name of this object, used to specify the object.

An attribute of an object is defined as $\{Al, Av\}$, where Al is an attribute label and Av is an attribute value. An attribute label is a unique name for attribute and it indicates what kind of information is needed to the object. Information itself is represented by the attribute value. The type of the attribute values is a number, string, label, or list of these types. Some attribute label is already defined in MonoProcess to specify object grouping, etc.

A method is defined as $\{Ml, Mv\}$, where Ml is a method label, and Mv is a method function. The method label is a unique name for this method function, showing what operation is done with this method. Actual operation is defined by the method function: a mapping among sets of objects. Operations of the method function are: to refer/change attribute values, to execute methods, to make new objects, to get a list of object in the environment, to get a attribute/method list of certain object, to search objects by attribute/method label/value, to invoke tools or operations to out of the model, to execute numeric/literal/collection operation found in common programming languages, and so on.

Figure 1 is an example of MonoProcess object. This description shows an object of design document named "`.Doc.Design`".

In this example, four attributes (`@Owner`, `@Type`, `@Input`, and `@Location`) are defined, to show information about this object. For example, an attribute `@Owner` shows who is the responsible person of this document, and is defined as the other object which represents the person. There are two methods, `&Edit` for editing this document, and `&View` for viewing this document.

```
Object .Doc.Design def
        Attribute @Owner .Person.Matsushita;
        Attribute @Type "Design Document";
        Attribute @Input (.Doc.Specification .Doc.Schedule);
        Attribute @Location .ShareDisk.Document
        Method &Edit def
                $editor = .caller&GetEditor(@Type);
                if ($editor) {
                        &View;
                        invoke($editor, @Location . "design.doc");
                }
        endMethod
        Method &View def
                $viewer = .caller&GetViewer();
                if ($viewer) {
                        invoke($viewer, @Input);
                }
        endMethod
endObject
```

Fig. 1. Object sample

2.2 Software Development Process

Partial object *Op* can be defined with respect to a certain object *O*. *Op* contains an object label whose prefix is the same as *O*, and it includes subset of attributes/methods of *O*. *Op* contains partial information of the target object *O*, and it represents a typical characteristics of *O*, illustrating what we are interested in.

Now we can define a status object *Os*, the *status* of software development environment, as the set of partial objects. We assume that a status in software development environment can be illustrated with artifacts in an environment. In MonoProcess, development process *P* is defined as a transition sequence of those status objects.

Consider a simple example of three objects, named `.SPEC`, `.CODE`, and `.TEST`, and these represent a specification, source code, and test result respectively. Each object has a same attribute label "`@FINISHED`", which indicates this object has been completed or not. When we define status object `.SAMPLEOBJ` as the collection of attributes `@FINISHED` of `.SPEC`, `.CODE`, and `.TEST`, we can see the process as the transition of `.SAMPLEOBJ`. At first, three of all attributes are set to "false". Next, attribute of `.SPEC` is changed to "true", and then `.CODE` is changed to "true". Finally all attributes are set to "true". Such sequence of `.SAMPLEOBJ` instance is a process.

2.3 Features

MonoProcess has various features to support project management, software development, and cooperation for developers. In this section, we show these features with some examples.

Reference Scope of Objects In general, all objects can refer and/or can be referred to/from other objects. However, each object can set a scope of reference with object own attribute.

Restriction is specified by "group" and "permitted operation". Group specifies a set of objects, identified by group name. Each object can join the group, and it is represented as `@GROUP` attribute. Objects which have no `@GROUP` attribute, or have an attribute but no values are defined, are considered not join any groups defined in other objects.

`@ACCESS` attribute represents allow/deny operation to this object. There are four types of operations, "read attribute", "change attribute", "execute method ", "allow inheriting".

`@GROUP` and `@ACCESS` are the access control for the object itself. Moreover, MonoProcess provides par-attribute or par-method access control, in the same manner of above.

Operation History An operation to an object such as referring attributes is processed by sending a message to the object and activating a method of the object. In the MonoProcess framework, any operation to all objects are recorded as a history[1]. Operation history is stored in an attribute, labeled `@ATTRIBUTE.HISTORY` or in a method, labeled `&METHOD.HISTORY`. The attribute history records a list of labels of the object which operates the attribute, operation time, and the contents of referenced value/changed value. The method history records a list of labels of the object which executes the method, beginning and ending time of the execution, and the results of execution.

These histories are automatically recorded, i.e., reading an attribute `@X` is really achieved after recording history `@X.HISTORY`, and executing `&Y` is really achieved after recording a history `&Y.HISTORY`.

Mapping Between a Model to Real Environment We need mapping between described process model with MonoProcess and the real software development environment. The mapping is achieved by the MonoProcess/SME system, a software development management environment based on MonoProcess framework. User interface of MonoProcess/SME is provided to operate objects themselves. With this interface, the developers can operate objects easily. Periodical search mechanism to files is used to synchronize automatically between

[1] This is possible under our assumption of the granularity of the MonoProcess description. The process descriptions are generally not so in fine granularity or they do not create and delete objects seriously as scientific calculation.

the model to the real environment. While a method is executed, files are modified/referenced or tools are invoked. The changes of the status in the model influence to the real environment.

Arbitrary and Multi-Grained Object Definition In this example (Figure 2), each source code is defined as one object (`.SRC1` and `.SRC2`), and these source codes are a part of one module (`.MODULESRC`).

```
Object .SRC1 def
        ...
        Attribute @Location "host/path/to/src1";
        Method &MAKEOBJ def
                invoke(.COMPILER@Location, @CompileFlags,
.SRC1.OBJ@Components)
        endMethod
        ...
endObject

Object .SRC2 def
        ...
        Attribute @Components "path/to/src2";
        Method &MAKEOBJ def
                invoke(.COMPILER@Location, @CompileFlags,
.SRC2.OBJ@Components)
        endMethod
        ...
endObject

Object .MODULESRC def
        ...
        Attribute @Components (.SRC1 .SRC2);
        Attribute @Location "host/path/to/module";
        Method &MAKEOBJ def
                foreach $Component (@Components) {
                        $Component&MAKEOBJ;
                }
                invoke(.LINKER@Location, @Components, @Location);
        endMethod
        ...
endObject
```

Fig. 2. Multi-grained objects

In this example, `.SRC1`/`.SRC2` and `.MODULESRC` have the same method “`&MAKEOBJ`

". However, the behavior is not the same; `.SRC1&MAKEOBJ` simply performs compilation, however, `.MODULESRC&MAKEOBJ` executes compilation of each source and also linking all the source. In MonoProcess, methods with same label could perform different activities. This means that MonoProcess supports grain sensitive operation.

If a developer decided to divide `.SRC1` into two sources, `.SRC1.MAIN` and `.SRC1.SUB`, he/she executes an object creation by using a special method `.SRC1&FORK` twice and makes new objects, which are managed by `.SRC1` object.

Easy Information Extraction We assume that object `.FOO` has an attribute `.FOO@MAINTAINER`, which represents the responsible person of this object. One day, the responsible person is changed from person A to person B. Later something wrong with `.FOO`, and we want to know which is the person in charge.

In MonoProcess, we may check an attribute `.FOO@MAINTAINER.HISTORY`, to catch up the transition of value `.FOO@MAINTAINER`, since if the responsible person of `.FOO` was changed, i.e., attribute value of `.FOO@MAINTAINER` was changed, `.FOO@MAINTAINER.HISTORY` was also changed.

Assume there is also object `.BAR`, which is the same type of object `.FOO`. If we decide to trace the change of responsible person, we may say:

```
StatusObject .MAINTAINER_STATUS def
        PObject .FOO.MTSTATUS;
        PObject .BAR.MTSTATUS;
endObject
PObject .FOO.MTSTATUS def
        Attribute @MAINTAINER;
endObject
PObject .BAR.MTSTATUS def
        Attribute @MAINTAINER;
endObject
```

In this description, two partial objects, `.FOO.MTSTATUS` and `.BAR.MTSTATUS` are defined, and using these objects we compose a new status object `.MAINTAINER_STATUS`. We can grasp the process by checking the transition of `.MAINTAINER_STATUS` object.

3 Design and Implementation of MonoProcess/SME

3.1 System Design

Figure 3 is the image of MonoProcess/SME system. MonoProcess/SME assumes that there are already in the network-based environment, co-exists with existing environment, and works for both developers and project managers. There are three parts in MonoProcess/SME; repository part, user-interface part, and method engine part.

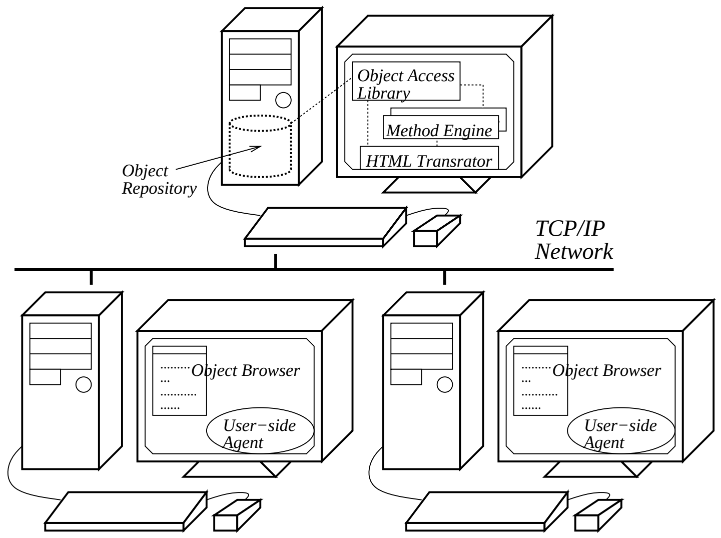

Fig. 3. MonoProcess/SME

Repository The repository part is for object storage management. It handles object descriptions, and stores software development project information such as object operation histories, and is composed of the object repository and the object access library.

The object repository is a storage of objects. It contains the structure information defined in the objects, attribute values, and method descriptions. In the viewpoint of the object repository, MonoProcess/SME can be seen as an object-oriented distributed database system.

The object access library increases accessibility to the object repository. MonoProcess allows to use two or more languages for the method descriptions, and the object access libraries have to be prepared for each language. We currently use language such as Perl-based description language or UNIX shell script.

User-Interface The user-interface part is for interactions between the system and the users. It provides the feature to browse object repository, execute a method, and so on. This part is composed of the object browser, user-side agent, and HTML translator.

The object browser is an user interface of MonoProcess/SME. We employ a web browser as the object browser, to achieve an interface independence to the platform. Translating from information in the object repository to web-browserable form is done in HTML[9] translator. It is embedded in the web server, performing as the interface to the other parts of the system. The object browser may be implemented as a proprietary program, to offer maximum strength of object repository operation.

The user-side agent is a back-end processor, and it exists per each user. It works with the object browser, and collects user-side information, invokes some tools, and supports user-dependent activities.

Method Engine The method engine part is for interpreter of method description. It works as the core engine of process execution. This part uses/is used by other two parts. The method engine is description language dependent part, and it is activated for each method. The method engine uses the object access library to check/modify the object repository.

3.2 Prototype System of MonoProcess/SME

We have already made a prototype system of MonoProcess/SME, to implement MonoProcess and its features listed in section 2.3. We employ Perl-based tiny description language for object description and implementations as the method engine and the object access library. The translator is the module of Apache[1] web server. We have used this prototype system to describe ISPW-6 process modeling example[20] (Figure 4). We also describes a detailed behavior of ISPW-6 example process to confirm that MonoProcess/SME provides the facility of software development support environment.

The description is enacted and the resulting environment established a proper behavior of the ISPW-6 example[14, 26]. Figure 5 is an sample screen-shot about using with MonoProcess/SME environment; activities in software development, such as tool invocation, product passing from an activity to an another activity, and so on is executed within MonoProcess/SME environment.

4 Conclusion

In this paper, we propose a new development environment MonoProcess/SME, which is based on an object-centered software process model MonoProcess.

MonoProcess is based on *Objects*, to represent all artifacts in software development environment. In this model, software process is shown as a state transition of objects. With this model, software process is illustrated clearly; that is powerful capability of the process management.

MonoProcess/SME is a MonoProcess-based software development environment. MonoProcess/SME is an "add-on" for current development environment, and it provides product management and reactive activity execution support. A prototype system of MonoProcess/SME has been implemented and ISPW-6 example has been described and executed on the prototype.

As a further work, a full implementation of MonoProcess/SME has to be completed. Validation of our model and more support for process enaction based on validation is also planned.

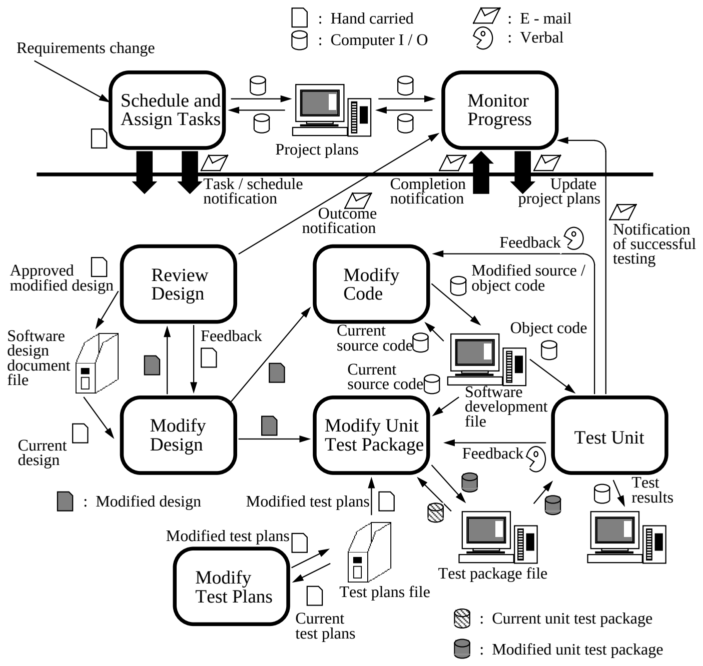

Fig. 4. ISPW6 Process Modeling Example

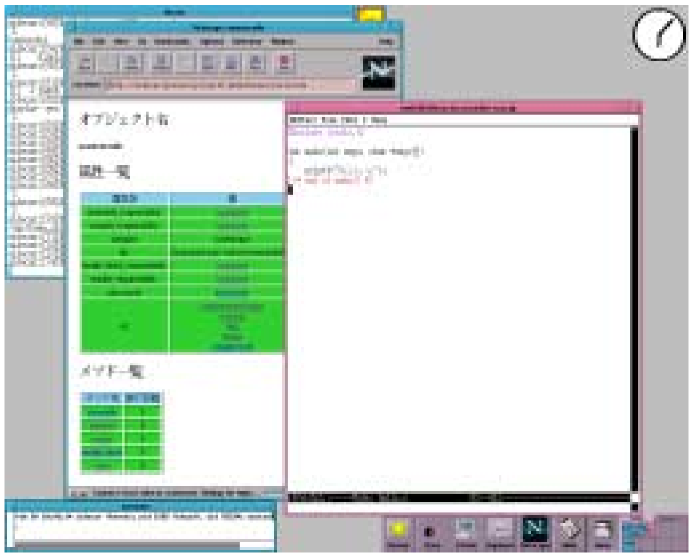

Fig. 5. Screen-shot of Prototype System

Acknowledgement

We are grateful to Tetsuo Yamamoto and Yutaka Fujiwara for their contribution of the development of a prototype of MonoProcess/SME.

References

1. Apache Project. Apache HTTP Server Project. ⟨URL:http://www.apache.org/⟩.
2. P. Armenise, S. Bandinelli, C. Ghezzi, and A. Morzenti. Software Process Representation Language: Survey and Assessment. In *Proceedings of the 4th Conference of Software Engineering and Knowledge Engineering*, pages 455–462, 1992.
3. S. Bandinalli, E. Nitto, and A. Fuggetta. Supporting Cooperation in the SPADE-1 Enviornment. *IEEE Transaction on Software Engineering*, 22(12):841–865, 1996.
4. S. Bandinelli, A. Fuggetta, and C. Ghezzi. Software Process Model Evolution in the SPADE Environment. *IEEE Transactions on Software Engineering*, 19(12):1128–1144, 1993.
5. S. Bandinelli, A. Fuggetta, and S. Grigolli. Process Modeling in-the-large with SLANG. In *Proceedings of the Second International Conference on the Software Process*, pages 75–83, 1993.
6. N. Barghouti. Supporting Cooperation in the MARVEL Process-Centered SDE. In *Proceedings of the 5th ACM SIGSOFT Symposium on Software Development Environments*, pages 21–31, 1992.
7. I. Ben-shaul and G. Kaiser. A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment. In *Proceedings of 16th International Conference on Software Engineering*, pages 179–188, 1994.
8. I. Ben-Shaul, G. Kaiser, and G. Heineman. An Architecture for Multi-User Software Development Environments. In *Proceeding of 5th ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments*, pages 149–158, 1992.
9. T. Berners-Lee and D. Connolly. Hypertext Markup Language – 2.0. RFC1866, 1995.
10. B. Curtis, M. Kellner, and J. Over. Process Modeling. *Communication of the ACM*, 35(9):75–90, 1995.
11. C. Ellis, S. Gibbs, and G. Rein. Groupware, Some Issues and Experiences. *Communications of the ACM*, 34(1):38–58, 1991.
12. C. Fernstrom and C. G. Innvation. PROCESS WEAVER: Adding Process Support to UNIX. In *Proceedings of 2nd International Conference on Software Process*, pages 12–26, 1993.
13. A. Fugetta. Functionality and architecture of PSEEs. *Information and Software Technology*, 38(4):289–293, 1996.
14. Y. Fujiwara. Software Process Modeling with Objects and its Software Development Management System – Repository Implementation and its Application to ISPW-6 Example –. Bachelor Thesis of Department of Information and Computer Sciences, Osaka University, 1997.
15. H. Iida, K. Mimura, K. Inoue, and K. Torii. HAKONIWA: Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model. In *Proceedings of 2nd International Conference on Software Process*, pages 64–74, 1993.
16. K. Inoue. Current Research Activities on Software Process. *Japan Society for Software Science and Technology Technical Re port*, 95(SP-2-1):1–10, 1995.
17. G. Kaiser, P. Feiler, and S. Popovich. Intelligent Assistance of Software Development and Maintenance. *IEEE Software*, pages 40–49, 1988.
18. 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.

19. T. Katayama. Software Processes and Their Research Topics. In *11th Conference Proceedings Japan Society for Software Science and Technology*, pages 433–436, 1994.
20. M. Kellner, P. Feiler, A. Finkelstein, T. Katayama, L. Osterweil, M. Penedo, and H. Rombach. Software Process Modeling Example Problem. In *Proceedings of the 6th International Software Process Workshop*, pages 19–29, 1991.
21. K. Ochimizu. Survey of Research Activities on Software Process. *Journal of Information Processing Society of Japan*, 36(5):379–391, 1995.
22. K. Shimamoto, M. Matsushita, H. Iida, K. Inoue, and K. Aoyama. Process Reporting Tool Using WWW. *Information Processing Sciety of Japan Technical Report*, SE-106:9–16, 1995.
23. S. Sutton Jr., D. Heimbigner, and L. Osterweil. APPL/A: A Language for Software Process Programming. *ACM Transactions on Software Engineering and Methodology*, 4(3):221–286, 1995.
24. P. Tarr and L. Clarke. PLEIADES: An Object Management System for Software Engineering. In *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering*, volume 18, pages 56–70, 1993.
25. R. Taylor, F. Belz, L. Clarke, L. Osterweil, R. Selby, J. Wileden, A. Wolf, and M. Young. Foundations for the Arcadia Environment Architecture. In *Proceedings of 3rd ACM SIGSOFT/SIGPLAN Symposium on Practical Software Development Environments*, pages 1–13, 1988.
26. T. Yamamoto. Software Process Modeling with Objects and its Software Development Management System – Modeling and User-Interface Implementation –. Bachelor Thesis of Department of Information and Computer Sciences, Osaka University, 1997.