LETTER

Experience of Solving Example Problem for Software Process Modeling

Hajimu IIDA†, Yoshihiro OKADA†, *Nonmembers,* Katsuro INOUE† *and* Koji TORII†*, *Members*

SUMMARY Marc Kellner proposed an example problem intending to compare modeling and describing techniques of software process.(6) In this paper, we will describe our approach to understanding and describing the problem, from a process/product relation view, and synchronization/concurrent view. Also, we will show that a description of the problem is translated for execution and its correctness is validated.

key words: *software process modeling, process and product relation, timing chart, petri net, PDL*

1. Introduction

Marc Kellner proposed "Software Modeling Example Problem" as a common basis for discussion and comparison of modeling/describing software development.(6) This problem defines various works caused by a requirement change with more than 10 pages long of English text, and it describes various aspects in the software development, such as processes, products, human resource, and management. Solving this problem is that we read and understand the text, and write those works with our modeling and describing techniques.

The problem is organized with two parts, core problem and extensions. The core problem defines activities of an overall process step, named *Develop Change and Test Unit*. This step is composed of 8 process steps (we may simply say *steps*), namely *Schedule and Assign Tasks, Modify Design, Review Design, Modify Code, Modify Test Plans, Modify Unit Test Package, Test Unit*, and *Monitor Progress*. Each process step is defined with inputs, outputs, responsibility, and constraints. The extensions of the problem define various optional activities, such as scheduling and resource constraints, or process change.

This problem has been used as the target problem in the International Software Process Workshop(1),(2) and Japanese Software Process Workshop,(5) and some researchers undertake to solve this problem in their ways.(7),(8),(10),(11) In this paper, we will describe our approach of understanding and solving the core part of this problem.

Our approach is basically to capture the process of the problem and to represent the process in simple and clear manner. We will study at first a process and product relation for easy understanding overall structure of the problem. Synchronization and concurrency views of the process will be further analyzed with the timing chart and Petri Net models. The correctness of the Petri Net description is validated by translating into an executable program.

2. Process/Product Relation View

Figure 1 depicts the relation of the 8 process steps and their input/output products..

At *Schedule and Assign Tasks* step, a schedule for the work is created, and individual works are assigned to specific staff members. *Modify Design* step involves the modification of the design for the code unit affected by the requirement change. At *Review Design* step, modified design is formally reviewed. *Modify Code* step is the implementation of the design changes into code, and also an compilation of the modified source code into object code. At *Modify Test Plans* step, test plans are modified according to the requirement change. *Modify Unit Test Package* step involves the modification of the actual unit test package for the affected code unit.

Figure 1 is obtained from the description of the inputs and outputs of each process step, and also from the overview description. Modeling process with input and output product relation as this figure gives an intuitive understanding of the problem through the process/product relation view, even to inexperienced developer of this kind of works.

3. Synchronization and Concurrent View

As a next stage of the problem solving, we further analyzed the process step of the problem. We investigate the synchronization and concurrency of the steps which are essentially performed concurrently; however, they were not expressed explicitly in the process/product relation view as Fig. 1.

We capture the overview by making a timing chart

Manuscript received March 5, 1992.

† The authors are with the Faculty of Engineering Science, Osaka University, Toyonaka-shi, 560 Japan.

* Presently, with Advanced Institute of Science and Technology, Nara.

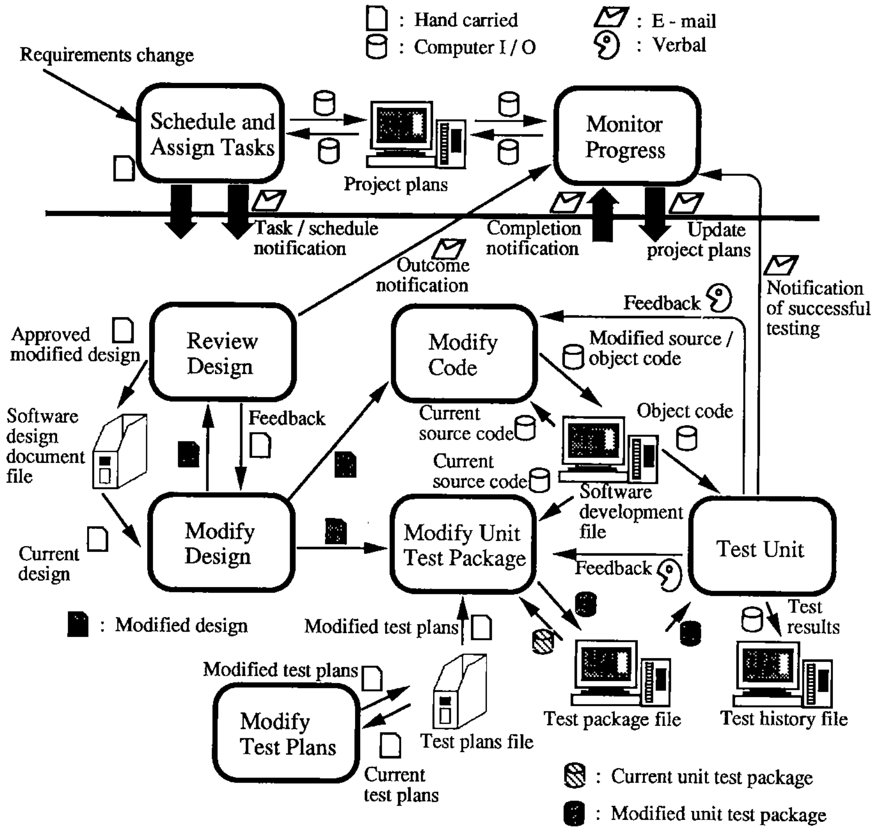


Fig. 1 Process/product relation view.

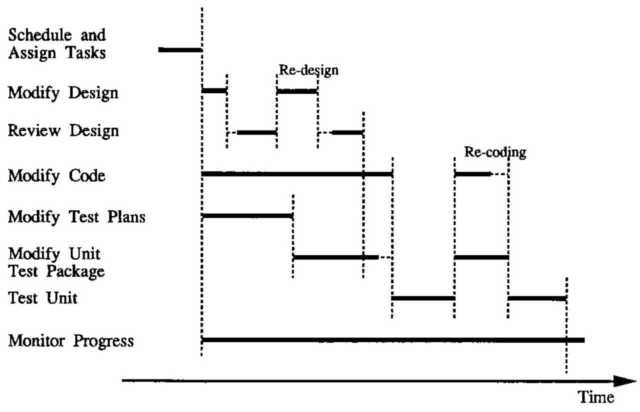


Fig. 2 Synchronization and concurrency view with timing chart.

as shown in Fig. 2. This figure simply shows the active period of each step with solid lines and uncertain period with dotted lines. This timing chart shows characteristics of initiation and termination of each step more clearly, compared with Fig. 1, but it remains some difficulties as follows.

- The steps whose initiations are not defined strictly are not expressed explicitly. For example, *Modify Code* step can be started any time after *Schedule and Assign Tasks* step, but it is not shown well on the timing chart.
- Two steps, any of which can terminate first, but both of which have to terminate before the following steps, would not be described clearly. *Modify Code* step and *Modify Unit Test Package* are such examples.
- The steps whose terminations are dominated by the terminations of other steps are not described explicitly. For example, approval by *Review Design* step is essential for the termination of *Modify Code* step; however, it is not shown explicitly.
- The selection of the next steps among possible candidate steps are not expressed clearly. For example, if *Test Unit* terminates unsuccessfully, then we may proceed either *Modify Code, Modify Unit Test Package*, or both of these.

4. Precise View with Petri Net

To overcome these difficulties, we employed a Petri Net for describing more precisely the synchronization and concurrency of the steps. Figure 3 shows the Petri Net model. In this figure, transitions (represented by vertical bars) mean initiation and termination of the steps. Places (represented by circles) mean the task bodies for steps (they are numbered and named on the figure), or preconditions and postconditions for initiation and termination (no names are

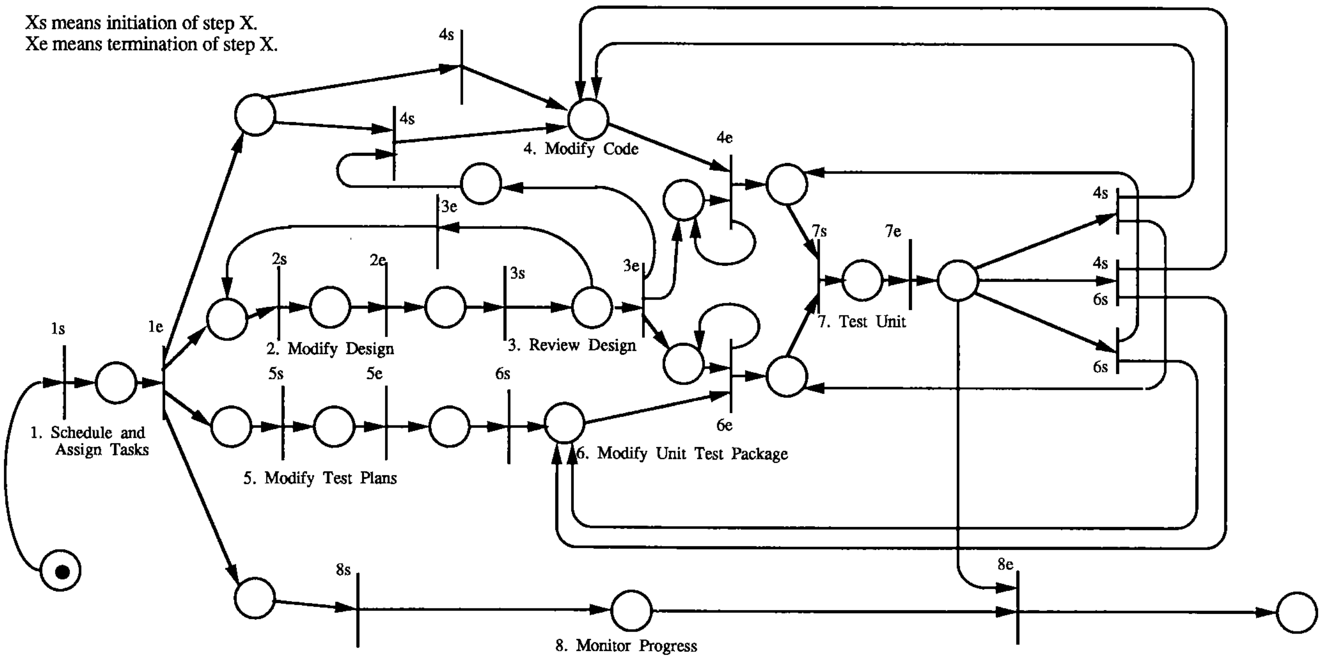


Fig. 3 Synchronization and concurrency view with petri net.

given).

By using this model, we can clarify the following.

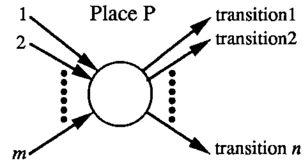
- There happen two cases after *Modify Design and Review Design* steps terminate. One case is to approve the changed design, and another is to repeat *Modify Design* again.
- *Modify Unit Test Package* step starts after *Modify Test Plans*.
- *Modify Code, Modify Test Plans*, and *Modify Unit Test Package* steps can be initiated even without the approval by *Review Design*; however, *Modify Code, Modify Unit Test Package* steps cannot terminate without the approval and termination of *Review Design*.
- When *Test Unit* terminates successfully, *Monitor Progress* step can terminate so that the overall process can terminate.

5. Validating Petri Net Modeling

This Petri Net model has been translated into PDL (Process Description Language) description systematically, and the obtained PDL program is executed by PDL system we have developed.[3],[4] PDL is a functional language, designed to describe various software processes with formal semantic definition and to enact the described processes.

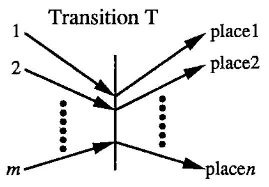
The execution of this PDL program works as the simulation of the Petri Net model, and this would be also a simulation of the original example problem, from the view of the synchronization and concurrency of the steps.

Figures 4 and 5 shows the overview of the translation methods for each place and transition. As shown in Fig. 4, a place is translated into a PDL menu-branch function, which displays a user selectable menu list. The strings in the menu-branch function appears on the display, and if the user select one item with a string, the following place function is executed. The parameter S is a state value containing system status. In the case of more than one output arcs, this user selection



```
P(S) == menubranch({ ["task1",  transition1(exec_p(S))],
                     ["task2",  transition2(exec_p(S))],
                       ...
                     ["taskn",  transitionn(exec_p(S))]},  S) ;
```

Fig. 4 Translation of place.



```
T(S) == if condition1 & condition2
          & ...      & conditionm
        then place1(exec_t(S)) @ place2(exec_t(S))
          @   ... @ placen(exec_t(S))
        else S;
```

Fig. 5 Translation of transition.

operation corresponds to a branch to a possible next step. Since the conditions for selecting the branches are not described on the Petri Net model, we used simple menu fnuctions; however, we may write the conditions for the next steps and embed those conditions into the PDL program, so that some of the selections could be eliminated.

Figure 5 shows the translation of a transition into a PDL function. Each condition represents the termination of the previous steps. If all of those conditions are satisfied, that is, all previous steps terminate, the following steps are initiated. Operator '@' means that the following process steps Place 1 to Place *n* are executed concurrently.

The overall program of PDL contains 96 function definitions including auxiliary ones. This program is considered to be a description of the Software Process Modeling Example Problem, from the view of the synchronization and concurrency of the steps. Execution of this program provides menus on the screen, and opens and closes windows according to the proceed of the steps. We have actually executed this PDL program using our PDL system. We have given this execution program various test cases, by which the interesting interrelation properties among process steps, such as listed in Sect. 4, were validated with respect to the original problem description. This is eventually a validation of the Petri Net model also.

6. Conclusion

We have shown our approach to the solutions of Kellner's Software Process Modeling Example Problem. We have at first modeled the process and product relation using a process/product relation graph, by which the overall structure of the problem can be easily understood. We have next focused on the process steps and shown a timing chart from the view point of synchronization and concurrency. We have further extend the analysis of the synchronization and concurrency with the Petri Net model, and validated our modeling by translation and execution of the PDL program.

The approach taken here is considered to be an essential one to model and describe software processes. Describing the target problem directly with low-level process languages would be difficult; therefore depicting figures of various models to grasp the overall structure of the problem is very important. After the model description is fixed, the concrete and detailed program in the low-level language is systematically obtained. Many other aspects on this example problem such as resource managements, or other target software processes will be modeled and described in the same way.

The approach shown in Ref. (7) is based on an extended data-flow diagram and a state transition diagram. It mainly concerns on the estimation of the schedule time duration. Our approach focuses on the concurrency and synchronization of each process step. In Ref. (8), all the problem description is written with algebraic expressions, by which the expressive power of the algebraic language is shown. The example problem is described using a protocol description language LOTOS in Ref. (10) where the description is executed for simulations, while our approach intends to establish real software development environment under the framework of PDL. For modeling dynamism of software processes, meta operations are introduced in Ref. (11), which are powerful but might degrade the clarity and simplicity of the language.

The PDL program obtained here would be a skeleton of the development support environment for such project as defined in the problem, by adding say, definitions of tools to be used for each step. In this paper, we have presented only limited views of the problem; however, it is necessary to employ many other views to complete the overall description of the problem.

Acknowledgment

We are grateful to Yasuaki Matsunaga of Osaka University for his help of developing models. We are also grateful to Mr. Wataru Nagaoka, Hajimu Umemoto, and Mutuo Sakai of Hitachi Ltd. for helpful discussions.

References

(1) *Proc. of International Software Process Workshop*, IEEE Press, Hakodate, Japan, Oct. 1990.
(2) *Proc. of International Software Process Workshop*, Yountville, CA., Oct. 1991, to be published by IEEE Press.
(3) Inoue, K., Ogihara, T., Kikuno, T. and Torii, K., "A Formal Adaptation Method for Process Descriptions," *Proc. of 11th International Conference on Software Engineering*, Pittsburgh, PA, pp. 145-153, May 1989.
(4) Inoue, K., Ogihara, K., Iida, H., Nitta, M., and Torii, K., "A Functional Language for Enacting Software Processes," *15th Computer Software and Applications Conference '91*, pp. 487-492, Tokyo, 219-224, Sep. 1991.
(5) *Proc. of Japanese Software Process Workshop*, sponsored by Japan Society for Software Science and Technology, Information Processing Society of Japan, and Software Engineers Association, Ito, Japan, Feb. 1991.
(6) Kellner, M., "Software Process Modeling Example Problem", *Private note* on Aug. 9, 1990, also, *Proc. of 1st International Conference on the Software Process,* pp. 176-186, IEEE Press, Redondo Beach, CA, Oct. 1991.
(7) Kellner, M., "Software Process Modeling Support for Management Planning and Control," *Proc. of 1st International Conference on the Software Process*, pp. 8-28, IEEE Press, Redondo Beach, CA, Oct. 1991.
(8) Nakagawa, A. and Futatsugi, K., "An Experimental Solution in OBJ for the ISPW-6 Example Problem," *Proc. of Japanese Software Process Workshop*, Ito, Japan,

Feb. 1991.

(9) Okada, Y., Matsunaga, Y., Iida, H., Inoue, K., Torii, K., Nagaoka, W., Umemoto, H. and Sakai, M., "A Description of Kellner's Software Process Problem," *Proc. of 43th Convention of Information Processing Society of Japan*, 7J-1, p. 5/365-366, Sep. 1991.

(10) Saeki, M., Kaneko, T., and Sakamoto, M., "A Method for Software Process Modeling and Description Using LOTOS," *Proc. of 1st International Conference on the Software Process*, pp. 90-104, IEEE Press, Redondo Beach, CA, Oct. 1991.

(11) Suzuki, M., and Katayama, T., "Meta-Operations in the Process Model HFSP for the Dynamics and Flexibility of Software Processes", *Proc. of 1st International Conference on the Software Process*, pp. 202-217, IEEE Press, Redondo Beach, CA, Oct. 1991.