社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS 94－38 (1994－11)

ペトリネットモデルを用いた ソフトウェアプロセスの記述とその分散実行制御

山口 弘純　岡野 浩三　東野 輝夫　谷口 健一

大阪大学　基礎工学部　情報工学科
大阪府　豊中市　待兼山町 1-3

あらまし　ソフトウェアプロセス全体を実働化する際, プロセスでの各作業内容とそれらの実行順序に加え, 技術者間のデータの交換作業や作業間の同期を考慮する必要がある. しかし, 設計者がそれらを陽に誤りなく記述することは困難であり, 繁雑でもある. そこで, 本稿では, ソフトウェアプロセスの (Kellner の) 標準的例題の全体をそのソフトウェアプロセスで行われる作業内容とそれらの実行順序のみで記述した. そして, その全体記述と, ソフトウェア開発に必要なファイルや文書などの各技術者への分散配置情報から, 各技術者のデータの送受信も含んだ動作記述を, 我々がすでに提案した導出法に従って機械的に導出した. 記述モデルとして, 並列動作等が扱える, レジスタを持ちデータを処理できるように拡張したペトリネットのサブクラス (FC ネット) を用いる. 導出した各技術者の動作記述もペトリネットモデルで記述されるため, 各技術者は, どの作業がどのような順番であるいは並列に行われているかを視覚的に把握することができる. 本研究により, 我々が提案したモデルでソフトウェアプロセスを記述でき, それを分散制御できるような動作記述を自動導出できることが示された. これにより本手法の有用性が確かめられた.

キーワード　ソフトウェアプロセス, ペトリネット, 分散システム, 分散制御

Software Process Description in a Petri Net Model and Its Distributed Execution

Hirozumi YAMAGUCHI, Kozo OKANO, Teruo HIGASHINO and Kenichi TANIGUCHI

Department of Information and Computer Sciences, Osaka University
Machikaneyama 1-3, Toyonaka, Osaka 560, JAPAN
E-mail : {h-yamagu, okano, higashino, taniguchi}@ics.es.osaka-u.ac.jp

Abstract　In order to enact a whole software process description, it is necessary to specify not only the contents of activities of the software process and their temporal ordering but also the communications among the engineers to synchronize or to exchange data values. However, it is very complicated for the designers to describe them correctly. In this paper, we have described only the whole description of the standard process modelling example problem (Kellner's example problem), where only the contents of activities of the software process and temporal ordering of the activities are specified, in an extended Petri net model with a finite number of registers which can treat data values. We have also derived each engineer's individual description including the communications among the engineers automatically from the whole description and a distributed allocation of the files and documents necessary for the software development, using the derivation algorithm which we have already proposed. Since the derived individual descriptions are also described in the same model, each engineer can understand his/her own working flows (including parallel actions). In our research, it has shown that in our proposed model, the software processes can be described, and that we can derive the individual descriptions which can be carried out at a distributed control. The usefulness of our approach is also shown in this example.

Key Words　Software Process, Petri Net, Distributed System, Distributed Control

1 Introduction

Recently, various approaches have been applied to describe formally the "Software Processes" [1, 2, 3, 4, 5, 6, 7, 8]. These approaches are useful for reducing the ambiguity of the software processes, helping to understand the processes, developing systems for the Computer-Supported Cooperative Work (CSCW), and so on.

In general, the above formal description techniques specify each activity of the software processes and the temporal ordering of the activities. However, in order to specify each individual engineer's process, the description must include not only his/her activities and their ordering but also the communications among the other engineers to synchronize or to exchange data values. Such a description becomes complex, if there are many communications. And it is not easy to find errors in it. Therefore, it is desirable that from a description of activities (without communications) of a whole software process and the temporal ordering of the activities (we call it *a whole description*), each engineer's process description can be derived automatically, which specifies his/her activities and communications, and their temporal ordering (we call it *an individual description*).

Some process models have been proposed to describe the whole description and/or the individual description: (1) the programming models such as APPL/A [9], (2) the functional models such as HFSP [3] and PDL [10], (3) the rule-based models such as GRAPPLE [11], MARVEL [12, 13], and Merlin [14], (4) the Petri net based models such as MELMAC [15] and SLANG [16], (5) the LOTOS based models [5, 8, 17] and so on.

In general, since the software processes contain parallel actions and they can be treated as distributed systems, the Petri net based models and LOTOS based models are useful. However, most approaches do not consider the derivation of individual descriptions. In [17], we have proposed a deriving technique using an extended Full LOTOS model (LOTOS/SPD), which can automatically derive correct individual descriptions including communications from a given whole description of the software process. However, in this model, the resources (the files and documents used in the software development) can not be allocated in a distributed environment. So in the model, we assume that all resources are kept in each engineer's work space. We also assume that the modification of resources must be controlled by one engineer and the changed values of the resources are distributed to every engineer, i.e., we have adopted the centralized controlled method. In order to enact software processes in distributed environments, it is desirable that each resource can be allocated to some engineers, and that the modification of the resources can be carried out at a distributed control.

In this paper, we adopt a Petri Net model with Registers (PNR model)[22], in order to treat the distributed resource allocation and to describe the software processes naturally. Using the PNR model, we have described a whole description of the software process modelling example problem by Marc Kellner [19]. We have also derived individual descriptions from the whole description and a resource allocation. In order to derive the individual descriptions, we have used the derivation algorithm in [22].

Our approach has the following advantages. (1) Working flows (including parallel actions) can be understood graphically. Especially in the individual descriptions, each engineer can understand his/her own working flows. (2) The resource allocation can be specified freely. This advantage is very important in application to the software processes, because software developing environments may be varied in a short cycle.

In Section 2, we give the definition of our PNR model. In Section 3, the whole description of Kellner's example problem in the PNR model is explained. In Section 4, we derive individual specifications and compare our approach with others.

2 Petri Net with Registers

We introduce a Petri Net model with Registers (PNR model).

Definition 1 (Petri Net with Registers) *A Petri Net with Registers (PNR) is denoted by a pair* $PNR = (PN, \Sigma)$, *where* Σ *is defined as a 7-tuple* $\Sigma = (G_s, \mathcal{A}, G, R, C, \delta, Init)$.

- PN *is a Petri net* [18], $PN = (P, T, F, W, M_0)$ *(or simply* (N, M_0) *where* N *denotes* (P, T, F, W) *), provided that the net* N *must not contain isolate transitions nor places.*
- G_s *is a finite set of gate symbols.*
- $\mathcal{A}$ *is a finite set of events, whose gates are the elements in* G_s.
- G *is a finite set of guards.*
- R *is a finite set of registers.*
- C *is a finite set of register definition statements.*
- δ *is a function representing the contents of transitions,* $\delta : T \rightarrow G \times \mathcal{A} \times C$.
- $Init$ *is a function specifying the initial values of registers.*

We call the net PN *the underlying net of the* PNR.

A PNR may have some registers R_1,...,R_n. Each R_i is called a register variable. Each transition in a PNR has a label of 3-tuple [a guard, an event, a register definition statement]. An event in the set $\mathcal{A}$ must have one of the the following three forms : $a?x$, $a!E(\ldots)$ and i. The $a?x$ denotes an input event and the variable x represents an input value given from the gate a (if more than one input values are given, it is denoted like $a?x_1, x_2, x_3, \ldots$). The $a!E(\ldots)$ denotes an output event and the value of the expression E is emitted from the gate a. E is an expression which may contain register variables (more than one output values may be emitted like $a!E_1(\ldots), E_2(\ldots), E_3(\ldots), \ldots$). The event i is an internal event which does not execute any input/output. A guard in the set G is a predicate which may contain the register variables and/or input

variables. A register definition statement in the set C has the form $R_{h_1} \leftarrow f_1(\ldots)$, ..., $R_{h_i} \leftarrow f_i(\ldots)$, where each f_j $(1 \leq j \leq i)$ is a function which may contain the register variables and/or input variables.

A transition t is enabled in a $PNR = (PN, \Sigma)$ *iff* t is enabled in the PN and the value of the guard of t is true. If an enabled transition t fires, the event of t is executed, and then the values of registers are changed simultaneously based on its register definition statement. For an enabled transition which executes an input event, we assume that it can not fire until input data are given.

If (1) the event of a transition t is the internal event i, (2) the guard of t is "true" and (3) the register definition statement of t is empty, then we call the transition t an ε-transition.

3 Software Process Example and Its Whole Description

In this section, we explain the "Software Process Modeling Example Problem" given by Marc Kellner in [19], and describe a whole description of the problem in our PNR model.

3.1 Kellner's Software Process Modeling Example

The problem is given by Marc Kellner in [19] as an example to specify the software processes formally where some cooperating work in a distributed environment is described. The problem contains a lot of parallel and selective work. This example problem is used by several researchers for evaluating their modeling abilities. It describes the software processes to modify the design of some software modules, to review the design, to modify the codes and to test them. It consists of a core problem and several optional extensions, and in general, the core problem is used for evaluating modeling abilities. Therefore, we focus on the core problem in this paper.

The software development organization which treats the core problem is described as follows. There is a project team. The project team consists of a Project Manager (denoted by PM) and software engineers, and the engineers are classified "design engineers" and "quality assurance (QA) engineers". The Design Engineers (denoted by DEs) modify the design and the codes, and the QA Engineers (denoted by QEs) modify the test plans and the unit test package, and test the codes. An additional group, which consists of a design engineer, a QA engineer and some software engineers, is needed to review the design (Design Review team, denoted by DR). And a Configuration Control Board (CCB) provides authorizations from outside of the project team.

The work which is described by the whole of the core problem is called "**Develop Change and Test Unit**", which starts when it receives a requirement change from CCB and ends when the unit testing has been successfully completed. This process is decomposed into the following eight processes.

1. **Schedule and Assign Tasks**, that involves the decision of a schedule for the work and assignment of tasks for each engineer.
2. **Modify Design**, that involves the modification of the design.
3. **Review Design**, that involves the formal review of the modified design.
4. **Modify Code**, that involves the implementation of the design modifications into the codes and compilation of the modified source codes into object codes.
5. **Modify Test Plans**, that involves the modification of the test plans.
6. **Modify Unit Test Package**, that involves the modification of the actual unit test package in accordance with the modification to the test plans.
7. **Test Unit**, that involves the application of the unit test package on the modified codes and analysis of the results.
8. **Monitor Progress**, that involves the project manager monitoring progress and status of the work.

3.2 Whole Description in the Petri Net Model

We have described a whole description for the core problem in our restricted PNR model. The restriction to the PNR model to describe the whole description is explained in the next section. Fig. 1 shows the whole description. For the description, we use the gates and registers in accordance with the problem as follows. The gate a represents CCB which gives a requirements change to the project team. The gate b represents the database of "software development files" which preserves the modified codes, and the gate c represents the database of "test package file" which preserves the modified unit test package. The gates d, e, f represent the working records of DEs, QEs and RD, respectively. The contents of registers R_1, ..., R_9 are also defined in Fig. 1.

As mentioned above, we only treat the core problem. However, for the case that we want to make an optional extension where the integrated test of several softwares which have been modified and tested by the project teams can be treated, we should distinguish two types of resources, (1) the resources that only the project team can use and (2) the resources that other project teams can also use. In our model, we represent the former type of resources as registers, and the latter type of resources as gates.

The transition t_1 decides a schedule for the work. The transitions t_2 and t_3 modify and review the design, respectively. The transitions t_8 and t_9 modify the codes. The transitions t_6 and t_7 modify the test plans. The transition t_{10} modifies the unit test package and the transition t_{11} tests the units. Some of those transitions can fire in parallel.

In Fig. 1, if the events are the internal event i, or if the guards are "true", or if the register definition statements

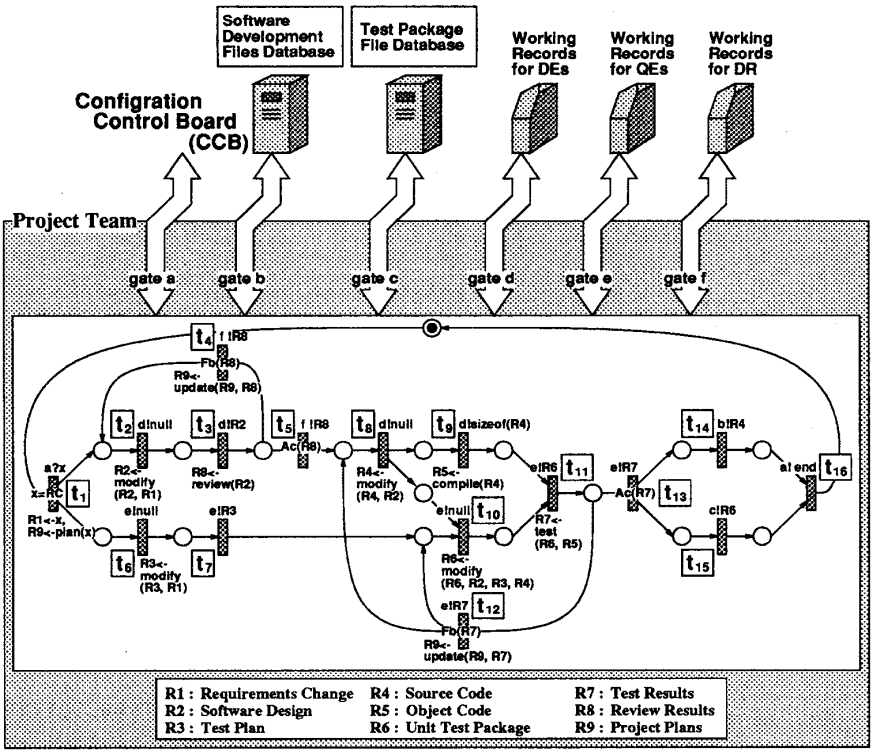

Fig. 1: A Whole Description in the PNR model.

are empty, then they are omitted. The functions used in Fig. 1 are explained in Table 1. Suppose that all the functions are called by values. We can define the details of the functions freely and they do not influence on the derivation algorithm described briefly in the next section.

4 Derivation of Individual Descriptions and Their Distributed Execution

In this section, we explain our derivation algorithm briefly, and derive the individual descriptions from the whole description in the previous section and a resource allocation. For the details of the algorithm, see [22].

4.1 Distributed Environment and Resource Allocation

Hereafter, for simple representations, we regard each engineers' group in the project team as one "engineer". And we call PM, DEs, QEs and DR by their ID numbers, such as engineer 1, engineer 2, engineer 3 and engineer 4. Let ID_k $(1 \leq k \leq 4)$ denote the individual description for the engineer k.

We assume that each communication channel from engineer i to engineer j is modeled as a FIFO queue (queue_{ij}) whose capacity is infinite. We call both sides of the channel the gate g_{ij}. If the engineer i executes an output event "$g_{ij}!d$", then the data d is enqueued to the queue_{ij}. If the engineer j executes an input event "$g_{ij}?x$" and the first element of the queue_{ij} is d, then the data d is dequeued from the queue_{ij} and the value of d is assigned to the input variable x. If there are no elements in the queue_{ij}, then we assume that engineer j cannot execute the input event $g_{ij}?x$.

We also assume that each gate must belong to exactly one of the engineers and that each register may be allocated to more than one engineers. This means a distributed allocation of resources. Let Θ denote such a resource allocation. Fig. 2 denotes a resource allocation Θ. It denotes the following resource allocation.

Engineer 1 (PM)	Engineer 2 (DEs)	Engineer 3 (QEs)	Engineer 4 (DR)
R_1, R_9	R_2, R_4, R_5	R_3, R_6, R_7	R_4, R_8
a	$b,\ d$	$c,\ e$	f

Note that the register R_4 is allocated to both engineers 2 and 4.

Since we assume that each gate must belong to one of four engineers, the engineer who executes the event of each transition t in the whole description can be determined uniquely. We call such an engineer a *responsible engineer* of the transition t, and denote it by $\text{RE}(t)$. Also, we call the responsible engineers of all transitions in $t \bullet \bullet$ *next responsible engineers*, and denote a set of such engineers by $\text{RE}(t \bullet \bullet)$.

4.2 Derivation Algorithm

Basically, for each transition t in a given whole description, we construct a sub-Petri net $SP^k(t)$ for each engineer k that simulates the transition t. An individual description ID_k is constructed by replacing each transition t in the whole description with the corresponding sub-Petri

Table 1: Functions used in the Whole Description.

Function	Contents of Calculation and Return Value
plan(x)	That makes a schedule for the project using x, and returns it.
modify($R_i, R_{j_1}, \ldots R_{j_k}$)	That modifies R_i using $R_{j_1}, \ldots R_{j_k}$ and R_i, and returns the modified R_i.
compile(R_i)	That compiles R_i, and returns the object codes.
update(R_i, R_j)	That updates R_i using R_i and R_j, and returns the updated R_i.
test(R_i, R_j)	That tests R_i using R_j, and returns the test results.
sizeof(R_i)	That calculates the size of R_i, and returns it.
null()	That returns nothing (without any argument).
end()	That returns the notice of the completion (without any argument).
Ac(R_i)	That returns true if R_i has been accepted, else returns false.
Fb(R_i)	That returns true if R_i has been rejected , else returns false.

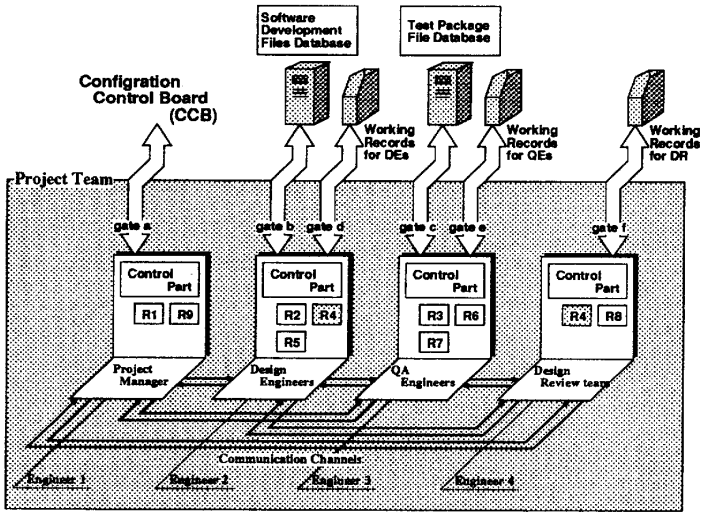

Fig. 2: A Resource Allocation.

net $SP^k(t)$. We restrict the class of the PNR model to describe whole descriptions. We give the following two essential restrictions, and here, we omit additional three non-essential restrictions. (a) A whole description must be modeled as a $PNR = (PN, \Sigma)$ and the PN must be a live and safe free-choice net (FC net) [18] (if the PN is not connected, then each connected net must be a live and safe FC net). (b) There are no *conflicts of registers* in a whole description. In regard to (a), a FC net has a simple selective structure and a live and safe FC net has a useful property for our algorithm. Also, parallel events and non-deterministic selections can be described in FC nets naturally. In regard to (b), the conflict of registers occuers in the case that a transition modifies a register and another one uses the register in parallel. It causes the inconsistency of registers' value between different engineeers. However, Kellner's example problem assumes that there are no resource conflicts in the core problem, as well as in many practical distributed systems.

The implementation principle of each $SP^k(t)$ is following. Now, suppose that for the transitions in p$\bullet$ of a place p with a token at the current marking M, the responsible engineer evaluates these guards, and then chooses non-determinately a transition t to be executed from the enabled transitions.

First, the responsible engineer of the transition t (RE(t)) executes the I/O event of t.

Secondly, RE(t) sends the messages to the related engineers (if neccesary). (a) Each engineer with the registers whose values must be changed in the transition t has to know the values of the registers and inputs necessary for changing its registers' values. Those values are sent from RE(t) as type 1 messages if it has them. If some of those values are not held in RE(t), then RE(t) sends the request messages (type 2 messages) to the engineers who have those registers. (b) Some engineers can change their registers' values by themselves. Type 3 messages are sent to those engineers from RE(t). (c) The next responsible engineers of the transition t (the engineers in RE($t \bullet \bullet$)) should know the values of the registers used in their guards and output events. RE(t) sends type 4 messages to some engineers who have such values.

Thirdly, each engineer who has received the messages works as follows. (a) Each one who has received the type 2 message sends the type 1 messages (including the values of registers) to the engineers who need them. (b) Each one who has received the type 4 message sends type 6 messages (including the values of registers) to some of the next responsible engineers. If such engineers have to change the values of the registers to be transmitted, the type 6 messages must be sent after changing the values of registers. (c) Each one who has received the type 1 or type 3 messages changes its registers' values. RE(t) should change its registers' values after executing the I/O event of the transition t.

Finally, each engineer who has changed its registers' values sends the type 5 messages to all of the next responsible engineers of t. In addition, if RE(t) has never sent the type 1, ..., type 6 messages, then RE(t) sends the type 7 messages to the next responsible engineers of t. These messages are used to inform that the responsible engineer has been changed.

We construct each $SP^k(t)$ in accordance with the above principle (if an engineer k is not concerned with the simulation of t, $SP^k(t)$ is an ε-transition). Then, an individual description ID_k is constructed by replacing each transition t in the whole description with the corresponding sub-Petri net $SP^k(t)$. In our algorithm, the total number of exchanged messages for each simulation of transition is minimized and each simulation is executed with possible concurrence. Also, the methed for removing all ε-transitions, which cannot be removed easily for their roles of synchronous points in the individual descriptions, are proposed. However, for the limitation of space, they are

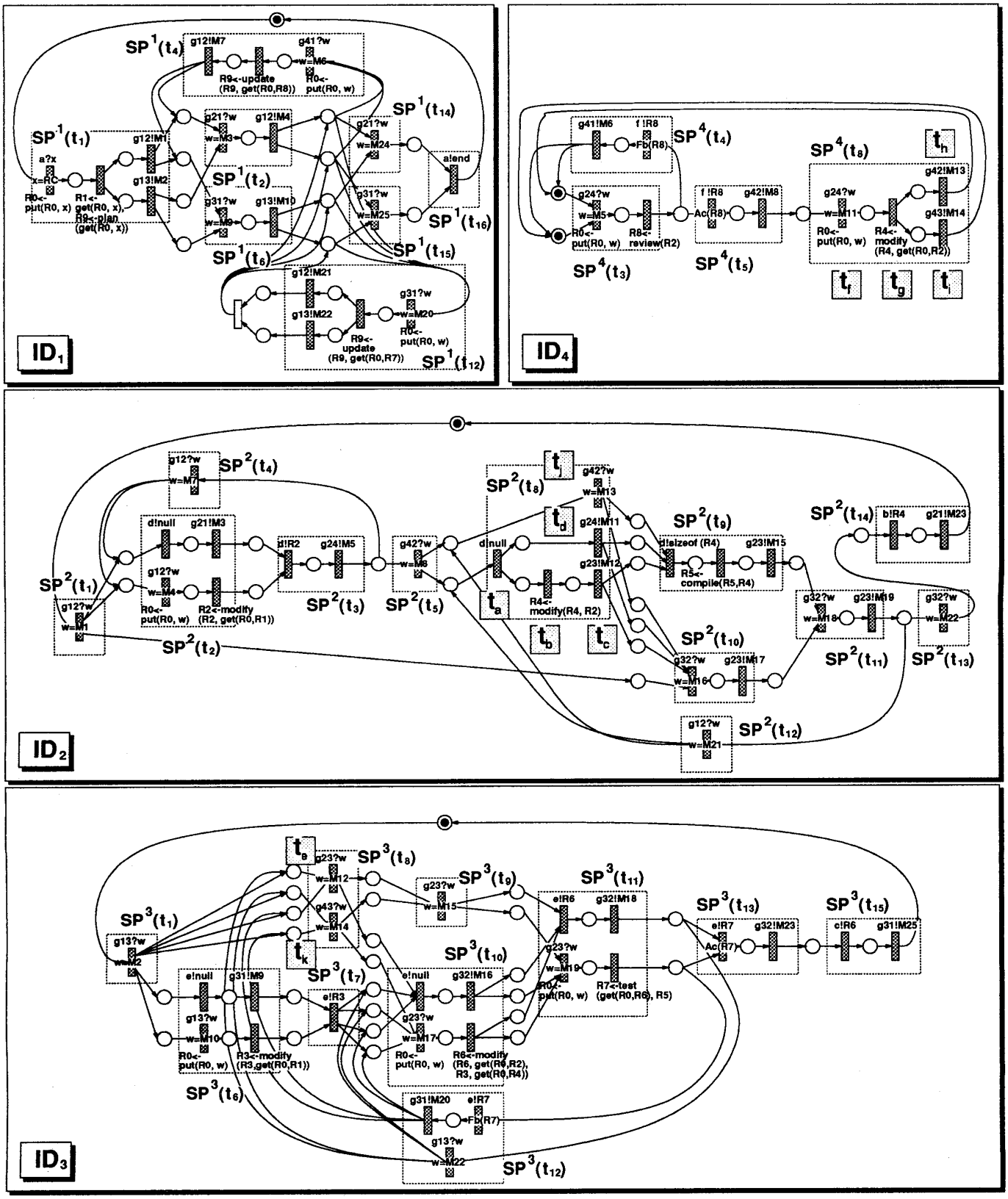

Fig. 3: Individual Descriptions, ID_1, ID_2, ID_3 and ID_4.

omitted (see [22]).

4.3 Derivation of Individual Descriptions

For the whole description in the previous section and the above resource allocation Θ, we have derived each engineer's individual description ID_k using our derivation technique.

Fig. 3 shows the derived individual descriptions ID_1, ID_2, ID_3 and ID_4. Here, for example, we focus on the transition t_2 in the whole description which represents the modification of the source codes, and show how the behaivour of the transition is practically simulated in the distributed environment. We simulate the transition t_2 by the transitions $t_a, \ldots, t_k$ in Fig. 3. (Note that some of them can fire in parallel.) The transition t_a simulates the event of t_2 and the transitions t_b and t_g simulate the register definition statement of t_2. The rest of them send/receive the messages including registers' values or synchronous messages. Each message used for simulating a transition t includes its identifier in order to be distinguished from other messages which used for simulating other transitions.

First, the engineer 2, the responsible engineer of t_2 ($\mathrm{RE}(t_2)$), executes the event of t_2 (the transition t_a). Then, it modifies the source codes (the register R_4) and sends the message M12 (type 5), to notice that the change of the R_4's value has been finished, to the engineer 3 who is one of the next responsible engineers ($\in \mathrm{RE}(t_2 \bullet\bullet)$) ($t_b$ and t_c). In parallel with above change and transmission, it sends the message M11 (type 1), including the value of the register R_2, to the engineer 4 who has R_4. The engineer 3 receives M12 (t_e). The engineer 4 receives M11 and modifies the value of R_4 (t_f and t_g). Then it sends the messages M13 and M14 in parallel, to notice that the change of R_4's value has been finished, to the engineers 2 and 3, respectively. Finally, the engineers 2 and 3 who are the next responsible engineers ($\in \mathrm{RE}(t_2 \bullet\bullet)$) receive the messages M13 and M14, respectively (t_j and t_k). $SP^2(t_9)$ and $SP^3(t_{10})$ cannot start executing until the firings of all transitions $t_a, \ldots, t_k$ are completely finished.

In Fig. 3, if the events are the internal event i, or if the guards are "true", or if the register definition statements are empty, then they are also omitted.

The properties of liveness and safeness in the whole description are preserved in each ID_k. So it can be said that each ID_k is simple enough to implement.

4.4 Discussion

There are several approaches to model software processes in Petri net based models such as MELMAC [15] and SLANG [16], and so on. And there are also some approaches in High-level Petri Nets (HPN) (for survey, see [20]). In such models, (1) the descriptions are readable using graphical representation of Petri nets, and (2) parallel actions can be naturally described, although interrupts can not be described naturally. As for (2), the modeling abilities of such models are so surpass as to describe the whole descriptions (including parallel actions) naturally, however, these approaches do not consider the distributed execution of the software processes. We focus on the distributed execution, and in this case, it is neccesary to consider the balance between the modeling ability and the simplicity of the derivation. In our derivation algorithm, we restrict the class of the underlying nets to live and safe FC nets, however, the standard example of the software process (Kellner's example problem) can be described naturally in this restricted class, and the individual descriptions are automatically derived from a whole description and a resource allocation. In addition, they can be carried out at the distributed control.

The dynamic modification of the processes is important in the process development environments. For example, MELMAC has such a facility. In our algorithm, such a facility is not supported. However, from the modified process, we can derive its individual descriptions mechanically.

The number of all transitions in ID_k ($1 \leq k \leq 4$) are 77, and 52 transitions of them are "communication transitions" (transitions for sending/receiving messages). From the result that the communication transitions are about twice as many as the transitions for simulating the events and register definition statements, it is complicated for the designers to describe such communication transitions. For more large systems, the derivation becomes more difficult. Then an automatic derivation is necessary, so our derivation algorithm is useful.

A method for simulating a transition of a whole description is given and the number of exchanged messages simulating the transition is minimized. However, the optimization as a whole description is not considered. Therefore, one may be able to derive more efficient individual descriptions. Now we have been investigating the efficiency of our derivation algorithm using this example quantitatively. We believe the derived individual descriptions are almost optimal, although we need more investigation.

5 Conclusion

In this paper, we have given a whole description of the software process modeling example in our Petri Net model with Registers (PNR model), and derived individual descriptions in the same model. In our method, a resource allocation can be described freely as a distributed allocation. By modeling such a standard example of software process, we have shown the usefulness of our derivation method.

Some system development methods using HPN are proposed (see [20]), however, general methods for system developments are not established. Therefore, our future work is to make a practical system of computer aided software engineering and evaluate the usefulness of our method quantitatively.

References

[1] Curtis, B., Kellner, M. and Over, J. : "Process Modeling," Commun. ACM, Vol. 35, No. 9, pp. 75-90

(1992).

[2] Osterweil, L. J. : "Software processes are software too," Proc. of the 9th Int. Conf. on Software Engineering, pp. 2-13 (1987).

[3] Katayama, T.: "A Hierarchical and Functional Software Process Description and Its Enaction," Proc. of the 11th Int. Conf. on Software Engineering, pp. 343-352 (1989).

[4] Kishida, K., et al.: "SDA: A Novel Approach to Software Environment Design and Construction," Proc. of the 10th Int. Conf. on Software Engineering, pp. 69-79 (1988).

[5] Saeki, M., Kaneko, T. and Sakamoto, M. : "A Method for Software Process Modeling and Description using LOTOS," Proc. of the 1st Int. Conf. on Software Process, pp. 90-104, Redondo Beach, CA (1991).

[6] Matsumoto, Y. and Ajisaka, T. : "A Data Modeling in the Software Project Database Kyoto-DB," Advances in the Software Science and Technology, Vol. 2, pp. 103-121 (1990).

[7] Iida, H., Mimura, K., Inoue, K. and Torii, K. : "Hakoniwa: Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model," Proc. of the 2nd Int. Conf. on Software Process (1993).

[8] Nakayama, T., Higashino, T. and Taniguchi, K. : "Derivation of Software Process Description for each Developer from Whole Software Process Description written in LOTOS," Technical Report of IEICE of Japan, COMP 91-65 (SS 91-22), pp. 59-67 (1991) (in Japanese).

[9] Sutton, S., Heimbigner, D. and Osterweil, L. J. : "Language Constructs for Managing Change in Process Centered Environments," Proc. of the 4th SIGSOFT Symposium on Software Development Environments, Software Eng., Notes 15, 6, pp. 206-217 (1990).

[10] Inoue, K., Ogihara, T., Kikuno, T. and Torii, K. : "A Formal Adaptation Method for Process Descriptions," Proc. of the 11th Int. Conf. on Software Engineering, pp. 145-153 (1989).

[11] Huff, K.E. and Lessor, V.R. : "A Plan-based Intelligent Assistant that Supports the Software Development Process," Proc. of the 3rd Software Engineering Symposium on Practical Software Development Environments, Software Eng., Notes 13, 5, pp. 97-106 (1989).

[12] Barghouti, N. S. : "Supporting Cooperation in the MARVEL Process-Centered SDE," ACM SIGSOFT, Vol. 17, No. 5, pp. 21-31 (1992).

[13] Kaiser, G. E., Barghouti, N.S. and Sokolsky, M.H. : "Preliminary Experience with Process Modeling in the Marvel Software, Development Environment Kernel," Proc. of the 23rd Annual Hawaii Int. Conf. on System Sci., Vol. II, pp. 131-140 (1990).

[14] Peuschel, B. and Schafer, W. : "Concepts and Implementation of a Rule-based Process Engine," Proc. of the 14th Int. Conf. on Software Engineering, pp. 262-279 (1992).

[15] Deiters, W. and Gruhn, V. : "Managing Software Processes in the Environment MELMAC," ACM SIGSOFT, Vol. 15, No. 6 (1990).

[16] Bandinelli, S., Fuggetta, A. and Grigolli, S. : "Process Modeling in-the-large with SLANG," Proc. of the 2nd Int. Conf. on Software Process, IEEE Press, pp. 75-83 (1993).

[17] Yasumoto, K., Higashino, T. and Taniguchi, K. : "Software Process Description using LOTOS and Its Enaction," Proc. of the 16th Int. Conf. on Software Engineering (ICSE-16), pp. 169-179 (May 1994).

[18] Murata, T. : "Petri Nets: Properties, Analysis and Applications," Proc. of the IEEE, Vol. 77, No. 4, pp. 541-580 (1989).

[19] Kellner, M. et al. : "ISPW-6 Software Process Example," Proc. of the 1st Int. Conf. on the Software Process, pp. 176-186 (Oct. 1991).

[20] Aoyama, M., Hiraishi, K. and Uchihara, N. : "Software Development Methodologies Based on High-Level Petri Nets," Computer Software, Vol. 11, No. 4, pp. 3-19 (July 1994) (in Japanese).

[21] Matsuura, S. and Honiden, S. : "Abstraction for Cooperation on Software Processes," Computer Software, Vol. 10, No. 2, pp. 48-64 (1993) (in Japanese).

[22] Okano, K., Yamaguchi, H., Higashino, T. and Taniguchi, K. : "Synthesis of Protocol Entities' Specifications from Service Specification in a Petri Net Model with Registers," IPSJ SIG Notes, Vol. 94, No. 39, pp. 157-162 (1994) (in Japanese).