Implementation of Service Specifications on Distributed Computing Systems

Hirozumi Yamaguchi

January 1998

Implementation of Service Specifications on Distributed Computing Systems

by

Hirozumi Yamaguchi

January 1998

Dissertation submitted to the Faculty of the Engineering Science
of Osaka University in partial fulfillment of the requirements for
the degree of a Doctor of Philosophy in Engineering

Abstract

This thesis summarizes the work of the author as a master/doctor student of Osaka University on the implementation of service specifications on distributed computing systems.

In a distributed computing system, multiple computers, called protocol entities, communicate with each other, exchanging messages for synchronization and data transfer. On the implementation phase of such a distributed system, the behavior of all the protocol entities must be specified. The specification of each protocol entity is called a protocol entity specification and the set of all the protocol entity specifications is called a protocol specification. In general, the control flow of each protocol entity specification may become complicated, since it contains communications among other protocol entities for cooperative computing. Therefore, it is hard for designers to describe a protocol specification directly without mistake.

In the recent years, for designing reliable distributed computing systems, many approaches based on a design method, called protocol synthesis, have been studied. In the studies of protocol synthesis, from a service specification of a distributed system and a resource allocation, a protocol specification of the distributed system is automatically derived. At the level of service specifications, protocol entities and communication channels among them are hidden, and only the actions of the system and their temporal ordering are specified. The resource allocation specifies an allocation of I/O ports and internal resources to protocol entities. On the other hand, at the level of protocol specifications, for each protocol entity, its own actions, communications with other protocol entities and their temporal ordering are specified. Both service and protocol specifications provide the same behavior for the service users of distributed system.

In general, tradeoff exists between the class of service specifications and the complexity of derivation algorithms. Therefore, it is one of the goals for the researches of protocol synthesis to provide an efficient derivation algorithm for service specifications written in a powerful model. From such a point of view, this thesis provides the following two issues of protocol synthesis.

As the first issue, an algorithm to derive a protocol specification from a service specification written in an extended model of Petri nets and a resource allocation is proposed. In general, almost all distributed systems perform inputs/outputs with parameters and the internal calculation of their new values. Also they may be performed concurrently on single/multiple protocol entities. Therefore, an extended Petri net model, called a Petri net model with Registers (PNR model in short) is used so that parallel and choice can be naturally specified and I/O's with parameters and internal calculation of their new values can be formally modeled. In the proposed method, a resource allocation, which specifies an allocation of I/O gates and registers to protocol entities can be given. Therefore the method is flexible for the change of distributed environments, such as the increase/decrease of the number of computers and the alternation of database servers.

In the derivation algorithm, first, for each transition in a service specification, how each protocol entity simulates the behavior of the transition is decided based on a fixed simulation policy. The behavior of each transition is simulated independently of other transitions, therefore, the correctness of the simulation can be proved easily. Note that in the simulation policy, the way of exchanging messages may not be uniquely decided. In such a case, the way which needs the least number of messages can be decided using a technique to solve 0-1 integer linear programming problems. Then each protocol entity performs the simulation of the transitions in the same execution order as that in the service specification. As a result, the behavior of all the transition sequences in the service specification is simulated. For such a purpose, each protocol entity needs to extract the execution order of the related transitions by removing other transitions from the service specification. However, in Petri net based models, transitions may represent synchronous points, therefore, they cannot be removed easily. In the algorithm, using a property of Petri nets, they are removed without inconsistency.

Based on the proposed method, a derivation system, which derives a protocol specification from a service specification and a resource allocation has been developed. Also an execution system, which interprets a protocol entity spec-

ification on one machine, communicating with others on a network, has been developed. Using these systems, the proposed method can be applied to Computer Supported Cooperative Work (CSCW). Suppose that all the activities of every worker are regarded as I/O events using computer tools, on a specific I/O gate for the worker. Also suppose that resources for a cooperative work, such as files, are represented as registers, and the modification of resources is regarded as the calculation of register values. At an abstract level, all of such I/O gates and registers are placed on one machine. Then the process of the cooperative work is described in PNR model. This process description is treated as a service specification. On the other hand, at an implementation level, suppose that each worker uses his own machine on a network. The machine is regarded as a protocol entity. Also suppose that the execution system is running on each protocol entity and I/O gates and registers are allocated to the protocol entities. The derivation system automatically derives a set of protocol entity specifications (a protocol specification) and each of them is interpreted by each execution system. As a result, the working process of each worker is automatically managed in a distributed environment. Using the above scheme, a protocol specification of ISPW-6 example process which specifies the modification of a software code and its examination is derived.

For distributed systems where time constraints between actions are specified, communication delays among protocol entities must be considered in their protocol specifications. As the second issue, an algorithm to derive a protocol specification from a service specification written in an extended model of time Petri nets, a resource allocation and maximum/minimum communication delays between each pair of protocol entities is proposed. In the proposed method, no synchronous clocks among protocol entities are assumed.

Protocol entities simulate the behavior of service specifications as follows. First, for each transition in a service specification, each protocol entity which has executed its I/O event sends notification to the protocol entities which will execute the I/O events of next transitions. Also each protocol entity which has calculated the new value of a register sends the value as early as possible, to the protocol entities which may need it in future. In this policy, it is necessary

to check whether there exist the time constraints of transitions in the protocol specification satisfying those in the service specification or not. It is also necessary to decide the time constraints if they exist. In the proposed method, time constraint of each transition to be decided is represented as a pair of variables and the restrictions among them are represented as linear inequalities. Using a technique to solve linear programming problems, they are decided automatically if they exist.

List of Major Publications

(1) Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K. : Synthesis of Protocol Specifications from Service Specifications of Distributed Systems in a Marked Graph Model, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E77-A, No.10, pp.1623–1633, Oct. 1994.

(2) Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K. : "Synthesis of Protocol Entities' Specifications from Service Specifications in a Petri Net Model with Registers, Proceedings of the 15th IEEE International Conference on Distributed Computing Systems (ICDCS-15), pp.510–517, May 1995.

(3) Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K. : Protocol Synthesis in a Petri Net Model with Registers and Its Application, The Transactions of the Institute of Electronics, Information and Communication Engineers, Vol.J80-A, No.7, pp.1064–1072, Jul. 1997. (in Japanese)

(4) Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K. : Protocol Synthesis from Time Petri Net Based Service Specifications, Proceedings of the IEEE 1997 International Conference on Parallel and Distributed Systems (ICPADS'97), pp.236–243, Dec. 1997.

(5) Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K. : Protocol Synthesis in a Time Petri Net Model with Registers, Transactions of Information Processing Society of Japan, Vol. 39, No. 3, Mar. 1998 (to appear). (in Japanese)

Acknowledgments

This work could be achieved owing to a great deal of helps of many individuals.

First, I would like to thank my supervisor Professor Kenichi Taniguchi of Osaka University, for his continuous support, encouragement and guidance of the work.

I'm very grateful to Professor Mamoru Fujii, Professor Toru Kikuno and Professor Hideo Miyahara for their invaluable comments and helpful suggestions concerning this thesis. I'm also very grateful to Professor Nobuki Tokura, Professor Masaru Sudo, Professor Toshinobu Kashiwabara, Professor Akihiro Hashimoto, Professor Kenichi Hagiwara, Professor Masaharu Imai, Professor Katsuro Inoue and Professor Toru Fujiwara for their valuable comments on this thesis.

I grieved to hear that Professor Seishi Nishikawa passed away in August, 1997. I will remember him with great affection and pray for the repose of his soul.

I would like to express my sincere gratitude to Associate Professor Teruo Higashino of Osaka University for his adequate guidance, valuable suggestions and discussions throughout this work. This work could not be achieved without his continuous support, encouragement and guidance.

I wish to express my special gratitude to research associate Dr. Kozo Okano of Osaka University for his continuous support, helpful suggestions, encouragement and adequate guidance.

I also wish to express my special gratitude to Assistant Professor Keiichi Yasumoto of Shiga University for his helpful suggestions and encouragement.

I wish to thank Professor Tadao Murata of University of Illinois for his helpful comments for starting this work.

Finally, I would like to thank all the members of Taniguchi Laboratory of Osaka University for their helpful advice.

Contents

Chapter 1

Introduction

Due to the recent progress of high-speed networks and high-performance computers, many systems are implemented as distributed computing systems on networks, where computers communicate with each other for cooperation. On the implementation phase of such a distributed computing system, the behavior of all the computers must be specified. However, their control flows may become complicated, since they contain communications among the computers. Therefore, it is hard for designers to describe their behavior directly without mistake.

In the recent years, in order to design and implement reliable distributed computing systems, many studies of protocol synthesis [10, 11] have been studied. In protocol synthesis methods, the specification of a distributed system is described at two different abstract levels. At an abstract level, computers (protocol entities) and communication channels among them are hidden. The specification at this level is described as the I/O events on Service Access Points (SAP's), the modification of internal resources and their ordering. SAP's represent interfaces between the system and users. On the other hand, at an implementation level, each protocol entity has some of SAP's and internal resources. The specification at this level consists of a set of all the protocol entities' specifications, each of which is called a protocol entity specification. Each protocol entity specification is described as the I/O events on its SAP's, the modification of its internal resources, communications and their ordering. The specifications at the former and latter levels are called service and protocol specifications, respectively. In protocol synthesis methods, a derivation algorithm automatically derives a protocol specification from a service specification and a resource allocation, which specifies an allocation of SAP's and internal resources to protocol entities. Therefore, it can prevent the mistake and reduce the complexity for describing protocol specifications.

Several approaches of protocol synthesis based on some computational models such as FSM/EFSM [23, 24, 25, 26], LOTOS [17, 18, 19, 20, 21, 22] and Petri nets [51, 53] have been proposed. In FSM/EFSM based approaches, parallel synchronization cannot be specified in service specifications essentially. It is a serious disadvantage point. In LOTOS based approaches, complex control flows can be specified in service specifications using several LOTOS operators such as parallel, choice, disenabling and process invocation. However, these approaches except [19] only focus on the efficient implementation of the control flows, therefore they do not consider to treat the calculation of state variables, which represent internal resources. Although variables are treated in [19], it only describes a way to delivery input values to protocol entities. Also the same problem can be found in Petri net based approaches, even though they have an advantage for the graphical representation of service and protocol specifications. It is desirable that both parallelism and choice can be specified and that the calculation of state variables can be modeled explicitly in service specifications.

Also those approaches are not sufficient for real-time distributed systems since they do not consider the urgency of services (time constraints). It is more complex and troublesome to decide suitable time constraints of protocol specifications under the presence of communication delays, satisfying the time constraints of service specifications. For service specifications with time constraints, a timed automaton based approach in [47] and a timed LOTOS based approach in [48] have been proposed. The former one is the first approach for this issue and the latter one is an extension so that they can treat service specifications with more complex structures, using a timed LOTOS model. Although these approaches can handle service specifications with time constraints between non-successive actions, they assume the existence of synchronous clocks among protocol entities. In addition, the state variables are not considered although the transmission of their values among protocol entities may need additional communication delays.

In this thesis, the following three issues are described.

1. A protocol synthesis method from service specifications written in an extended Petri net model is proposed. The method provides how to simulate a service specification including parallelism and the calculation of state variables in a distributed environment, without inconsistency. In order to make the correctness of protocol specifications easy to prove, a simulation policy, which simulates the behavior of each transition independently, is adopted. The number of messages exchanged for simulating the behavior of each transition based on the simulation policy is minimized.

2. An application of the proposed protocol synthesis method is shown. From a whole process description of cooperative work in PNR model and an allocation of tasks to workers, a set of process descriptions of workers is automatically derived. A derivation system which derives a set of process descriptions of workers and an execution system which interprets a process description of a worker have been developed. Using the system, the efficiency of the synthesis method for cooperative work support is shown.

3. A protocol synthesis method from service specifications written in an extended time Petri net model is proposed. The method provides how to simulate a service specification with time constraints in a distributed environment under the presence of communication delays. Therefore the different cost measure, urgency of time, is introduced and an adequate simulation policy suitable for the cost measure should be adopted. In this method, a simulation policy, where each protocol entity holding the latest data sends it as soon as possible to the protocol entities which may need it in future is adopted, even though it may lead unnecessary messages. The total sum of possible time range intervals of actions in the protocol specification is maximized.

As the first issue of this thesis, a method for deriving a protocol specification from a service specification and a resource allocation is proposed. Both service and protocol specifications are written in an extended Petri net model, called Petri Net model with Registers (PNR model in short). In general, almost all distributed systems perform inputs/outputs with parameters and the internal calculation of their new values. Also they may be performed concurrently on single/multiple protocol entities. In this thesis, a Petri net, where concurrency can be naturally treated, is extended so that such I/O's with parameters and the internal calculation of their new values can be treated. In PNR model, an I/O event executed on an I/O gate (SAP) and the calculation of new values of registers (state variables) are specified as an action of a transition. Also a predicate on input and register values can be also specified as one of the firing conditions of a transition. In the proposed method, a resource allocation, that is an allocation of I/O gates and registers to protocol entities, can be specified. It means that the method can cope with the change of distributed environments, such as increase/decrease of computers and alternation of database servers, by deriving a protocol specification again from the same service specification and a new resource allocation.

In the derivation algorithm, first, for each transition in a service specification, the behavior of each protocol entity for simulating the behavior of the transition is decided based on a fixed simulation policy. Note that not all the protocol entities are related to the simulation of the transition. Here, the behavior of each transition is simulated independently of other transitions. Therefore, the correctness of the simulation can be proved for each transition. It makes the correctness proof of the derivation algorithm easy. In the simulation policy, the ways of exchanging messages may not be uniquely decided. In such a case, it is desirable that the best one, which needs the least number of messages, can be found. In the proposed method, a 0-1 integer variable is introduced for each message to be exchanged. Its value is 1 if and only if the corresponding message is sent. Then the conditions to be satisfied based on the simulation policy are represented as 0-1 linear inequalities on those variables. Using a technique to solve 0-1 integer linear programming problems, a best solution to provide the least number of messages can be found. Then each protocol entity performs the simulation of the related transitions in the same execution order as that in the service specification. As a result, the behavior of all the transitions in the service specification is simulated in the protocol specification. For such a purpose, each protocol entity extracts the execution order of the related transitions from the service specification. In general, such an extraction is simple if the service specification is written in an FSM based model, since every transition can be removed by merging its unique input state with its unique output state. However, in Petri net based models, such a technique cannot be easily applied to all the transitions, since some transitions may represent synchronous points, that is, they may have several input/output states. In the algorithm, the Petri net of a service specification is restricted to be live and safe. Such a Petri net can be divided into a tuple of FSM's. Then the transitions which are not related to the protocol entity can be removed easily, and each protocol entity obtains the execution order of the related transitions.

As the second issue of this thesis, a derivation system, which derives a protocol specification from a service specification and a resource allocation has been developed. Also an execution system, which interprets a protocol entity specification on one machine, communicating with others on a network has been developed. Using these systems, the proposed method can be applied to Computer Supported Cooperative Work (CSCW). Suppose that all the activities of each worker are regarded as I/O events using some tools, on a specific I/O gate for the worker. Also suppose that resources for cooperative work, such as files, are represented as registers, and the modification of resources is regarded as the calculation of the new values of the registers. At an abstract level, all of such I/O gates and registers are placed on one machine. Then the process of the cooperative work is described in PNR model. This working process description is treated as a service specification. On the other hand, at an implementation level, suppose that each worker uses his own machine on a network. The machine is regarded as a protocol entity. Also suppose that the execution system is running on each protocol entity and I/O gates and registers are allocated to protocol entities. The derivation system automatically derives a set of protocol entity specifications (a protocol specification) and each of them is interpreted by each execution system. As a result, the working process of each worker is automatically managed on a network. In this thesis, using the above scheme, a protocol specification of ISPW-6 example process [43] which specifies the modification of a software code and its examination is derived.

As the third issue of this thesis, a method for deriving a protocol specification from a service specification written in an extended model of time Petri nets [4], a resource allocation and maximum/minimum communication delays among protocol entities is proposed. The extended model is called a Time Petri Net model with Registers (TPNR model in short). The derived protocol specification satisfies the time constraints of the service specification.

In general, synchronous clocks which provide a precise global time are not always available in distributed systems, even though almost existing methods assume such a global time. Therefore, in distributed systems where a global time is not available, the way of simulating the behavior of service specifications with time constraints should be considered. In the derivation algorithm, the TPNR model is used, where each transition's time constraint is specified by the upper and lower bounds of the elapsed time from the time when all the previous transitions have been executed. Then for each transition in a service specification, the protocol entities which have executed the I/O events of its previous transitions send notification messages to the protocol entity which executes the I/O event of the transition. By receiving those messages, it can estimate the time when all of the I/O events of its previous transitions have executed in other protocol entities. Therefore, the time constraint of the I/O event can be decided. Also, messages for transmitting register values may need additional delays. Therefore, each protocol entity which has calculated the new value of a register sends the value as soon as possible to the protocol entities which may need it in future. Note that such a policy may lead unnecessary messages. Here, if the delays of notification and value transmission messages are considered, there may not exist the time constraints of transitions in protocol specifications which satisfy those in service specifications. Therefore, it is necessary to check whether such time constraints exist or not. It is also necessary to decide time constraints when needed. In the proposed method, the time constraint of each transition to be decided is represented as a pair of variables.

Then the relations among such variables are represented as linear inequalities. Using a technique to solve linear programming problems, they are decided automatically if they exist. Otherwise it is decided that the service specification cannot be implemented on the resource allocation and the minimum/maximum communication delays.

In this thesis, two methods for deriving protocol specifications from service specifications and resource allocations are presented. These methods are based on different concepts, therefore, the simulation policy and cost measure of one method are quite different from those of the another. The results of these researches can spread the applicable class of service specifications in protocol synthesis methods. They may be helpful for the design phase of reliable distributed computing systems.

Chapter 2

Definition

2.1. Petri Net and Its Properties

2.1.1 Petri Net

The formal definition of Petri nets [4] is given below.

Definition 2.1 *Petri Net*: A Petri net is a kind of weighted, directed, bipartite graph, with an initial state called the initial marking, and is denoted by a 5-tuple $PN = (P, T, F, W, M_0)$ (or simply (N, M_0) where N denotes (P, T, F, W)).

- P is a finite set of places,
- T is a finite set of transitions, $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$,
- F is a set of arcs, $F \subseteq P \times T \cup T \times P$,
- W is a weight function, $W : F \to \{1, 2, 3, \ldots\}$ and
- M_0 is the initial marking, $M_0 : P \to \{0, 1, 2, 3, \ldots\}$.

If a marking assigns a nonnegative integer k to a place p, we say that p is marked with k tokens. □

For $u, v \in P \cup T$, we denote the arc from u to v by (u, v) and the weight of the arc (u, v) by $w(u, v)$. A Petri net is said to be *ordinary* if the weights of all arcs are 1's. We also denote the preset $\{v|(v, u) \in F\}$ of u by $\bullet u$, the postset $\{v|(u, v) \in F\}$ of u by $u\bullet$ and the set of all the elements in the postsets for the elements in $u\bullet$ by $u \bullet \bullet$.

A transition t is said to be *enabled* if each input place p of t is marked with at least $w(p,t)$ tokens. A firing of an enabled transition t removes $w(p,t)$ tokens from each input place p and adds $w(t,p)$ tokens to each output place p of t.

2.1.2 Properties

Definition 2.2 *Liveness*: A Petri net $PN = (N, M_0)$ is said to be *live* if, for any pair of a marking M reachable from the initial marking M_0 and a transition t, there exists a finite length of firing sequence (enabled transition sequence) from M which let fire the transition t. □

Definition 2.3 *Safeness*: A Petri net $PN = (N, M_0)$ is said to be *safe* if, for any marking M reachable from M_0, the number of tokens for each place is at most one. □

Liveness and safeness guarantee that a net is deadlock-free and that the number of its reachable states from the initial marking is finite, respectively.

Now we define a class of subgraphs of Petri nets.

Definition 2.4 *SM-Component on PN*: A subgraph PN' of a Petri net PN is said to be an SM-component on PN if each transition in PN' has exactly one input place and one output place. □

Here, we introduce the following theorem for live and safe Petri nets.

Theorem 2.1 *Decomposition of Live and Safe PN*: For a live and safe Petri net PN, there exists a set of strongly connected SM-components which covers PN.
Proof: Live and safe Petri nets are strongly connected [4]. The procedure "Find" (it is described in 3.4.2) finds all the transitions and places which do not belong to any SM-component and appends them to one of SM-components. Also "Find" finds a loop and all the place-to-place paths belonging to the loop, and regards them as an SM-component. Therefore, each SM-component is strongly connected. □

2.2. Time Petri Net

Merlin's time Petri net (TPN) [1] is defined as follows.

Definition 2.5 *Time Petri Net*: A time Petri net is an extended model of Petri nets and it is denoted by a 6-tuple (P, T, F, W, C, M_0) where

- $PN = (P, T, F, W, M_0)$ is a Petri net and
- $C : T \rightarrow 0 \cup Z^+ \times 0 \cup Z^+$ is a time constraint labelling function where Z^+ denotes nonnegative rational numbers. □

Each transition t in TPN is said to be enabled if, it is enabled in PN. Now suppose that t becomes enabled at time T and has a time constraint [Eft(t), Lft(t)] (Eft(t) $\leq$ Lft(t)). t must not fire before $T +$ Eft(t) and must not stay enabled beyond $T +$ Lft(t).

In time Petri nets, only upper and lower bounds between the time when t becomes enabled and the time when t fires can be specified as constant values. Therefore, if we would like to specify the time constraints between two transitions from t_i to t_j, they must be successive transitions, that is, $t_j \in t_i \bullet \bullet$ must hold.

2.3. Extension of Petri Net and Time Petri Net

2.3.1 Petri Net Model with Registers

In this chapter, we introduce a *Petri Net model with Registers* (*PNR model* in short) to formalize I/O's with values and the calculation of the new values of the state variables using input values. Suppose that a system has a finite number of registers which represent state variables (internal resources of a system) and I/O gates which represents SAP's (interfaces between the system and its users). In PNR model, a pair of an I/O event which occurs on an I/O gate and the calculation of new register values can be specified as the action of each transition.

Definition 2.6 *Petri Net Model with Registers*: A Petri Net Model with Registers (PNR model) is denoted by a pair $PNR = (PN, \Sigma)$, where Σ is defined as a 8-tuple $\Sigma = (G_s, R, V, E, G, C, \delta, Init)$.

- PN is a Petri net,
- G_s is a finite set of I/O gate symbols,
- R is a finite set of register variables,
- V is a finite set of input variables,
- E is a finite set of I/O event expressions, whose I/O gates are the elements in G_s,
- G is a finite set of guard expressions,
- C is a finite set of register value substitution statements,
- $\delta : T \rightarrow G \times E \times C^*$ is a labelling function where $C^* = \{s \mid s \subseteq C\}$ and
- $Init : R \rightarrow D$ is a function specifying the initial values of registers where D is a domain of register values.

Each transition t has a label of 3-tuples $\langle\ G(t),\ E(t),\ C^*(t)\ \rangle$. An I/O event $E(t)$ has the either form "$a!Exp_1(...), Exp_2(...), ...?x, y, ...$" or an internal event "$i$". "$a$" is an I/O gate symbol, "$!Exp_1(...), Exp_2(...), ...$" is an output action emitting the values of "$Exp_1(...), Exp_2(...), ...$". "$?x, y, ...$" is an input action getting input values and assigning them to input variables "$x, y, ...$". The scope of those input variables is on the transition t. Register variables are global variables whose scope is on all the transitions. An internal event "i" means that no I/O event occurs on any I/O gate. A guard expression $G(t)$ is a predicate whose arguments may be input variables on t and/or register variables. Each register value substitution statement in $C^*(t)$ has a form like "$R_q \leftarrow f(..)$". f is a function whose arguments may be input variables on t and/or register variables, and returns the new value of R_q. $C^*(t)$ is allowed to be an empty set $\emptyset$. $\square$

Definition 2.7 *Firing Rule of Transitions in PNR Model:* A transition t is said to be enabled in PNR model $PNR = (PN, \Sigma)$ *iff* t is enabled in PN and the value of guard expression $G(t)$ is true. If an enabled transition t fires, the I/O

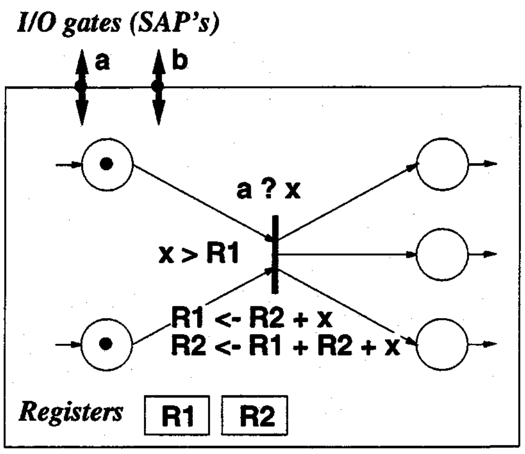


Figure 2.1: A Transition in PNR Model.

event $E(t)$ is executed. If $E(t)$ includes an input action and no input values are given from the I/O gate, t cannot fire until input values are given. Then all of the register value substitution statements in $C^*(t)$ are executed in parallel. Note that each transition t is regarded as an atomic action. $\square$

An example of a transition in PNR model is shown in Fig. 2.1. Suppose that an input value given from I/O gate "a" is 3, and the values of registers R_1 and R_2 are 1 and 2, respectively. Now the value of the guard expression "$x > R_1$" is true and each input place has a token. Therefore, the transition is enabled. If it fires, the I/O event "$a?x$" is executed (that is, the input value is actually read from I/O gate "a"). Then the register value substitution statements "$R_1 \leftarrow R_2 + x$" and "$R_2 \leftarrow R_1 + R_2 + x$" are executed in parallel. In this case, the values of registers R_1 and R_2 are substituted to 5 $(= 2 + 3)$ and 6 $(= 1 + 2 + 3)$, respectively.

2.3.2 Time Petri Net Model with Registers

A *Time Petri Net Model with Registers* (*TPNR model* in short) is an extended model of time Petri nets. The definition is given below.

Definition 2.8 *Time Petri Net Model with Registers*: A Time Petri Net Model with Registers (TPNR model) is denoted by a pair $TPNR = (TPN, \Sigma)$ where

- TPN is a time Petri net and
- Σ is a 8-tuple $\Sigma = (G_s, R, V, E, G, C, \delta, Init)$.

The definition of Σ is the same as that in Definition 2.6. □

A transition t in TPNR model $TPNR = (TPN, \Sigma)$ is said to be enabled if, it is enabled in TPN and the value of its guard expression is true. Now suppose that t becomes enabled at time T and has a time constraint [Eft(t), Lft(t)] (Eft(t) $\leq$ Lft(t)). t must not fire before $T+$ Eft(t) and must not stay enabled beyond $T+$ Lft(t).

For example, assume that the transition in Fig. 2.1 has a time constraint **[3, 5]**. The transition must fire between times $T+3$ and $T+5$ unless it becomes disabled by time $T+5$. If it fires, its behavior is the same as that of the transition explained in Section 2.3.1.

Finally, in PNR and TPNR models, there can be a pair of transitions which substitutes/refers the value of the same registers concurrently.

Definition 2.9 *Register Conflict Transition Pair:* In PNR or TPNR model, for a pair of t_i and t_j each of which is firable in parallel with the other, if (a) both t_i and t_j have the same register R's value substitution statements or (b) t_i has a register R's value substitution statement and t_j refers the value of R, the pair is called a *register conflict transition pair.* □

Chapter 3

Protocol Synthesis from Petri Net Based Service Specifications

In this chapter, we propose an algorithm to derive a protocol specification in PNR model from a service specification in the same model and a resource allocation. In Section 3.1, we will show examples of a service specification and a resource allocation. The behavior of the service specification is also explained. In Section 3.2, we will show the derived protocol specification from the service specification and a resource allocation in Section 3.1. The behavior of the protocol specification is also explained. Then we formally define the derivation problem treated in this chapter, in Section 3.3. The derivation algorithm is described in Section 3.4. In Section 3.5, we will give a sketch of correctness proof. The experimental results of derivation time are shown in Section 3.6. Finally, we conclude this chapter in Section 3.7.

3.1. Service Specification and Resource Allocation

3.1.1 Service Specification

Let *Sspec* denote a service specification of a distributed system. *Sspec* is described in PNR model. Figure 3.1 shows an example *Sspec*. At the level of service specifications, protocol entities (computers) and communication channels among them are hidden. I/O gates "*a*", "*b*" and "*c*" are used as interfaces between the system and its users. Registers R_{db}, R_{idx}, R_{log} and R_{num} represent a database, a database index, a log-file and an access number, respectively. R_{usr}

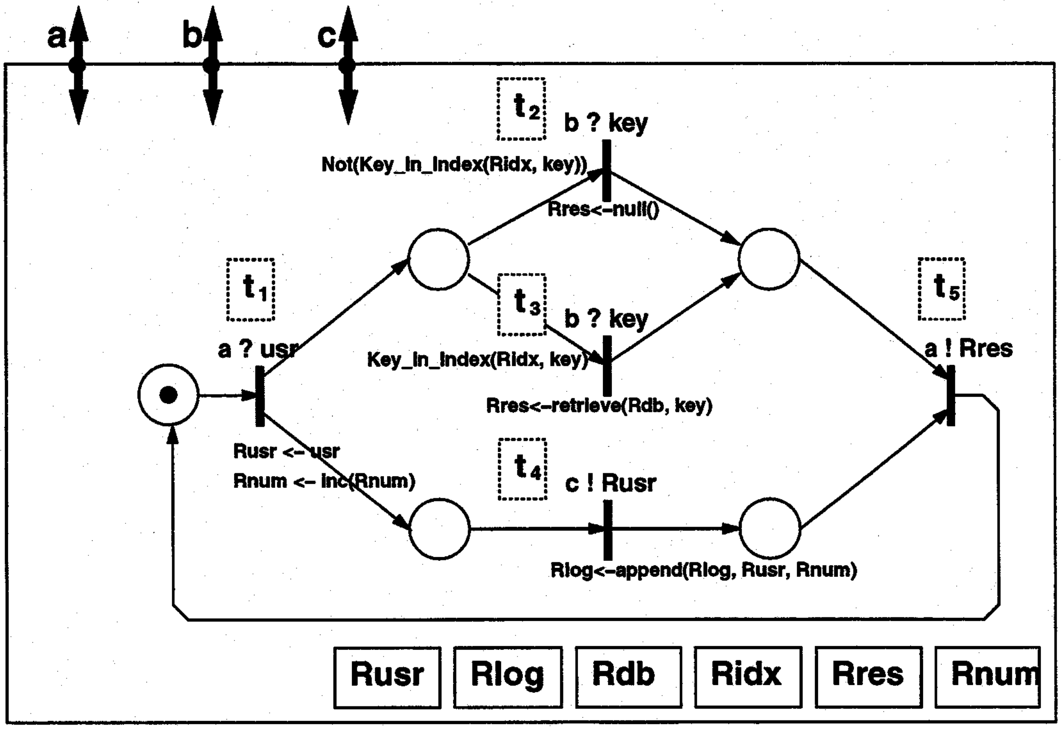


Figure 3.1: Service Specification in PNR Model.

and R_{res} are used for holding an input value and a calculation result, respectively.

On the transition t_1, a user ID given from I/O gate "a" (the value of usr), is stored to R_{usr} and an access number (the value of R_{num}) is incremented. Then, which transition t_2 or t_3 fires is decided according to a keyword given from I/O gate "b" (the value of key). If the keyword is not found in the index (the value of R_{idx}), then t_2 fires and the value of R_{res} is set to null. Otherwise t_3 fires and the value of R_{res} is set to the result of the retrieval of the database R_{db}. In parallel with t_2 or t_3, the user ID and access number are appended to the log-file (the value of R_{log}) on the transition t_4. Finally, the result of the retrieval (the value of R_{res}) is emitted to I/O gate "a". Here, functions such as "retrieve" and "append" are user-defined. Their semantics are not necessary in the algorithm. Hereafter, the guard expressions whose values are identically true and the empty sets of register value substitution statements are omitted in figures.

In this chapter, for simplifying the algorithm, we assume that *Sspec* satisfies the following restrictions.

- **[Restriction 1]** The Petri net of *Sspec* must be live and safe.
- **[Restriction 2]** *Sspec* must not include any register conflict transition pairs.
- **[Restriction 3]** *Sspec* must not include internal events "i".

The liveness and safeness are defined in Definitions 2.2 and 2.3, respectively. Restriction 1 is necessary for the derivation algorithm. It will be discussed in Section 3.4.2. The reason to assume Restriction 2 is shown in Section 3.4.3. Restriction 3 is not essential.

3.1.2 Resource Allocation

At the level of protocol specifications, a distributed system consists of p protocol entities (PE's) 1, ..., p. Here, a protocol entity k is denoted by PE_k. The I/O gates and registers used in *Sspec* are allocated to p protocol entities. Such an allocation is called a resource allocation and denoted by Alloc(p, *Sspec*). In Alloc(p, *Sspec*), each I/O gate is allocated to exactly one protocol entity, while each register can be allocated to more than one protocol entity.

Table 3.1 shows an example resource allocation Alloc(3, *Sspec*). In this resource allocation, one register R_{usr} is allocated to two protocol entities PE_1 and PE_3. Similarly, the register R_{db} is allocated to two protocol entities PE_2 and PE_3. Such a multiple allocation of one register means that there are some copies of the register and they are maintained by multiple protocol entities independently. It does not mean that there is a shared variable.

Here, we assume a restriction for Alloc(p, *Sspec*).

- **[Restriction 4]** For any pair of transitions t_i and t_j which share an input place p in *Sspec*, if $G(t_i) \cap G(t_j) \neq$ false for some values of registers/inputs, I/O gates of their I/O events must be allocated to one protocol entity in Alloc(p, *Sspec*), that is, $\mathrm{RPE}(t_i) = \mathrm{RPE}(t_j)$ must hold.

Table 3.1: Resource Allocation.

	PE_1	PE_2	PE_3
I/O gate	a, c	b	
Register	R_{usr}	R_{idx}, R_{log}, R_{res}, R_{db}	R_{usr}, R_{db}, R_{num}

RPE(t) is defined as follows.

Definition 3.1 *Responsible Protocol Entity*: For each transition t in $Sspec$, suppose that the I/O event of t is executed on an I/O gate a. The protocol entity which has the I/O gate a is called a responsible protocol entity of t and denoted by RPE(t). □

Restriction 4 prohibits a distributed choice, where more than one protocol entity has the initiative of the choice. In our derivation algorithm, we assume that the responsible protocol entity of t_i has the initiative of t_i. If Restriction 4 holds, for each choice, a protocol entity which has the initiative of all the transitions in the choice is uniquely decided. Note that there are several methods to decide such a protocol entity that has the initiative of a choice. However, we assume this for the simplicity of discussion.

3.2. Protocol Specification

A service specification $Sspec$ is implemented on p protocol entities. Here, the specification of each PE is called a protocol entity specification, and protocol entity specification of PE_k is denoted by $Pspec_k$. Also a set of p protocol entity specifications $\langle$ $Pspec_1$, ..., $Pspec_p$ $\rangle$ is called a protocol specification, and denoted by $Pspec^{\langle 1,p \rangle}$.

For each pair of PE_i and PE_j, there is a reliable duplex communication channel and the PE_i's (PE_j's) side of its ending point is modeled as an I/O gate g_{ij} (g_{ji}). If PE_i executes an output event $g_{ij}!d$, the data d (a message) is transmitted to PE_j (message sending). If PE_j executes an input event $g_{ji}?w$, the transmitted message is assigned to the input variable w, and removed from

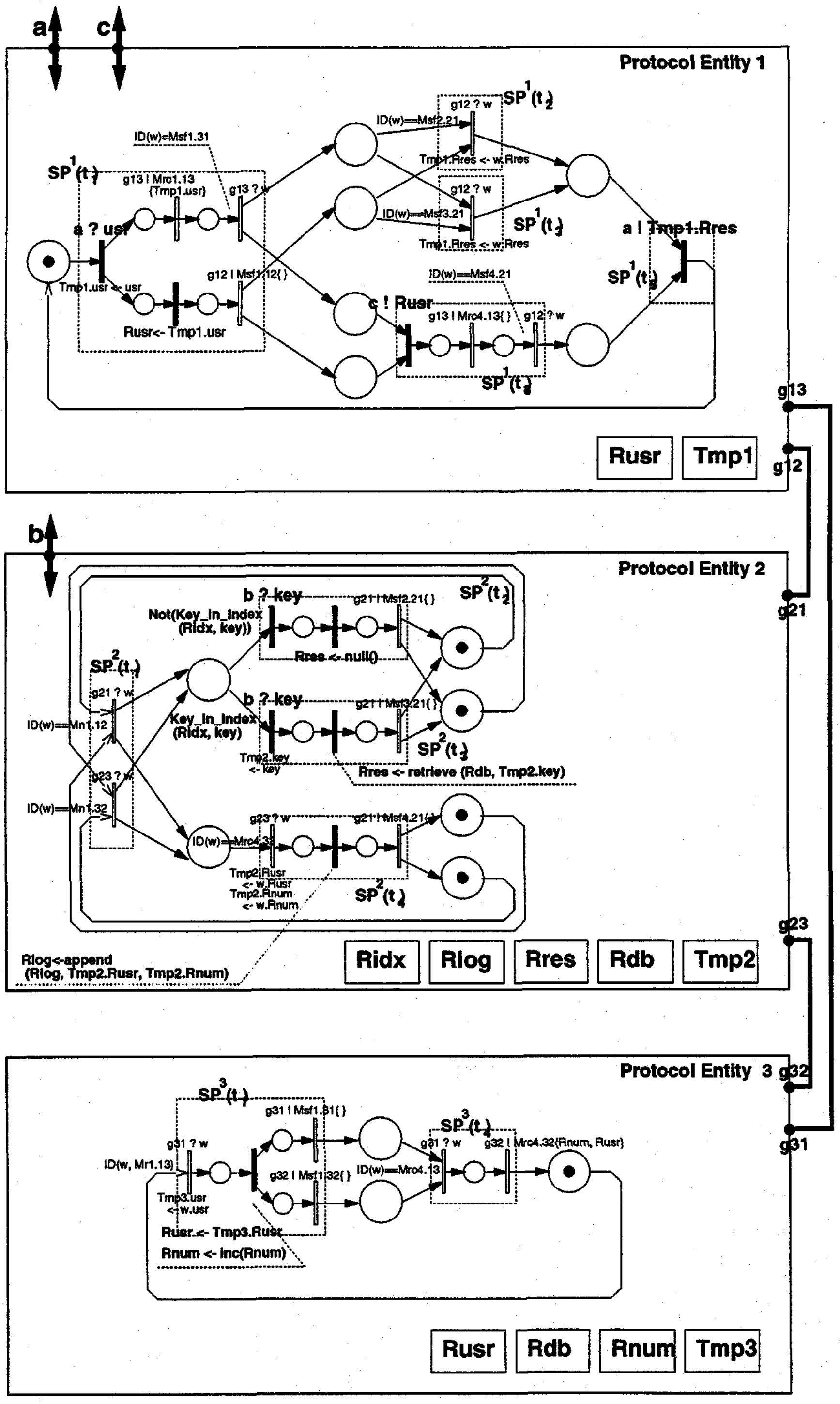


Figure 3.2: Protocol Specification in PNR Model.

the communication channel (message receiving). If there is no message in the channel, PE_j cannot execute any input action via g_{ji}. We assume that each message has its identifier (ID). PE_j gets the message ID from received data using a function ID. $ID(w)$ returns the message ID included in an input variable w. We also assume that each PE_k has a local register Tmp_k. Tmp_k can hold multiple values. It can hold the values of input variables and registers received from the other protocol entities. $Tmp_k.R_q$ ($Tmp_k.x$) denotes the value of the register R_q (the input variable x) stored in Tmp_k.

For *Sspec* in Fig. 3.1 and Alloc(3, *Sspec*) in Table 3.1, $Pspec^{\langle 1,3\rangle}$ in Fig. 3.2 is derived in our derivation algorithm. In $Pspec^{\langle 1,3\rangle}$, I/O event sequences executed on I/O gates "*a*", "*b*" and "*c*" are the same as those in *Sspec*. Note that in Fig. 3.2. for better readability, transitions for sending/receiving messages are represented as white rectangles.

3.3. Derivation Problem

We formally define a derivation problem treated in this chapter as follows. From a given pair of a service specification *Sspec* described in PNR model and a resource allocation to p protocol entities Alloc(p, *Sspec*), which satisfy Restrictions 1, 2 and 3 in Section 3.1.1 and Restriction 4 in Section 3.1.2, the problem is to derive a *correct* protocol specification $Pspec^{\langle 1,p\rangle}$.

Definition 3.2 *Correctness*: We say that $Pspec^{\langle 1,p\rangle}$ is correct with respect to *Sspec* *iff* it is *equivalent* to *Sspec*. □

The equivalence is defined below.

Definition 3.3 *Equivalence*: Suppose that all the I/O events in $Pspec^{\langle 1,p\rangle}$ via each I/O gate g_{ij} and internal events "*i*" are unobservable, and the other I/O events are observable. We say that *Sspec* and $Pspec^{\langle 1,p\rangle}$ are equivalent if, both specifications are observational congruent [16]. □

Definition 3.4 *Observational Congruence*: *Sspec* and $Pspec^{\langle 1,p\rangle}$ are said to be observational congruent if the following conditions hold. (a) All the observable

I/O sequences in $Sspec$ can be also observable in $Pspec^{\langle 1,p \rangle}$ and vice versa. (b) After the execution of every observable I/O event sequences, the set of all the executable and observable I/O event sequences in $Sspec$ and $Pspec^{\langle 1,p \rangle}$ are the same. □

For example, if a sequence of four I/O events by transitions t_1, t_3, t_4 and t_5,

$a?$"ID(352)"; $b?$"query=book"; $c!$"ID(352)"; $a!$"13 records were found"

is executable from the initial marking in $Sspec$ in Fig 3.1, it is also executable in $Pspec^{\langle 1,p \rangle}$ in Fig 3.2 and vice versa.

3.4. Derivation Algorithm

Basically, our algorithm takes the following two steps to derive $Pspec^{\langle 1,p \rangle}$.

- **[Step 1]** For each t_i in $Sspec$, a set of p PNR-subnets $SP^1(t_i)$,...,$SP^p(t_i)$ is constructed. Each $SP^k(t_i)$ is a part of $Pspec_k$ and $SP^1(t_i)$, ..., $SP^p(t_i)$ simulate the behavior of t_i. How to simulate t_i is decided based on a fixed policy (simulation policy of t_i).

- **[Step 2]** Each $Pspec_k$ is derived using the control flow of $Sspec$. For each k ($1 \leq k \leq p$), $Pspec_k$ is derived by replacing t_i in $Sspec$ with $SP^k(t_i)$. As a result, $SP^k(t_1)$, ..., $SP^k(t_n)$ are executed in the same order as that of t_1, ..., t_n.

3.4.1 Simulation of Each Transition

The components of each $SP^k(t_i)$ are some of following transitions : (a) a transition with the I/O event and guard expression of t_i, (b) transitions with some of the register value substitution statements of t_i, (c) transitions with an output event for sending messages and (d) transitions with an input event for receiving messages, a guard expression for checking message ID's and register value substitution statements for storing received values of registers and/or input variables to a local temporary register Tmp_k.

Each $SP^k(t_i)$ is constructed by composing these transitions based on the simulation policy of t_i described below.

[Simulation Policy of t_i]

- Suppose that RPE(t_i) = PE$_u$. PE$_u$ executes t_i's I/O event if (a) the value of the guard expression of t_i is true and (b) the simulation of all the previous transitions of t_i has been completed (PE$_u$ can know it by receiving SF messages explained later). Note that PE$_u$ may not know the values of some arguments (value of registers) of the I/O event and/or the guard expression. We assume that PE$_u$ has already know those values by the above SF messages (this is also explained later). At the initial marking, we also assume that each protocol entity knows the initial values of all the registers.

- Each register R_q's value substitution statement is executed by each protocol entity which has R_q (denoted by PE$_v$). PE$_u$ sends some of the following messages after the execution of the I/O event of t_i.

  - Suppose that PE$_v$ needs the values of some arguments (input variables and/or registers) of register R_q's value substitution statement. If PE$_u$ has some of those values, it sends them as a message (called a RC message) to PE$_v$.
  - If an another protocol entity PE$_{v'}$ ($v' \neq u$) has some of such values, PE$_u$ sends a message which requests PE$_{v'}$ to send a RC messages (called a RC$'$ message) to PE$_v$. If PE$_{v'}$ receives the RC$'$ message, it sends a RC message including those values to PE$_v$.
  - If PE$_v$ does not need any values of the arguments of the statement, PE$_u$ sends a RC message including no value to PE$_v$.

  As a result, each PE$_v$ receives at least one RC message.

- Each PE$_v$ receives all of RC messages sent to PE$_v$ and executes some of the register value substitution statements of t_i. If $u = v$, PE$_u$ executes them after the execution of the I/O event of t_i.

- After the execution of the statements, each PE_v sends a message (called a SF message) to each $\text{RPE}(t_j)$ (denoted by PE_z) where t_j is one of the next transitions of t_i. If PE_v holds the values of some arguments of t_j's I/O event and/or guard expression and PE_z does not know those values, PE_v includes those values in the SF message.

- If a protocol entity $\text{PE}_{z'}$ ($z' \neq z$ and $z' \neq v$) holds the values of some arguments of t_j's I/O event and guard expression and PE_z does not know those values, PE_u sends a message which requests $\text{PE}_{z'}$ to send a SF message (called a SF′ message) to PE_z.

- If PE_u does not send any of the above messages, PE_u sends a SF message to each PE_z. □

For example, for the transition t_2 in *Sspec*, $SP^2(t_2)$ has three transitions of types (a), (b) and (c) (see Fig. 3.2). Also, $SP^3(t_2)$ is not appeared, since PE_3 does not concern the simulation of t_2. Note that a message ID "$\text{Mrc}_{i.xy}$" ("$\text{Msf}_{i.xy}$") represents that it is a RC (SF) message sent from PE_x to PE_y for simulating the transition t_i.

Minimizing Number of Messages

Here, we consider to optimize $Pspec^{\langle 1,p \rangle}$ based on an appropriate cost measure. In general, both communication and processing costs can be considered as the cost measures of distributed systems. The communication costs are, for example, the number or size of messages, the number of hops, average link utilization, and so on. Also the processing costs are load distribution, average CPU utilization and so on.

In our approach, it is uniquely decided which protocol entities execute t_i's I/O event and register value substitution statements, based on the simulation policy of t_i and Alloc(p, *Sspec*). However, there are several ways of exchanging messages to simulate the behavior of t_i. For example, suppose that a register R_q is allocated to both $\text{RPE}(t_i)$ (say PE_u) and $\text{PE}_{v'}$. Also suppose that PE_v ($\neq u, v'$) needs to refer the value of R_q for executing a register value substitution statement of t_i. In this case, there are two ways to sends the value of R_q to

PE_v. If $\text{PE}_{v'}$ sends it, a RC$'$ message is necessary from PE_u to PE'_v, therefore, the number of message is one more than that in the case PE_u directly sends the value to PE_v. However, suppose another case that PE_v needs not only the value of R_q but also the value of $R_{q'}$ which is allocated only to $\text{PE}_{v'}$. If $\text{PE}_{v'}$ sends them as a RC message, the number of message is one lesser than that in the case PE_u and $\text{PE}_{v'}$ send the values of R_q and $R_{q'}$, respectively.

As mentioned above, processing costs are fixed in our method, however, communication costs are not. Therefore, we adopt the communication costs, especially the number of messages exchanged in simulating each t_i, as the cost measure and optimize it.

In this algorithm, we formalize such a problem as a 0-1 integer linear programming problem (a 0-1 ILP problem) and get an optimal solution. The following is the technique to get such a solution.

First, for each transition t_i, we introduce boolean variables. We assume that the values of these variables are 1 (true) or 0 (false).

[Variables]

- RC'_{xy} : its value is 1 *iff* a RC$'$ message is sent from PE_x to PE_y, otherwise 0.
- RC_{xy} : its value is 1 *iff* a RC message is sent from PE_x to PE_y, otherwise 0.
- $\text{RC}_{xy}\_R_h$ ($\text{RC}_{xy}\_x_k$) : its value is 1 *iff* the RC message sent from PE_x to PE_y contains the value of a register R_h (an input variable x_k), otherwise 0.
- SF'_{xy} : its value is 1 *iff* a SF$'$ message is sent from PE_x to PE_y, otherwise 0.
- SF_{xy} : its value is 1 *iff* a SF message is sent from PE_x to PE_y, otherwise 0.
- $\text{SF}_{xy}\_R_h$: its value is 1 *iff* the SF message sent from PE_x to PE_y contains the value of a register R_h, otherwise 0.

Then we represent the conditions to be satisfied if we follow the simulation policy of t_i, as 0-1 integer linear inequalities on these variables.

[0-1 Integer Linear Inequalities]

- For each tuple of protocol entities PE_x, PE_y and a register R_h, the following inequalities must hold, due to the definitions of the variables RC_{xy}, $\mathrm{RC}_{xy}\_R_h$, SF_{xy} and $\mathrm{SF}_{xy}\_R_h$.

  $$\mathrm{RC}_{xy} \geq \mathrm{RC}_{xy}\_R_h \tag{3.1}$$

  $$\mathrm{SF}_{xy} \geq \mathrm{SF}_{xy}\_R_h \tag{3.2}$$

  Also for each tuple of PE_x, PE_y and an input variable x_k, the following inequality must hold, due to the definitions of the variables RC_{xy} and $\mathrm{RC}_{xy}\_x_k$.

  $$\mathrm{RC}_{xy} \geq \mathrm{RC}_{xy}\_x_k \tag{3.3}$$

- Hereafter, we denote the responsible protocol entity of t_i ($\mathrm{RPE}(t_i)$) by PE_u. For each pair of a protocol entity PE_v ($v \neq u$) which executes at least one of the register value substitution statements of t_i and an input variable x_k, if PE_v needs the value of x_k to execute substitution statements, the following must hold.

  $$\mathrm{RC}_{uv}\_x_k = 1 \tag{3.4}$$

  The equation means that the value of the input variable is sent from PE_u to PE_v.

- For each pair of a protocol entity PE_v which executes at least one of the register value substitution statements and a register R_h, if PE_v does not hold R_h and PE_v needs the value of R_h to execute the substitution statements, the following inequality must hold.

  $$\sum_{1 \leq x \leq p, x \neq v} \mathrm{RC}_{xv}\_R_h \geq 1 \tag{3.5}$$

  It means that at least one RC message which contains the value of R_h is sent to PE_v.

- For each pair of a protocol entity $\mathrm{PE}_{v'}$ $(v' \neq u)$ and a protocol entity PE_v $(v \neq v')$ which executes at least one of the register value substitution statements of t_i, the following inequality must hold.

$$\mathrm{RC}'_{uv'} \geq \mathrm{RC}_{v'v} \tag{3.6}$$

  It means that an RC′ message is sent from PE_u $(\mathrm{RPE}(t_i))$ to $\mathrm{PE}_{v'}$ if $\mathrm{PE}_{v'}$ sends at least one RC message.

- For each protocol entity PE_y which executes at least one of the register value substitution statements of t_i, the following inequality must hold.

$$\sum_{x \neq v} \mathrm{RC}_{xv} \geq 1 \tag{3.7}$$

  It means that at least one RC message must be sent to PE_v.

- For each pair of a protocol entity PE_v which executes at least one of the register value substitution statements of t_i and the responsible protocol entity of a transition t_j $(t_j \in t_i \bullet \bullet)$ (denoted by PE_z and $z \neq v$), the following equation must hold.

$$\mathrm{SF}_{vz} = 1 \tag{3.8}$$

  It means that a SF message is sent from PE_v to PE_z.

- For each pair of a protocol entity PE_w $(w \neq u)$ which does not execute any of the register value substitution statements of t_i and the responsible protocol entity of a transition t_j $(t_j \in t_i \bullet \bullet)$ (denoted by PE_z and $z \neq w$), the following inequality must hold.

$$\mathrm{SF}'_{uw} \geq \mathrm{SF}_{wz} \tag{3.9}$$

  It means that a SF′ message is sent to PE_w which sends a SF message and does not execute any of the register value substitution statements of t_i.

- For each pair of a register R_h and the responsible protocol entity of each t_j $(t_j \in t_i \bullet \bullet)$ (denoted by PE_z), if PE_z needs the value of R_h to execute the I/O event of t_j, the following inequality must hold.

$$\sum_{x \neq z, \mathrm{PE}_x \mathrm{holds} R_h} \mathrm{SF}_{xz} \_R_h \geq 1 \tag{3.10}$$

It means that at least one SF message which contains the value of R_h is sent to PE_z.

- For the responsible protocol entity of each t_j ($t_j \in t_i \bullet \bullet$) (denoted by PE_z), the following inequality must hold.

$$\sum_{x \neq z} \mathrm{SF}_{xz} \geq 1 \tag{3.11}$$

It means that a SF message is sent to PE_z.

There may be some solutions which satisfy all of the above inequalities (3.1) - (3.11). In order to find the best solution which minimize the number of the messages for simulating the behavior of t_i, the following formula N is introduced.

$$N = \sum_{v'} \mathrm{RC}'_{uv'} + \sum_{v'} \sum_{v} \mathrm{RC}_{v'v} + \sum_{z} \mathrm{SF}'_{uz} + \sum_{w} \sum_{z} \mathrm{SF}_{wz}$$

N represents the total number of messages exchanged. A solution which minimizes N can be found using a technique to solve 0-1 ILP problems, where N is regarded as an objective function. Therefore, the way to exchange messages whose total number is minimized can be uniquely decided.

3.4.2 Simulation of Transition Sequences

Each $Pspec_k$ is derived using the control flow of $Sspec$.

It is derived by replacing t_i in $Sspec$ by the corresponding PNR-subnet $SP^k(t_i)$ if $SP^k(t_i)$ is not empty. Otherwise t_i is replaced by a ε-transition (a transition with no label). Note that if $SP^k(t_i)$ has only one input transition (a transition which has no input arcs) and only one output transition (a transition which has no output arcs), such replacement is only to reconnect the input (output) arc of t_i to the input (output) transition of $SP^k(t_i)$. Otherwise we add a ε-transition as a unique input (output) transition of $SP^k(t_i)$.

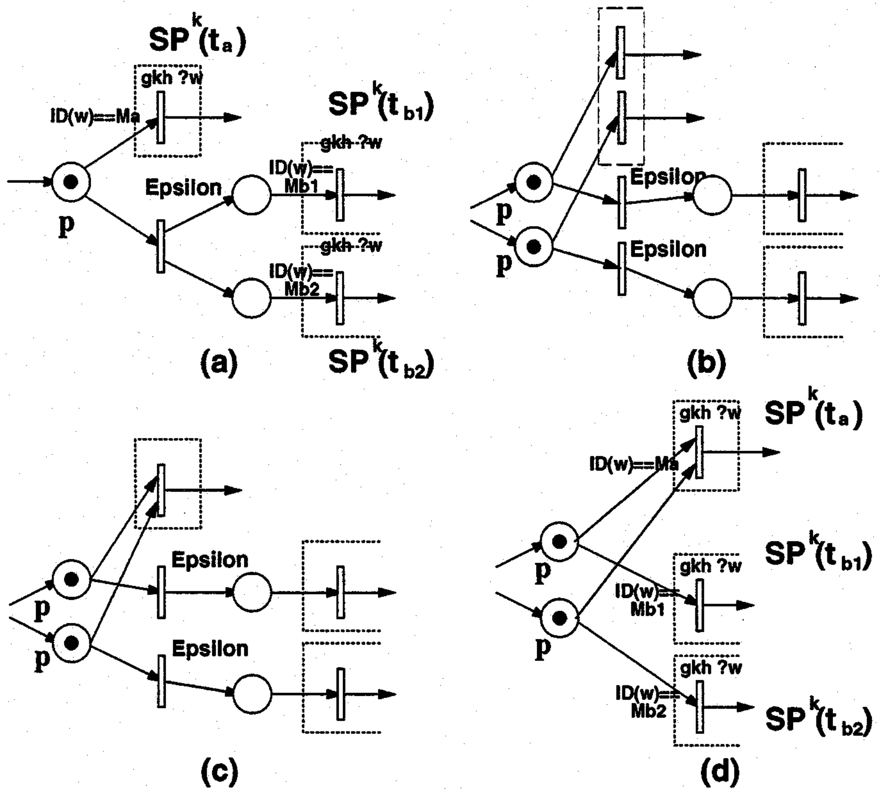


Figure 3.3: Removal of an ε-transition.

Figure 3.3(a) shows an example protocol entity specification $Pspec_k$ with a ε-transition. In this marking, $Pspec_k$ may receive one of three messages which force to start executing PNR-subnets $SP^k(t_a)$, $SP^k(t_{b_1})$ and $SP^k(t_{b_2})$. PE$_k$ must check the message ID and decide which subnets to execute. Here, ε-transitions can fire according to the firing rule of original Petri nets. However, after the free firing of the ε-transition, $SP^k(t_{a_1})$ may not be executed even though the message "M$_a$" arrives. To do so, we want to get such a structure that makes it possible to select $SP^k(t_a)$ or both of $SP^k(t_{b_1})$ and $SP^k(t_{b_2})$, as shown in Fig. 3.3(d).

We remove all ε-transitions in $Pspec_k$. The idea of removing ε-transitions is as follows. In Petri nets, ε-transitions represent synchronous points (that is, each of them has more than one input/output place). Therefore, it is difficult to use a well-known technique, which removes a ε-transition in a finite state automaton, by merging the input and output states of the ε-transition. So we decompose the Petri net of $Sspec_k$ into a set of finite state automata, and apply such technique to each decomposed ε-transitions.

In Section 3.1.1, we assume Restriction 1, where the Petri net of $Sspec$ must be live and safe. Therefore, we can say that $Pspec_k$ is also live and safe. From the live and safe Petri net PN_k of $Pspec_k$, the following procedure "Find" finds strongly connected SM-components SM_1, ..., SM_m which cover PN_k (see Theorem 2.1). Note that a SM-component is a subnet of PN_k, where each transition t_i has exactly one input place and one output place (see Definition 2.4).

The following procedure finds such a set of SM-components SM_1, ..., SM_m.

[Procedure Find]

Initially, suppose $i = 1$.

1. Suppose that there are $i-1$ SM-components SM_1, ..., SM_{i-1} found on a given live and safe Petri net PN_k. If there exists an arc e which does not belong to any of SM_1, ..., SM_{i-1}, a directed loop which contains e exists on PN_k since PN_k is live and safe. Denote the loop by SM_i and go to the next step. If there is no such e, exit this procedure with $m = i-1$.

2. For each pair of places p_x and p_y on SM$_i$, if there exists a directed path from p_x to p_y on PN_k any of whose components except p_x and p_y does not belong to SM$_i$, append the path to SM$_i$. Continue this step until such a path does not exist. Go to the next step.

3. $i = i + 1$ and go to the first step.

Secondly, using SM_1,..., SM_m obtained by the above procedure, a procedure "Remove" removes all ε-transitions from $Pspec_k$.

[Procedure Remove]

1. Decompose PN_k into a set of SM-components SM_1, ..., SM_m using "Find".

2. Merge the decomposed transitions t_i which are not ε-transitions into one transition t_i.

3. For each ε-transition t, merge its input place with its output place (it is possible since it has only one input place and one output place).

4. Remove t by deleting t and its input and output arcs. □

Finally, $Pspec_k$ without ε-transitions is obtained.

An example to remove a ε-transition is shown in Fig. 3.3. A Petri net which contains a ε-transition t (Fig. 3.3(a)) is decomposed into two SM-components (Fig. 3.3(b)). Then decomposed transitions except t, which were the same transition are merged into one transition (Fig. 3.3(c)). Then each t is removed (Fig. 3.3(d)).

3.4.3 Register Conflict

We assume that there is no register conflict transition pair in service specifications (see Restriction 2 in Section 3.1.1). If such a pair exists in service specifications, we may have to consider an additional control to prevent inconsistency in protocol specifications.

Suppose that there are two transitions t_1 and t_2 which are a register conflict pair in a service specification. Also suppose that their register value substitution statements are "$R \leftarrow 1$" and "$R \leftarrow 2$", respectively. Here, the action of each transition is regarded as an atomic action in PNR model (see Definition 2.7). Therefore, in the service specification, if the transition t_1 fires after t_2, the values of R is 1, otherwise 2. On the other hand, in a protocol specification, suppose that R is allocated to both PE_a and PE_b. In this case, both PE_a and PE_b execute both statements "$R \leftarrow 1$" and "$R \leftarrow 2$". Here, in each PE, these two statements are parallel actions, since t_1 and t_2 in the service specification can fire in parallel. Here, for example, PE_a may execute "$R \leftarrow 1$" after "$R \leftarrow 2$" (the value of R becomes 1 in PE_a) and PE_b may execute "$R \leftarrow 1$" before "$R \leftarrow 2$" (the value of R becomes 2 in PE_b). In a correct protocol specification, all the values of R must be substituted to the same value. Therefore, the values of R's must be all 1's or all 2's in this case.

In our experiences, the service specifications of many practical distributed systems do not contain register conflict transition pairs. However, some service specifications may contain a few (not so often) register conflict pairs. For such a class, we use mutual exclusion control methods. Several methods have been investigated [12, 13] for such a purpose (for survey, see [14]). Here, we give following two solutions.

Decentralized Solution

We use Lamport's algorithm [12]. To make the discussion simple, we assume that each PE has its own clock and all the clocks show the same time. Note that Ref. [12] also shows that logical clocks which only need counter registers and time stamps and so on can be substituted for these clocks. Here, for each register conflict pair t_1 and t_2 in a service specification, we regard the section from the starting point of the simulation of t_1 (t_2) to its ending point as a critical section in a protocol specification, and use Lamport's algorithm.

Suppose that there are a set of transitions t_{c_1}, ..., t_{c_k} in a service specification, each of them conflicts a register R with others. The algorithm uses three kinds of messages "req", "reply" and "release". Each message includes a pair (T_{c_i}, c_i). c_i means that the message is related to a transition t_{c_i}. T_{c_i} is the time when the message generated (a time stamp). The responsible protocol entity of t_{c_i} (denoted by PE$_{u_i}$) has a message queue Q_{u_i} which can sort entries by ordering of their time stamps. Now, we regard the section from the start of the simulation of each t_{c_i} (the head of $SP^{u_i}(t_{c_i})$) to its end (the tail of $SP^{v_i}(t_{c_i})$, where PE$_{v_i}$ is the responsible protocol entity of a transition in $t_{c_i}\bullet\bullet$) as a critical section, and denote it by CS(c_i). The algorithm is described below.

[Decentralized Algorithm]

1. If PE$_{u_i}$ tries to start the simulation of t_{c_i} (tries to enter the critical section CS(c_i)), it sends a request message req(T_{c_i}, c_i) to each PE$_{u_j}$ $(1 \leq j \leq k)$ and enqueues req(T_{c_i}, c_i) to Q_{u_i}.

2. Each PE$_{u_j}$ which has received the message req(T_{c_i}, c_i) enqueues it to the queue Q_{u_j} and sends a reply message reply(T_{c_j}, c_j) to PE$_{u_i}$.

3. For each received message reply(T_{c_j}, c_j) $(1 \leq j \leq k)$, PE$_{u_i}$ checks (a) $T_{c_i} > T_{c_j}$ and (b) the top entry of the queue Q_{u_i} has an identifier c_i. If so, PE$_{u_i}$ starts the simulation of t_i (starts the execution of the critical section CS(c_i)).

4. In our simulation policy, each of the responsible protocol entities of transitions in $t_{c_i} \bullet\bullet$ can know the end of the simulation of t_{c_i} by receiving SF messages. Therefore any of such protocol entities (denoted by PE$_{v_i}$) sends a release message release(c_i) to each PE$_{u_j}$ if it knows that the execution of critical section CS(c_i) has been finished.

5. Each PE$_{u_j}$ receives the release message release(c_i) and dequeues the entry with an identifier c_i from the queue Q_{u_i}.

An implementation of the algorithm in PNR model is shown in Fig.3.4.

Note that the end of the critical section is informed by PE_{v_i} (the responsible protocol entity of a next transition of t_{c_i}), not PE_{u_i} (the responsible protocol entity of t_{c_i}). This is different from the original algorithm. Therefore PE_{v_i} must know the identifier c_i. To do so, we assume that all messages for simulating the behavior of t_{c_i} include the identifier c_i.

The algorithm can guarantee load fairness, on the other hand, it needs additional number of messages. Let k be the number of transitions which conflict the same register, and the algorithm needs $3(k-1)$ additional messages.

Centralized Solution

If we adopt a centralized control technique, the solution is simple. If PE$_{u_i}$ tries to execute the critical section CS(c_i), it sends a request message to a specific protocol entity PE$_x$. On t_{c_i}, the request message includes the names of all the registers whose values are substituted. If PE$_{u_i}$ received a reply message from PE$_x$, PE$_{u_i}$ can start the execution of CS(c_i). If PE$_{v_i}$ finishes the execution of CS(c_i), it sends a release message to PE$_x$.

PE$_x$ has a table which keeps the states of all the registers. Each register has two states "locked" and "unlocked". Also PE$_x$ has a message FIFO queue. If PE$_x$ receives a request message, it enqueues the message. For the top entry

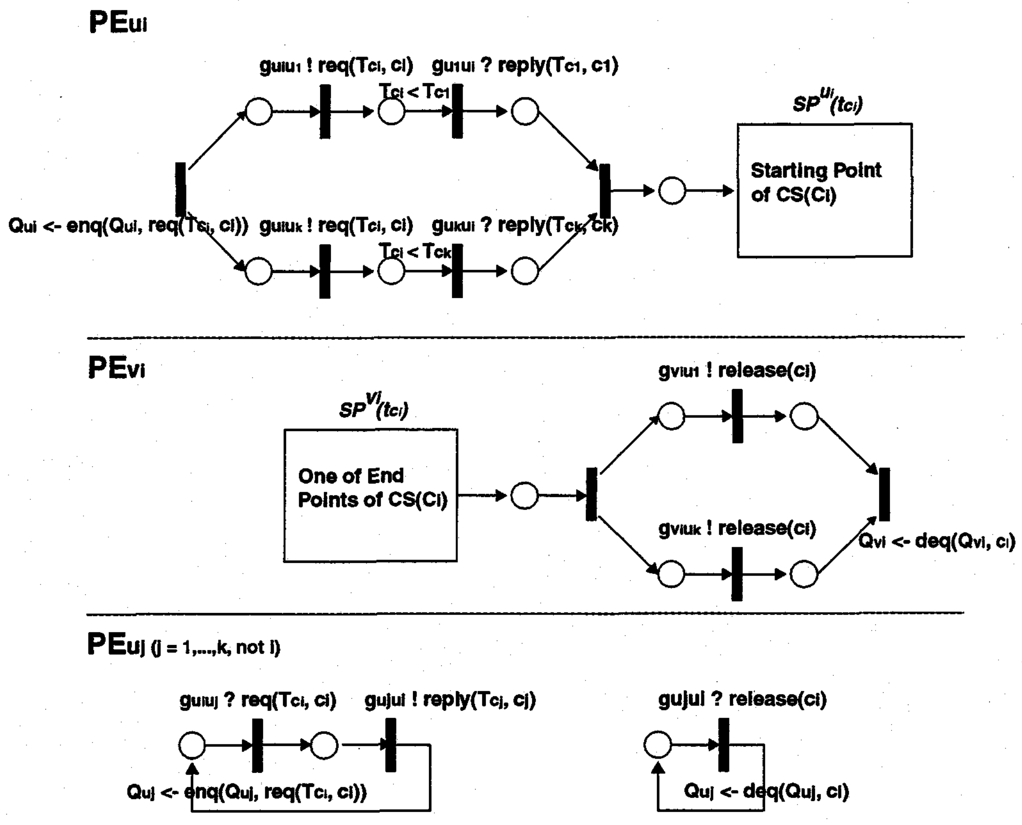


Figure 3.4: Algorithm by Lamport.

of the queue, PE$_x$ checks whether the states of all the registers specified in the entry are "unlocked" or not. If so, PE$_x$ changes their states to "locked" and sends a reply message. If PE$_x$ receives a release message, it dequeues the entry.

This technique needs the lesser number of messages than that of the previous one, however, the load of a specific protocol entity may become high.

3.5. Sketch of Correctness Proof

Here, we give a sketch of correctness proof for our derivation algorithm.

According to the algorithm, we can divide the proof into two independent phases. The first phase guarantees the correctness of the simulation of each transition. The second phase guarantees that the transitions are simulated in the same order as the execution order of the transitions in service specifications.

In the first phase, for each t_i, we should show that the followings hold.

(A1) (a) the evaluation of the guard expression's value, (b) the execution of the I/O event and (c) the execution of register value substitution statements are done in this order.

(A2) (a), (b) and (c) are done with correct values.

In the simulation policy, RPE(t_i) first evaluates the value of t_i's guard expression and executes t_i's I/O event. Each protocol entity which executes some of t_i's register value substitution statements cannot execute them without receiving RC messages. Here, all the RC messages are sent after the execution of the I/O event, since every protocol entity which sends RC messages sends them after receiving a RC$'$ message from RPE(t_i) and RPE(t_i) sends RC$'$ messages after the execution of the I/O event. Therefore, we can show that (A1) holds.

In order to show that (A2) holds, it is sufficient to show that both (A2.1) and (A2.2) hold.

(A2.1) RPE(t_i) knows all the values of the arguments of the guard expression and I/O event.

(A2.2) Each protocol entity which executes some of register value substitution statements knows all the values of the arguments of the statements.

Now we assume that (A2.1) holds and show that (A2.2) holds.

Each protocol entity which executes some of register value substitution statements, say PE$_v$, can get the values of their arguments as follows. If PE$_v$ needs the values of input variables and/or registers which RPE(t_i) holds, RPE(t_i) sends their values as a RC message to PE$_v$. Similarly, if PE$_v$ needs the values of some registers which PE$_{v'}$ holds (PE$_{v'} \neq$ RPE(t_i)), PE$_{v'}$ sends their values as a RC message. In this case, RPE(t_i) sends a RC$'$ message to PE$_{v'}$. Therefore, PE$_v$ can know all such arguments' values. Here, we should notice that if PE$_v$ needs the value of a register R_q which PE$_{v'}$ holds and the value of R_q is substituted on t_i, PE$_{v'}$ must send the value before it executes the register R_q's value substitution statement. In the simulation policy of t_i, this is guaranteed. From the above, we can show that (A2.2) holds.

Each responsible protocol entity of each $t_j \in t_i \bullet\bullet$ (RPE(t_j)) can get the argument values of the I/O event and guard expression of t_j as follows. If RPE(t_j) needs the values of registers which PE$_v$ holds, PE$_v$ sends their values as an SF message to RPE(t_j) after the execution of the register value substitution statements of t_i. Similarly, if RPE(t_j) needs the values of the registers which PE$_w$ ($w \neq v$) holds, PE$_w$ sends their values as a SF message to RPE(t_j). In this case, RPE(t_i) sends a SF$'$ message to PE$_w$, after the execution of the I/O event of t_i. Therefore, RPE(t_j) can evaluate the value of the guard expression of t_j and execute the I/O event of t_j with correct values. Similarly, at the start of the simulation of t_i, we can say that RPE(t_i) can evaluate the guard expression of t_i and execute the I/O event of t_i with correct values. As a result, we can show that (A2.1) holds.

Finally, we can show that (A2) holds, since both (A2.1) and (A2.2) hold.

In the second phase, we should show that the followings hold.

(B1) For each transition t_i, the simulation of t_i has been finished when the simulation of each $t_j \in t_i \bullet\bullet$ is started.

(B2) The simulation of transitions t_1, ..., t_m are done in the same execution order as that of t_1, ..., t_m.

In the simulation policy, only RPE(t_i) can start the simulation of t_i. Also each RPE(t_j) do not start the simulation of t_j before it receives all the SF messages. Here, we should notice that there may be more than one next transition of t_i (it means that there are more than one responsible protocol entity which receives SF messages). Now suppose that $t_i \bullet\bullet = \{t_j, t_k\}$. RPE($t_j$) may not be able to know that the simulation of t_i has been finished. The reason is as follows. Suppose that a protocol entity PE$_v$, which should send SF messages to both RPE(t_j) and RPE(t_k), has sent it to RPE(t_j) and has not sent it to RPE(t_k) yet. When RPE(t_j) receives the SF message from PE$_v$, it may start the simulation of t_j. However, at this time, PE$_v$ may not have sent the SF message to RPE(t_k) yet. Therefore, the simulation of t_j may be started before the simulation of t_i has not been finished.

In this case, the problem is only the following issue. Suppose that PE$_v$ sends the value of a register R_q (suppose that the value is one) as the SF message. Here, if the value of R_q is substituted to two on the transition t_j in the service specification, the simulation of t_j substitutes the value of R_q. If PE$_v$ sends the SF message to RPE(t_k) after the simulation of t_j, the value of R_q sent to RPE(t_k) is not one but two. PE$_v$ may send the value two to RPE(t_k) although it must send the value one.

In our simulation policy, such a situation does not occur. When the simulation of t_j tries to substitute the value of R_q on PE$_v$, PE$_v$ has already finish the simulation of t_i, since the PNR-subnet $SP^v(t_j)$ is executed after the execution of $SP^v(t_i)$. PE$_v$ sends the SF messages with a value one to both RPE(t_j) and RPE(t_k), and then substitutes the value of R_q to two. As a result, we can show that (B1) holds.

For each $Pspec_k$, PNR-subnets $SP^k(t_1)$,...,$SP^k(t_m)$ are executed in the same order as that of t_1,...,t_m in $Sspec$. Also if there is a nondeterministic choice of t_x and t_y in the service specification, we assume that RPE(t_x) and RPE(t_y) are the same protocol entity PE$_u$ (see Restriction 4 in Section 3.1.2). Therefore it can be decided which transition to be simulated. Notice that Procedure "Remove" removes all ε-transitions in each $Pspec_k$. When "Remove" tries to remove ε-transitions, it may destroy their parallel synchronization structures. Although many reformation rules of Petri nets have been proposed, however, "Remove" is different from any of them, because it does not guarantee the equivalence in terms of Petri nets. However, the reformation is correct in $Pspec_1$, ..., $Pspec_p$. we will show that below.

Basically, from the above discussion, we have shown $Sspec$ and $Pspec^{\langle 1,p\rangle}$ are equivalent if there are no ε-transitions in each $Pspec_k$. First, we focus on the procedure to decompose a live and safe Petri net into a set of SM-components, and recompose them by merging decomposed transitions into one original transition. Obviously this procedure do not change the firability of any transition in the Petri net, since it only adds some redundant places as a result. Then, assume that there is an ε-transition t_i, which is not merged in "Remove". The difference between merging decomposed transitions into an original transition and leaving them decomposed is only whether the parallel synchronization structure is restored or not. Here, suppose $\bullet\bullet t_i = \{t_{e_1}, ..., t_{e_s}\}$. From the previous discussion, RPE(t_i) starts the simulation of t_i after the simulation of all the transitions t_{e_1}, ..., t_{e_s}. It means that the simulation of t_{e_1}, ..., t_{e_s} is synchronized on the start of the simulation of t_i on RPE(t_i). Therefore, there is no mean to have a synchronization structure of t_i on PE$_k$. As a result, in each $Pspec_k$, there is no difference between merging decomposed ε-transitions into one ε- transition and leaving them decomposed. We can easily show that "Remove" can remove ε-transitions from $Pspec_k$ correctly if any ε-transition does not have parallel synchronization structure, since it is the same technique to remove ε-moves from finite state automata. From the above discussion, we can show that (B2) holds.

3.6. Evaluation

In our algorithm, the number of messages for simulating each transition is minimized using a technique to solve 0-1 ILP problems. However, in general, it takes much time to solve such problems. In this section, in order to show that our derivation algorithm can derive protocol specifications within realistic time, we have measured the derivation time for typical service specifications. We have developed a derivation system and run the system on PC/AT compatible (CPU: Intel Pentium 200MHz, Memory: 96M byte). First, we have measured the time to derive a set of PNR-subnets from a transition in a service specification with 5 registers on 10 protocol entities. We tried 20 patterns of register allocations and could get solutions within 10 seconds for all the register allocations. As a result, we can get optimal protocol specifications within realistic time for such a number of registers and protocol entities. Secondly, we tried another derivation from a transition with 50 registers on 10 protocol entities. In this case, the derivation needed more than 10 minutes in almost all cases. So we implement a heuristic algorithm and the derivation can be done within only 2 seconds, and the increase rate of the number of messages compared with the optimal number of messages is almost within 10%. As a result, we can get protocol specifications which are near optimal within realistic time although the number of registers is rather large.

3.7. Conclusion

In this chapter, we have proposed a method to derive a protocol specification of a distributed system from a service specification and a resource allocation. One of our future work is to extend the class of service specifications so that we can treat more general distributed systems in our method.

Chapter 4

Application of Protocol Synthesis : Cooperative Work Support

In this chapter, we show the efficiency of our protocol synthesis method described in Chapter 3 by applying it to cooperative work support. In Section 4.1, we discuss why the protocol synthesis method is useful for cooperative work support. In Section 4.2, we show how we model the process of a cooperative work in PNR model using an example. Then in Section 4.3, we derive the description of each engineer's working process from the description of the process of the cooperative work in PNR model, using the derivation algorithm written in Chapter 3. Section 4.4 describes how we realize cooperative work support using prototype systems which we have developed. Finally, we conclude this chapter in Section 4.5.

4.1. Why Is Protocol Synthesis Useful?

Since the size of software becomes larger, software development needs cooperative work among some engineers. However, there are some problems in such cooperative work.

Here, we focus on the following two problems. The first one is how we clarify each engineer's work from a whole working process and the second one is how we cope with the modification of the working process and the change of the environments during work.

1. In general, process designers specify a whole working process description, and each engineer must consider what he should do now and in future, from the whole working process description. For each engineer, it is obviously desirable that his own working process description is specified. Each engineer's working process description specifies what he should do now and in future clearly. However, such a description may contain communications with the other engineers for synchronization and data exchange. On the contrary, for process designers, it is desirable that they only specify the whole working process description, since it contains no communications. Therefore, there are two types of descriptions at the different levels. Here, each engineer's working process description is called an *individual description* and a whole working process description is called a *whole description.*

2. Due to the uncertain aspects of software development, the working process may be modified frequently during the work. Also environments (the number of engineers who participate in the work, the allocation of tasks to the engineers, the allocation of resources (such as databases) to machines, and so on) may be changed frequently during the work. Such modification/change during work is called *dynamic modification.* The process designers must modify each engineer's working process description for every dynamic modification.

We have developed a derivation system and an execution system, based on our protocol synthesis method, to provide solution for the two problems mentioned above. The derivation system automatically derives a set of individual descriptions and the execution system automatically interprets an individual description. Using the derivation system, the process designers do not have to specify each engineer's individual description. They only specify a whole description in PNR model and an allocation of tasks and resources to engineers (a task/resource allocation). Then a set of individual descriptions in PNR model is automatically derived (automatic derivation). Here, we assume that each engineer uses his own machine and the execution system is running on the machine. The execution system interprets the engineer's individual description and tells the engineer what tasks he should do now (guidance of work). Also the system can show the individual description graphically to the engineer. Therefore the engineer can understand easily what tasks he should do not only now but also in future (work process visualization). This is our solution for the first problem.

The execution system on each machine can suspend the execution of the individual description keeping the consistency with other execution systems. Also it can reload a modified individual description and restart the execution. Therefore, if process designers need to modify the working process and/or the task/resource allocation, they indicate all the execution systems to suspend the execution of individual descriptions. Then they only modify the whole description and/or the task/resource allocation, and call the derivation system to derive new individual descriptions. After the modification, they send the modified individual descriptions to the execution systems. The execution systems receive and reload them, and then restart the execution (dynamic modification). This is our solution for the second problem.

In the recent years, various approaches to formally describe the processes of software development have been studied [31, 27, 28, 29, 30, 32]. They can be classified by their process models: (a) the programming models such as APPL/A [33], (b) the functional models such as HFSP [28] and PDL [34], (c) the rule-based models such as GRAPPLE [35], MARVEL [36, 37] and Merlin [38], (d) the Petri net based models such as MELMAC [39] and SLANG [40, 41], (e) the LOTOS based models [30, 44] and so on. These approaches are useful for reducing the ambiguity of the processes, understanding and evaluating the processes, developing systems for Computer-Supported Cooperative Work (CSCW) and so on. However, most approaches except [44] do not consider the automatic derivation. And in [44], resources cannot be allocated to engineers (that is, they must assume that the copy of each resource is held by every engineer). Also the process visualization is not so sufficient since the method is based on LOTOS model. In Petri net based models, individual descriptions

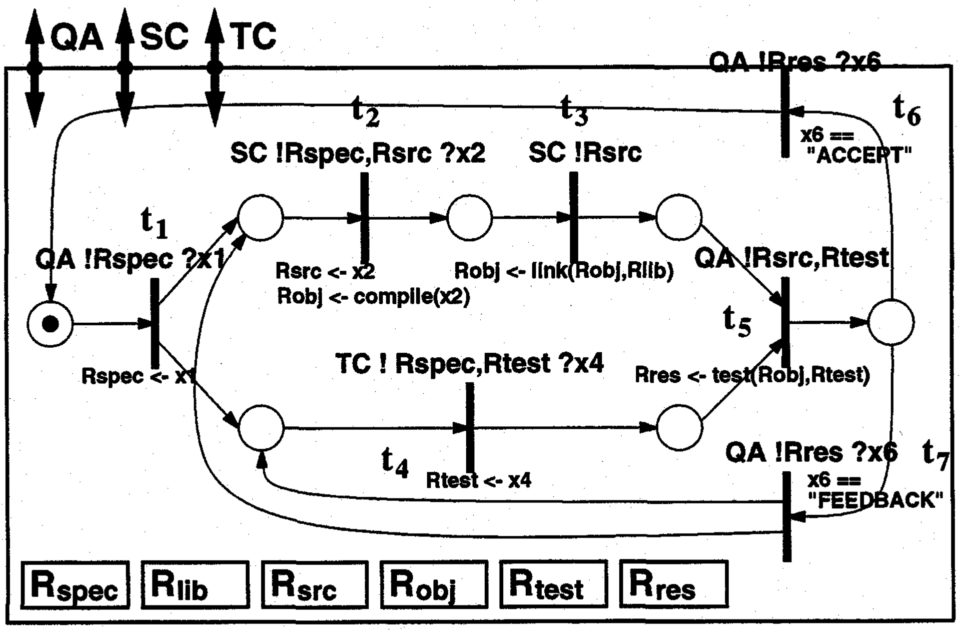


Figure 4.1: Whole Description in PNR Model.

can be represented graphically.

4.2. Process Modeling in PNR Model

4.2.1 Whole Description

Figure 4.1 shows a whole description of a cooperative software development process in PNR model.

At the level of whole descriptions, we assume that there is one computer machine. We also assume that engineers do their tasks using computer tools (such as editors) on the machine. The interfaces between engineers and the machine are represented as I/O gates. Therefore, all the engineers' tasks are represented as I/O events. Here, each I/O gate is related to a role in the working process. For example, the I/O gates QA, SC and TC in Fig. 4.1 are related to the roles of the quality assurance, the modification of the code and the test of the code, respectively. The transition t_2 in Fig. 4.1 has an I/O event "$SC!R_{spec}, R_{src}?x_2$". It means that the specification R_{spec} and the source code

R_{src} of a software are shown to the engineer which uses the I/O gate SC. The engineer modifies the source code and the result x_2 is returned via I/O gate SC. Note that at the level of whole descriptions, process designers do not have to consider how many engineers participate in the work. They only classify the tasks into some roles.

The machine has all the resources (files) necessary for the work. The inputs are modified and stored to the resources. These resources are represented as registers and the modification of resources is represented as register value substitution statements. For example, the transition t_2 has register value substitution statements "$R_{src} \leftarrow x_2$" and "$R_{obj} \leftarrow$ compile(x_2)". They mean that the input x_2 (modified source code) is directly stored to R_{src}, and also modified to an object code using the tool "compile" and the object code is stored to R_{obj}.

Now we will explain the whole description in Fig. 4.1. I/O gates QA, SC and TC represent the interfaces between engineers and the machine. Also registers R_{spec}, R_{src}, R_{obj}, R_{lib} and R_{test} represent the specification, source code, object code, library code and test package of a software, respectively. R_{res} holds a test result.

First, the transition t_1 fires. The value of R_{spec} (the specification) is emitted to the I/O gate QA and the modified specification x_1 is stored to R_{spec}. Then the transition t_2 fires and the values of R_{spec} and R_{src} (the modified specification and the source code) are emitted to the gate SC. After that the value of an input variable x_2 (the modified source code) is stored to R_{src}. Simultaneously, the value of the function "compile(x_2)" (an object code) is stored to R_{obj}. After the firing of t_2, t_3 fires and the value of R_{src} (the modified source code) is emitted to the gate SC. Then the value of the function "link(R_{obj},R_{lib})" (a final object code) is stored to R_{obj}. In parallel with the sequence of t_2 and t_3, t_4 fires and the values of R_{spec} and R_{test} (the modified specification and the test package) are emitted to the gate TC. The value of input variable x_4 (the modified test package) is stored to R_{test}. After the firing of both t_3 and t_4, t_5 fires and the values of R_{src} and R_{test} (the modified source code and test package) are emitted to the gate QA. The value of the function "test(R_{obj},R_{test})" (the test result) is stored to R_{res}. After the firing of t_5, the shared input place of t_6 and t_7 has a

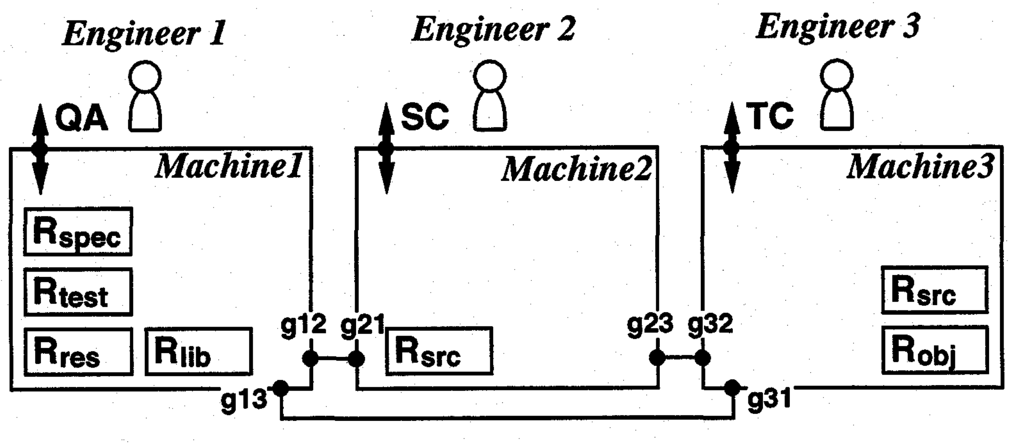


Figure 4.2: Task/Resource Allocation.

token. In this case, it is decided either t_6 or t_7 fires by an input value from the gate QA. If the input value is the string "ACCEPT", t_6 fires and the marking becomes the initial one (finish of the process). If the input value is the string "FEEDBACK", t_7 fires (repeat of the process from t_2 and t_4).

4.2.2 Task/Resource Allocation

Suppose that there are p engineers 1,...,p. At the level of whole descriptions, we assume that all the engineers use one machine. On the other hand, at the level of individual descriptions, we assume that each engineer k uses his own machine k. For each pair of two machines i and j, there is a communication channel between them and the machine i's (machine j's) side of the channel is denoted by an I/O gate g_{ij} (g_{ji}). We specify an allocation of I/O gates and registers to p machines as described in Section 3.1.2.

In our method, an allocation of an I/O gate "a" to machine k means that the engineer k plays the role related to "a". For example, the I/O gate SC is allocated to the machine 2. It means that all the tasks related to SC (code modification) are assigned to the engineer 2. Also, the allocation of a register R to machine k means that a resource R (such as a file and a database) is placed on the machine k. For example, the register R_{src} (source code) is allocated to the machines 2 and 3. It means that two copies of R_{src} are placed on the machines

2 and 3. Such an allocation of I/O gates and registers is called a task/resource allocation in this chapter.

Figure 4.2 shows an example of a task/resource allocation for the whole description in Fig. 4.1.

4.3. Derivation of Individual Descriptions

We derive each engineer's individual description from a whole description and a task/resource allocation. We use the derivation algorithm described in Section 3.4.

4.3.1 Individual Descriptions

Figure 4.3 shows the derived individual descriptions from the whole description in Fig. 4.1 and task/resource allocation in Fig. 4.2.

Now we focus on the transitions t_1 and t_2 in the whole description. At the level of individual descriptions, the engineer 1 first evaluates the value of the guard expression of t_1. It is identically true (therefore that is omitted in Fig. 4.1), then the engineer 1 executes the I/O event of t_1 "$QA!R_{spec}?x_1$", because the I/O gate QA is allocated to the machine 1. Then the register value substitution statement "$R_{spec} \leftarrow x_1$" is executed by the engineer 1 (because the register R_{spec} is allocated to the machine 1). Then the engineer 1 sends two messages "Msf$_{1.12}$" and "Msf$_{1.13}$" which inform the completion of "$R_{spec} \leftarrow x_1$" to the engineers 2 and 3, respectively. The engineer 2 receives the message "Msf$_{1.12}$" and evaluates the value of the guard expression of t_2. Now the engineer 2 knows that the simulation of t_1 has been completed and t_2 is executable. Therefore, the engineer 2 executes the I/O event of t_2, "$SC!R_{spec}, R_{src}?x_2$". However, the engineer 2 does not know the value of R_{spec}, since it is allocated to the machine 1. In this case, the engineer 1 attaches the value to the message "Msf$_{1.12}$". After the execution of the I/O event of t_2, the engineer 2 executes the register value substitution statement "$R_{src} \leftarrow x_2$". In parallel with it, the engineer 2 sends the message "Mrc$_{2.23}$" which includes the value of x_2 to the engineer 3. The engineer 3 receives the message and executes the statements "$R_{src} \leftarrow x_2$" and "$R_{obj} \leftarrow compile(x_2)$". After the execution, the engineer 3 sends the message

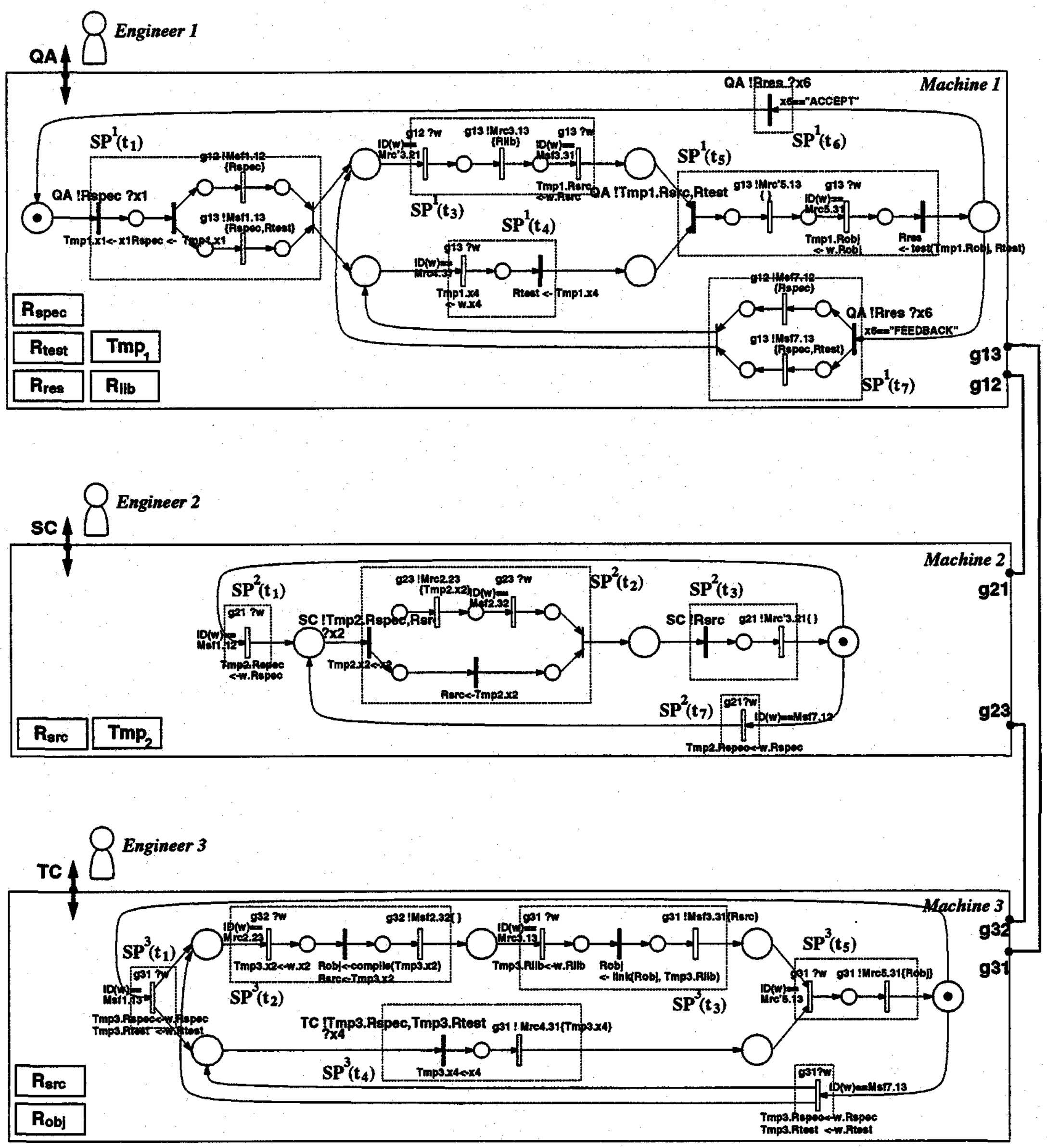


Figure 4.3: Individual Descriptions in PNR Model.

"$\mathrm{Msf}_{2.32}$" to the engineer 2. Note that a message ID "$\mathrm{Mrc}_{i.xy}$" ("$\mathrm{Msf}_{i.xy}$") means that it is a RC (SF) message for simulating t_i, sent from PE_x to PE_y.

As mentioned above, engineers cooperate with each other by exchanging messages. So each engineer's individual description contains communications with other engineers.

Table 4.1: Resource Allocation for ISPW-6 Example Process.

	Project Manager	Design Engineer	QA Engineer	Design Reviewer
I/O gate	a	b, d	c, e	f
Register	R_1, R_9	R_2, R_4, R_5	R_3, R_6, R_7	R_4, R_8

4.4. Computer Support for Cooperative Work

We have developed a derivation system which derives each engineer's individual description from a given whole description and a task/resource allocation. We have also developed an execution system which interprets each engineer's individual description on his machine. In this section, we will describe how we use these systems for cooperative work support.

4.4.1 Derivation System

The derivation system can automatically derive a set of individual descriptions.

In the derivation system, both the whole description and task/resource allocation can be given. At the level of whole descriptions, the process designers do not have to consider the number of engineers who participate and the allocation of tasks in the work. The process designers can give these information separately, as a task/resource allocation. Therefore, if the circumstances about the engineers are changed, the process designers do not have to change the process description. They only modify the task/resource allocation and derive the new individual descriptions again. It is the main advantage of our method for cooperative work support.

We described ISPW-6 example process [43] and measured the derivation time. ISPW-6 example process is used by several researchers for evaluating their modeling abilities. It describes the process of modifying the design of some software modules, reviewing the design, modifying the codes and testing them. It assumes a project manager, software engineers and a design reviewer. The software engineers are classified into a design engineer and a quality assurance (QA) engineer. The design engineer modifies the design and codes, and the QA

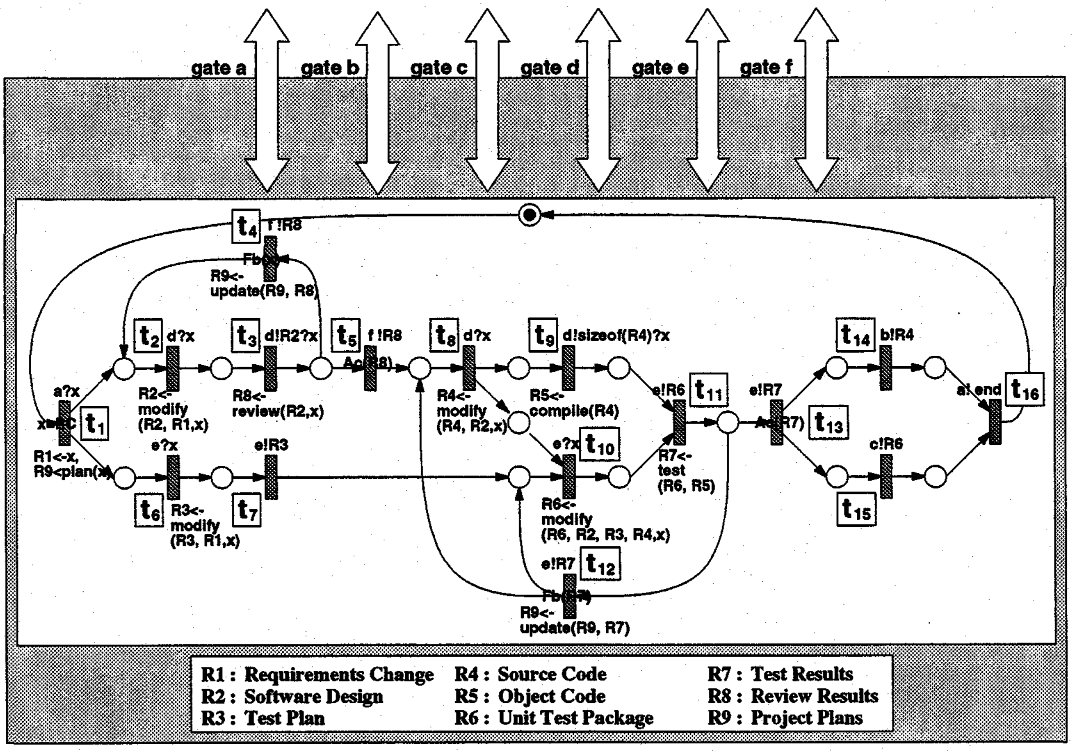


Figure 4.4: Whole Description of ISPW-6 Example Process.

engineer modifies the test plans and unit test package, and tests the codes.

Figure 4.4 shows the whole description in PNR model. It needs 16 transitions and 9 registers. From this description and a resource allocation shown in Table 4.1, it took about 3 minutes to derive 4 individual descriptions (CPU: Intel Pentium 200MHz, Memory: 96M byte) shown in Fig. 4.5. It shows that the derivation system can derive individual descriptions from a whole description of a typical example within realistic time.

The number of transitions in the individual descriptions are 77, and 52 of them are transitions for communication. As a result, the number of transitions for communication is twice as that of the other transitions. Even though the individual descriptions include many communicating transitions, the derivation system can derive them without mistake.

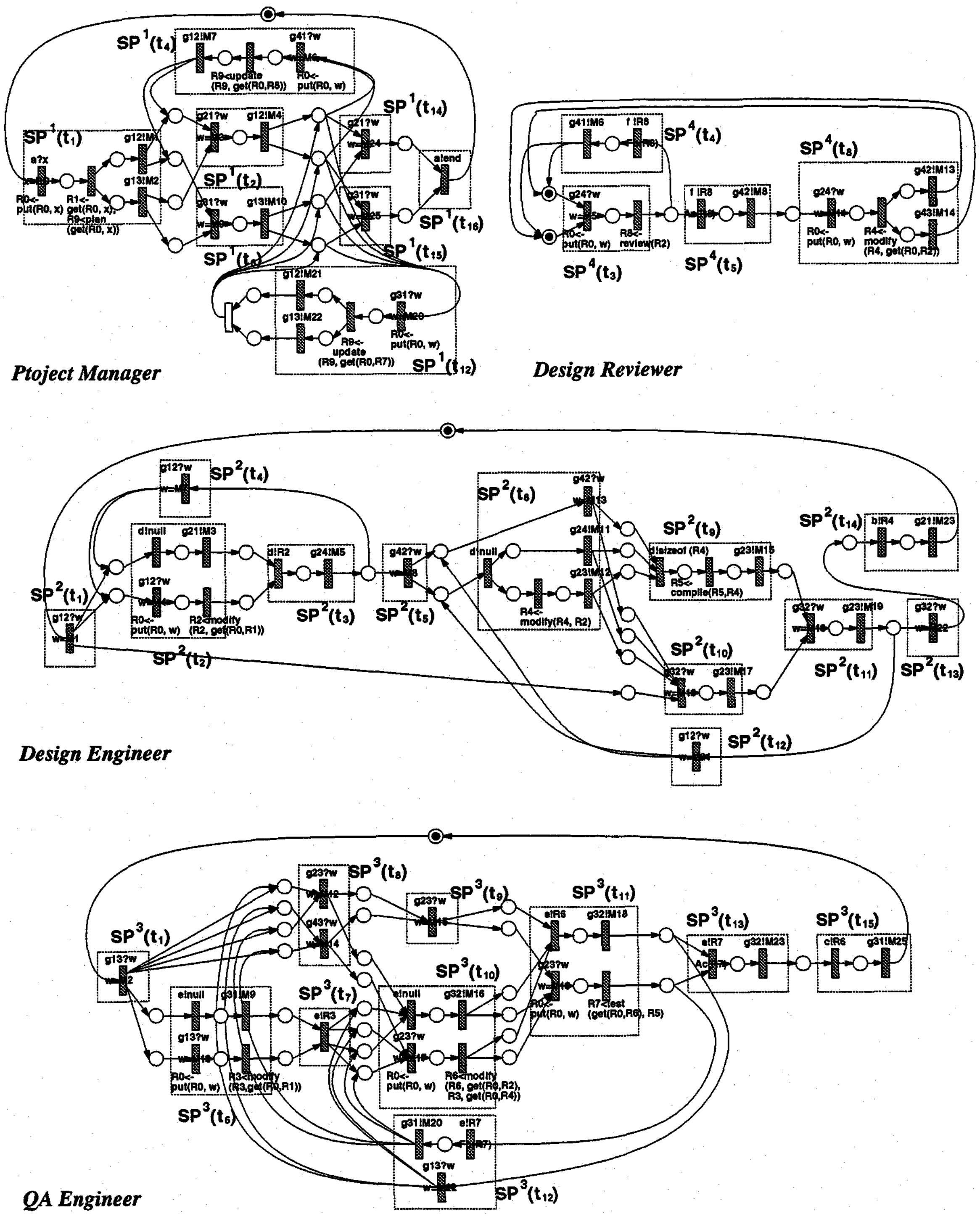


Figure 4.5: Individual Descriptions of ISPW-6 Example Process.

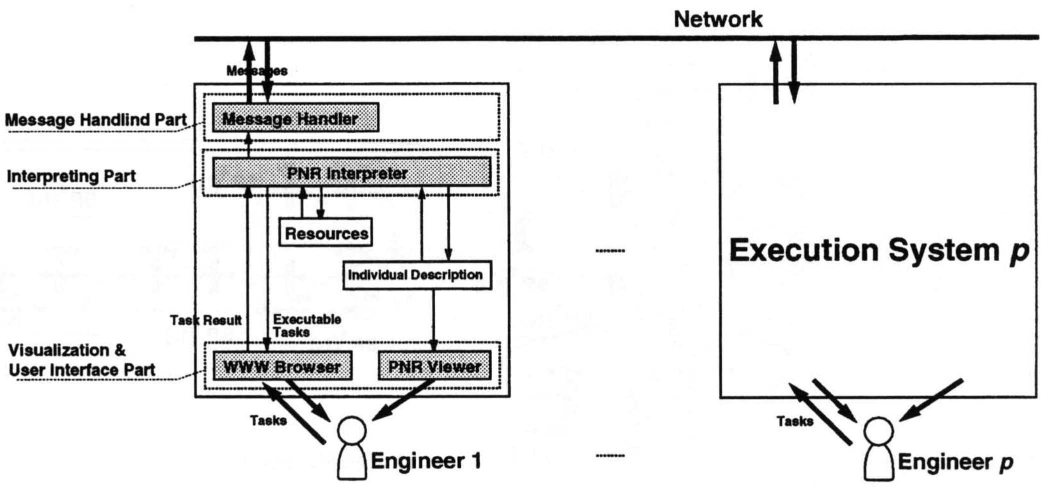
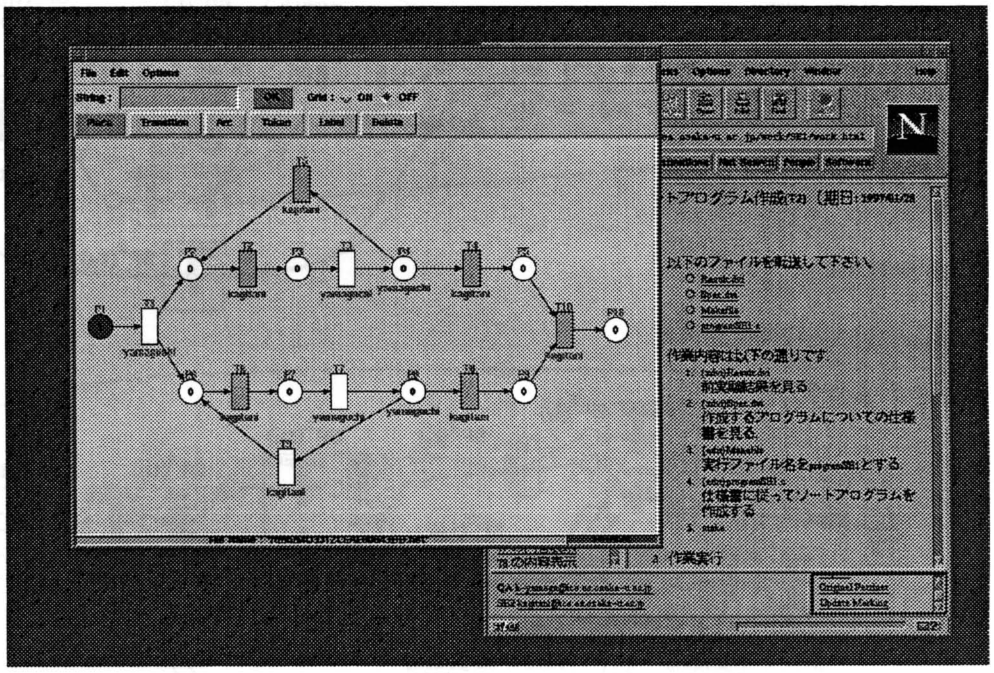


Figure 4.6: Design of Execution System.


トプログラム作成(T2) (期日:1997/01/28

以下のファイルを転送して下さい。
- Result.txt
- Spec.txt
- Makefile
- programSE1.c

作業内容は以下の通りです。
1. (edit)Result.txt
   前実験結果を見る
2. (edit)Spec.txt
   作成するプログラムについての仕様書を見る
3. (edit)Makefile
   実行ファイル名をprogramSE1とする
4. (edit)programSE1.c
   仕様書に従ってソートプログラムを作成する
5. make

3. 作業実行

Original Partner
Update Marking

Figure 4.7: Snapshot of Execution System.

4.4.2 Execution System

The execution system is shown in Fig. 4.6. It is placed on each machine. It consists of an interpreting part, a user interface part, a visualization part and a message handling part. It works as follows.

First, the interpreting part extracts enabled transitions from the individual descriptions. Then it sends the list of I/O events of the enabled transitions to the user interface part. The user interface part shows the list to the engineer and lets the engineer select one of them. The engineer selects and executes one of the I/O events. Then the interpreting part receives the working result (the input data) from the user interface part. After that, the interpreter executes the register value substitution statements of the transition. Then it changes the current marking. Similarly, if the enabled transitions need to send/receive messages, the interpreting part receives/sends messages from/to the message handling part. The visualization part can represent the engineer's individual description by using the graphical representation of Petri nets.

4.4.3 Supporting Facilities

Our goal is to provide a solution for two problems in cooperative work. The first one is how we can clarify each engineer's individual working process. Each engineer's working process may contain communications with other engineers. So process designers do not want to describe such an each engineer's working process. The second one is how we realize dynamic modification. The working process may be modified frequently during the work. Also environments (the number of engineers who participate in the work, task and resource allocation to each engineer, and so on) may be changed frequently during the work. However, it is troublesome for process designers to modify each engineer's individual description for every change.

We provide following facilities for these problems.

- Automatic Derivation: The derivation system can derive individual descriptions from a whole description and a task/resource allocation. It is helpful for process designers, since they do not have to write individual descriptions directly.

- Guidance of Work: The execution system shows each engineer his executable tasks. Also it hides communications and register value substitution from the engineer (they are executed by the execution system).

Therefore, the engineer can know which tasks he should do now and does not have to consider communications and register value substitution. Note that the user interface part is provided by World Wide Web browsers.

- Visualization of Work Process: The execution system can represent an engineer's individual description using the graphical representation of Petri nets. Therefore the engineer can know which tasks he should do not only now but also in future graphically.

  Figure 4.7 shows a snapshot of the execution system.

- Dynamic Modification: For dynamic modification, each execution system can suspend the execution of its own individual description. Here, if some execution systems suspend the execution of their individual descriptions at non-global states, some unreceived messages may be left in communication channels and the values of some registers may be incorrect (this is explained later by using an example). In our system, if an engineer k wants to modify the working process, the number of engineers and so on, he sends a suspend message to each engineer's machine. Then the execution system on the machine suspends the execution of its individual description at one of its global states. As a whole, the execution can be suspended correctly on all the machines. Then the engineer k collects the states of all the other engineers' individual descriptions and modifies the whole description and/or task/resource allocation. After that, the engineer k derives new individual descriptions using the derivation system. The engineer k sends each of individual descriptions with the new task/resource allocation to each engineer's machine. Each execution system receives the new description and task/resource allocation, and restarts the execution of the new description.

Example of Dynamic Modification

Here, we give a simple example of dynamic modification. The whole description of the example working process is shown in Fig. 4.8. It specifies the modification and test of a system code. The tasks are classified into quality assurance and

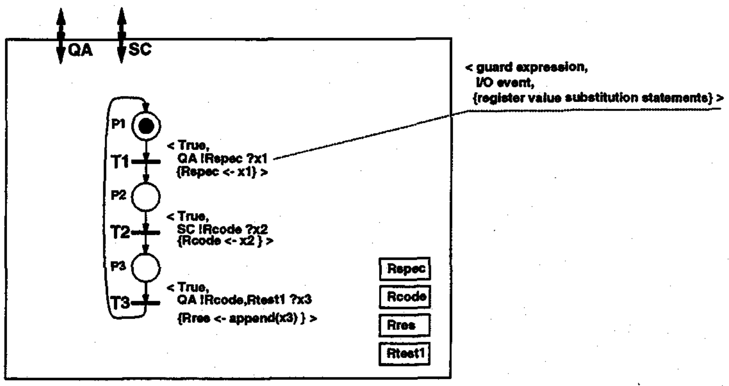


Figure 4.8: Whole Description.

Table 4.2: Task/Resource Allocation.

Machine (User)	Machine 1 (Engineer 1)	Machine 2 (Engineer 2)
Task Allocation	QA	SC
Resource Allocation	$R_{spec}, R_{code}, R_{test1}, R_{res}$	R_{code}

system coding. Suppose that they are represented by I/O gates QA and SC, respectively. Also Table 4.2 shows a task/resource allocation. There are two engineers 1 and 2. Figure 4.9 shows the derived individual descriptions for the engineers 1 and 2 (their markings show that now they are in execution). Now, let us consider the following case.

- **[Case 1]** The engineer 1 was required to test the code twice, using another test package (we assume that the package is stored on the machine 1).

The engineer 1 suspends the execution of both execution systems 1 and 2. The machine 1 sends the suspend message to machine 2. However, suspension at non-global states causes inconsistency of the system status. For example, if the execution systems 1 and 2 are suspended at the markings in Fig. 4.9, the value of the register R_{code} allocated to the machine 1 is not the same value as the one allocated to the machine 2, since the transition t in the engineer 2's individual description in Fig. 4.9 is not firing yet. To avoid this problem, all

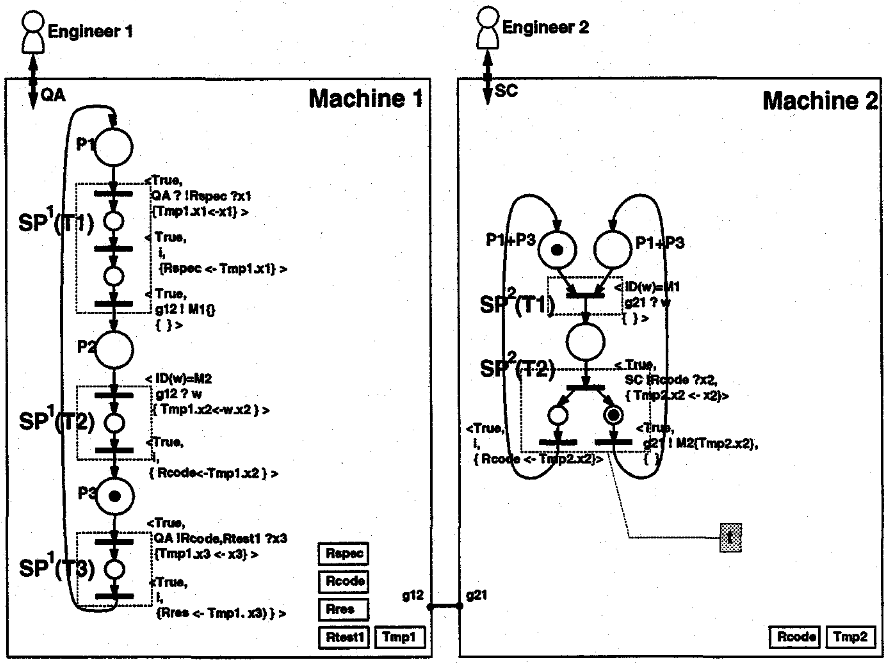


Figure 4.9: Individual Descriptions.

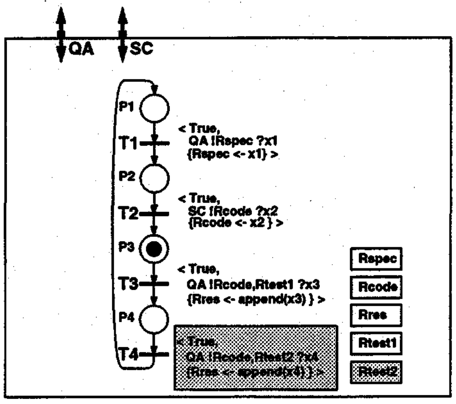

QA
SC
P1
T1
P2
T2
P3
T3
P4
T4
< True, QA !Rspec ?x1 {Rspec <- x1} >
< True, SC !Rcode ?x2 {Rcode <- x2} >
< True, QA !Rcode,Rtest1 ?x3 {Rres <- append(x3) } >
< True, QA !Rcode,Rtest2 ?x4 {Rres <- append(x4) } >
Rspec
Rcode
Rres
Rtest1
Rtest2

Figure 4.10: Modified Whole Description for Case 1.

systems do not start the new simulation of transitions after they received the suspend messages. Also the systems continue the simulation of the transitions which have been already started. In this case, the PNR-subnet $SP^1(t_3)$ is not executed while $SP^2(t_2)$ is continued to execute. After the execution finished,

Table 4.3: Modified Task/Resource Allocation for Case 2.

Machine (User)	Machine 1 (Engineer 1)	Machine 2 (Engineer 2)
Task Allocation	QA	SC
Resource Allocation	$R_{spec}, R_{code}, R_{test1}, R_{res}$	

Table 4.4: Modified Task/Resource Allocation for Case 3.

Machine (User)	Machine 1 (Engineer 1)	Machine 2 (Engineer 2)
Task Allocation	QA,SC	
Resource Allocation	$R_{spec}, R_{code}, R_{test1}, R_{res}$	R_{code}

all the systems are suspended (of course, $SP^2(t_1)$ is not executed). Then the execution system 2 sends the marking to the system 1. Engineer 1 modifies the whole description and task/resource allocation. The modified whole description is shown in Fig. 4.10. Then each engineer's individual description is derived using the derivation system, and sent to each system. In this way, the systems can restart the process.

In addition, let us consider another case.

- **[Case 2]** Assume that the system code on the machine 2 is no longer available due to crash, maintenance and so on. We want to continue the work process using the system code on the machine 1.

In this case, the whole description need not be modified. Engineer 2 suspends all the systems, and modifies just the task/resource allocation. The modified task/resource allocation is shown in Table 4.3.

Let us consider another case similar to case 2.

- **[Case 3]** The engineer SC cannot continue his work. The engineer QA must do the tasks of SC.

Also in this case, the whole description need not be modified. Engineer 1 suspends all the systems, and modifies the task/resource allocation shown in

Table 4.4.

The PNR model is suitable for dynamic modification. Since a system status can be represented by the pair of a marking and values of registers, a process modifier can easily specify the restart point by a marking, like the above case 1. Also, since the whole description and the resource allocation are separated, the process modifier need not modify the whole description if the resource allocation is only changed, like the above case 2.

The facility of dynamic modification of a process allows flexible change of the process and environment. Due to the facility, more practical supports of developments will be possible. And our modification method can ensure no inconsistency of the system status before and after the modification.

4.5. Conclusion

In this chapter, we have applied the protocol synthesis method described in Chapter 3 to cooperative work support, and shown the efficiency of the method. We have developed the derivation system which derives each engineer's individual description from a whole description and a task/resource allocation. We have also developed the execution system which interprets each engineer's individual description on his machine. We have shown that we can realize CSCW based on our protocol synthesis method using those systems in this chapter.

The main advantages of our approach are following.

- We can clarify each engineer's work process from the whole work process. Each engineer can easily know what he should do.
- We can modify the work process and the work environment during the work, without modifying each engineer's individual description directly. It is important since they are frequently changed during the work.

Chapter 5

Protocol Synthesis from Time Petri Net Based Service Specifications

In this chapter, we will propose an algorithm to derive a protocol specification in TPNR model from a service specification in the same model, a resource allocation and maximum/minimum communication delays. In this algorithm, the time constraints of actions are specified in service specifications. Also for each communication channel between two protocol entities, we assume that there is a communication delay, whose minimum and maximum values are bounded. We address in this chapter how we implement such service specifications as protocol specifications, where communication delays exist.

In Section 5.1, we will show an example of a service specification. The behavior of the service specification is also explained. In Section 5.2, a resource allocation and communication delays are shown. Then we will show the derived protocol specification in Section 5.3. The behavior of the protocol specification is also explained. We formally define the derivation problem treated in this chapter, in Section 5.4. The derivation algorithm is described in Section 5.5. In Section 5.6, we will give a sketch of correctness proof. We discuss in Section 5.7, the different points from the previous algorithm described in Chapter 3. Finally, we conclude this chapter in Section 5.8.

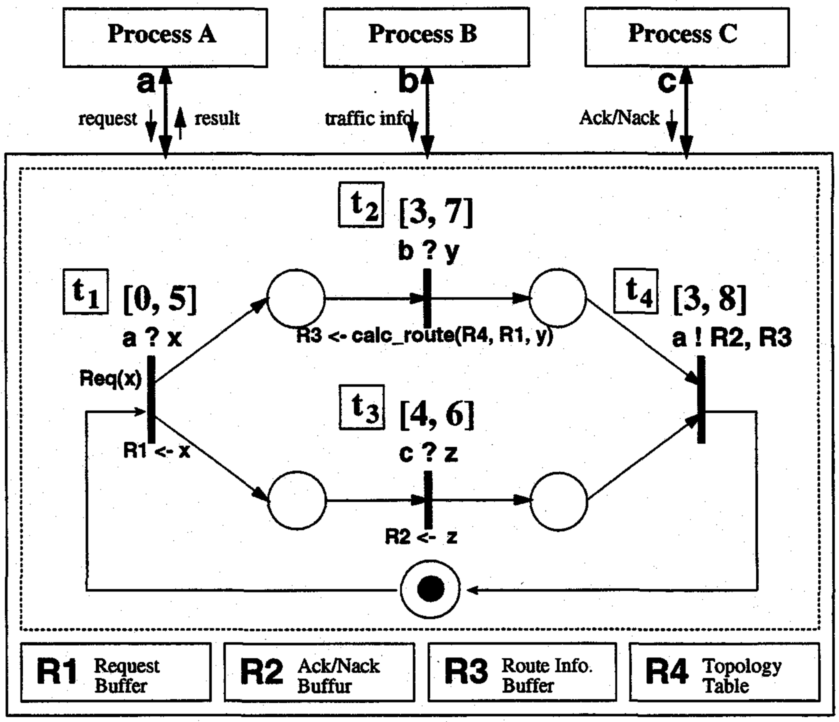


Figure 5.1: Service Specification in TPNR Model.

5.1. Service Specification

Figure 5.1(a) shows a service specification of an example protocol sub-layer in TPNR model.

The example is a simple connection management sub-layer. It communicates with its upper sub-layer, which consists of three processes A (request process), B (network observation process) and C (connection policing process), via I/O gates a, b and c. It has a topology table (register R_4). Registers R_1, R_2 and R_3 are used as temporary buffers.

The sub-layer first receives a connection request (the value of input variable x) from process A (via I/O gate a) and stores it to R_1 (request buffer). Then it gets the current network traffic information (the value of input variable y) from process B (via I/O gate b) and calculates a route using its topology table (R_4), the request (R_1) and the network traffic information (y). Then the calculated route is stored to R_3 (route information buffer). It also receives an acknowledge-

Table 5.1: Resource Allocation.

	PE_1	PE_2	PE_3
I/O gate	a	b	c
Register	R_1, R_2		R_3, R_4

ment for the request (Ack or Nack, the value of input variable z) from process C (via I/O gate c) and stores it to R_2 (Ack/Nack buffer). After that, it passes a pair of the acknowledgement and the route to process A. "**calc_route**" is a user-defined function.

We assume that service specifications satisfy Restrictions 2 and 3 described in Section 3.1.1. Also the following restriction (Restriction 1′) must be satisfied. (Restriction 1′) The Petri net of *Sspec* must be a live and safe free-choice net.

A Petri net is a free-choice (FC) net if, for any pair of transitions which share an input place p, they do not have other input places except p. In a FC net, the decision which side of transitions fires in each choice structure is done only by a token in the input place p. Restriction 1′ simplifies the control flow of *Sspec*, and therefore simplifies the derivation algorithm (see Section 5.4). Note that for a given free-choice net, there exist polynomial algorithms for safeness and liveness problems [5, 6].

5.2. Resource Allocation and Communication Delays

5.2.1 Resource Allocation

We assume that the example sub-layer consists of three protocol entities 1, 2 and 3. Each protocol entity has some of I/O gates and registers. We can specify a resource allocation Alloc(p, *Sspec*), as described in Section 3.1.2. An example resource allocation Alloc(3, *Sspec*) is shown in Table 5.1. Also in this method, an I/O gate is allocated one protocol entity, and a register can be allocated more than one protocol entity.

We also assume that Alloc(p, *Sspec*) satisfies Restriction 4 in Section 3.1.2.

Table 5.2: Minimum/Maximum Communication Delays.

	PE_1	PE_2	PE_3
PE_1	0 / 0	3 / 4	1 / 3
PE_2	1 / 4	0 / 0	0 / 1
PE_3	2 / 3	1 / 3	0 / 0

5.2.2 Communication Delays

For any pair of PE_i and PE_j $(i \neq j)$, we also assume that there is a full duplex communication channel, as described in Section 3.1.2. Here, each communication channel is sufficiently reliable, so that the maximum/minimum communication delays can be bounded by constant values. The maximum/minimum communication delays from PE_i to PE_j are denoted by $Dmax_{ij}/Dmin_{ij}$. These are shown in Table 5.2.

5.3. Protocol Specification

The protocol entities communicate with each other by exchanging messages in order to provide the same behavior in *Sspec*. For example, PE_1 executes the I/O event "$a?x$" of t_1 (since PE_1 has the I/O gate a) and sends two messages to PE_2 and PE_3 for notifying the completion of the I/O event execution. The messages "$Mn_{1.12}$" and "$Mn_{1.13}$" are sent to PE_2 and PE_3 via I/O gates g_{12} and g_{13}, respectively. PE_2 and PE_3 receive these messages and then execute the I/O events of the next transitions t_2 and t_3, that is, "$b?y$" and "$c?z$", respectively. On the other hand, PE_1 executes the substitution statement of register R_1 (since PE_1 has register R_1) using the value of input variable x and sends the message to transfer the latest value of the register. The message "$Mr_{1.13}$" is sent to PE_3 to transfer the value of R_1. PE_3 receives the message and stores the received value as the latest value of R_1 to its local temporary register, Tmp_3. This value is used to execute the substitution statement of register R_3 (See Fig. 5.2 to find $Tmp_3.R_1$ in the second argument of the substitution statement). Here, we assume that if the received register R_q's value is stored to the local temporary register, the name of the transition where the value of R_q is substituted is also stored. In Fig. 5.2, for simplicity, we assume that sending/receiving transitions can be executed immediately. So, the time constraints of all the sending/receiving transitions (represented by white rectangles) are $[\mathbf{0},\ \mathbf{0}]$, and they are omitted. Note that we can easily extended our method for treating the case that it takes some units of time to execute sending/receiving transitions (See Section 5.8).

5.4. Derivation Problem

In this section, we formally define the derivation problem treated in this chapter.

Suppose that all I/O events executed on the I/O gates used in *Sspec* are treated as observable, and that all sending/receiving events on the I/O gates for communication among protocol entities (such as "$g_{ij}?x$" and "$g_{ij}!E(\ldots)$") and internal events ("i") are treated as unobservable. If all the observable I/O event sequences (including a time interval between each pair of successive I/O events in the sequences) in *Sspec* can be observed in $Pspec^{\langle 1,p\rangle}$ and vice versa, we say that *Sspec* and $Pspec^{\langle 1,p\rangle}$ are equivalent.

It is ideal that for any given *Sspec*, $Pspec^{\langle 1,p\rangle}$ which is equivalent to *Sspec* can be derived. However, considering communication delays, for almost all cases, such a derivation is impossible. For example, suppose that there is an I/O event sequence "$a; b$" in *Sspec* and the time constraint of b is $[\mathbf{3},\ \mathbf{6}]$. In $Pspec^{\langle 1,p\rangle}$, also suppose that PE_1 and PE_2 have the I/O gates a and b, respectively and $Dmin_{12}$ and $Dmax_{12}$ are 4 and 5 units of time, respectively. PE_2 must know that the action a has been executed in PE_1, therefore PE_1 sends a message after the execution of the action a. PE_2 receives the message and executes the action b. If the message arrives at PE_2 in the shortest delay (4 units of time), the action b can be executed immediately after receiving the message. On the other hand, if the message arrives at PE_2 in the longest delay (5 units of time), the action b must be executed within 1 unit of time after receiving the message. As a result, the time constraint $[\mathbf{3},\ \mathbf{6}]$ of the action b in *Sspec* should be modified to $[\mathbf{4},\ \mathbf{6}]$ $(=[Dmin_{12} + 0,\ Dmax_{12} + 1])$ in $Pspec^{\langle 1,p\rangle}$. In this case, although

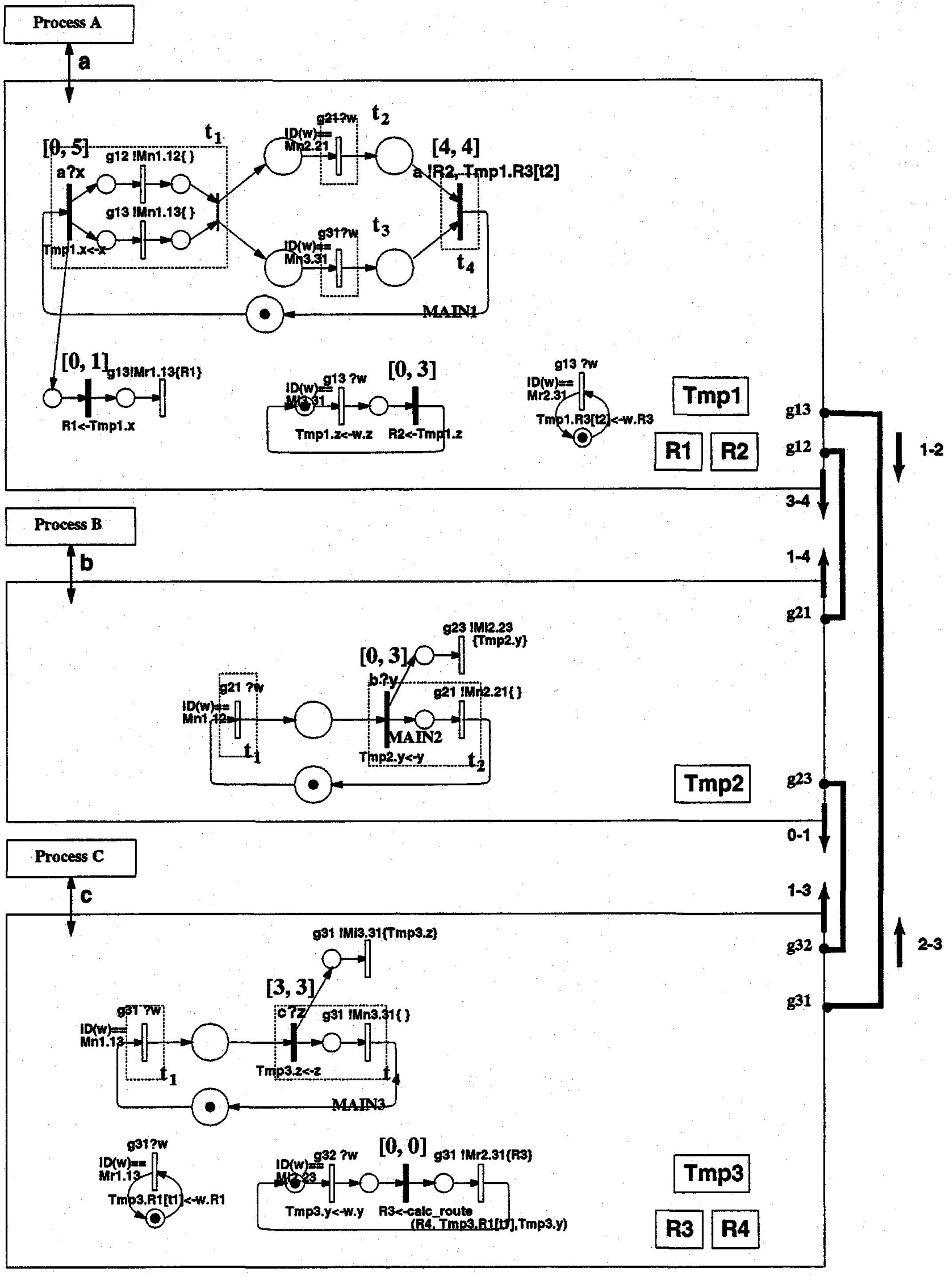


Figure 5.2: Protocol Specification in TPNR Model.

$Pspec^{\langle 1,p\rangle}$ satisfies the time constraint of *Sspec*, it is not equivalent to *Sspec*.

Here, considering the case above, we define the *correctness* of $Pspec^{\langle 1,p\rangle}$ w.r.t. *Sspec* as follows. Suppose a specification denoted by *Sspec'* which is obtained from *Sspec* by narrowing time constraints of some transitions in *Sspec*. We say that $Pspec^{\langle 1,p\rangle}$ is *correct* w.r.t. *Sspec* if, there exists *Sspec'* where (A) *Sspec'* and $Pspec^{\langle 1,p\rangle}$ are equivalent and (B) selectable transitions in *Sspec* can be also selectable in $Pspec^{\langle 1,p\rangle}$. The condition (A) means that if the time constraints in $Pspec^{\langle 1,p\rangle}$ are narrowed due to communication delays, $Pspec^{\langle 1,p\rangle}$ is correct as long as they are within those in *Sspec* like the above case. However, narrowing time constraints may change selectability in a choice structure. In the specification in Fig. 5.3(a), both actions a and b can be executed. On the other hand, in the specification in Fig. 5.3(b), which is time-narrowed specification in Fig. 5.3(a), the action b cannot be executed according to the firing rule of TPN (the action a MUST be executed within 5 units of time). That is, the action b is a "dead " (unexecutable) transition. The condition (B) is used to prevent "dead" transitions appeared in $Pspec^{\langle 1,p\rangle}$. In a choice structure as shown in Fig. 5.3(c) which is not a free-choice net, checking the selectability is difficult, since it depends on time when two independent tokens come into the places. In our method, we restrict the class of the Petri net of a service specification to a free-choice net. Free-choice nets have a simple choice structure where input places are at most 1. It facilitates to guarantee the correctness of derived $Pspec^{\langle 1,p\rangle}$.

Note that even if the condition (B) is satisfied and we ignore the occurrence time of I/O events, the observable I/O event sequences in *Sspec'* may be lesser than those in *Sspec*. The reason is as follows. In Petri net models, we can specify parallel transitions. Here, narrowing time constraints does not prevent the occurring of events, however, the possible range of event occurrence time is reduced. As a result, in such a case as there are the event sequences generated by the interleave of parallel transitions, the possible interleave patterns may be reduced. For example, suppose that there are four transitions a_1, a_2, b_1 and b_2 shown in Fig. 5.4(a). In Fig. 5.4(a), the event sequences $a_1; b_1; a_2; b_2$, $a_1; b_1; b_2; a_2$, $b_1; a_1; b_2; a_2$ and $b_1; a_1; a_2; b_2$ can be executed by the interleave of the transitions. However, in Fig. 5.4(b), which is a time restricted version

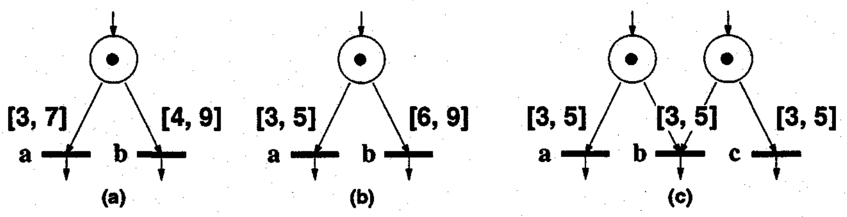


Figure 5.3: Narrowing Time Constraints in a Choice Structure.

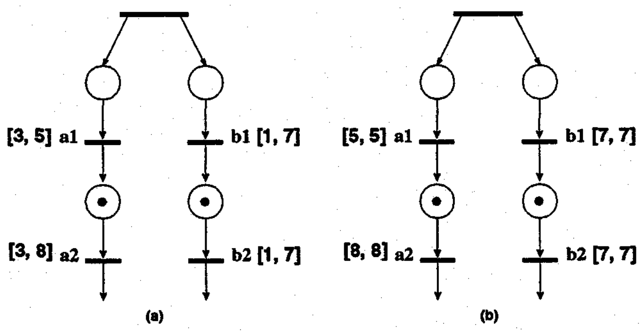


Figure 5.4: Narrowing Time Constraints in a Parallel Structure.

of Fig. 5.4(a), only one sequence $a_1; b_1; a_2; b_2$ can be executed due to the time constraints. Here, we do not consider this case, since designers are not interested in the execution order of a_1 (or a_2) and b_1 (or b_2) in a parallel structure in general.

In this chapter, we treat a problem to derive a correct $Pspec^{\langle 1,p\rangle}$ from a given tuple of $Sspec$, Alloc(p, $Sspec$) and $Dmin_{ij}/Dmax_{ij}$ for each communication channel where Restrictions 1′, 2, 3 and 4 hold.

5.5. Derivation Algorithm

Our algorithm consists of two steps: (1) deriving each protocol entity specification without time constraints from $Sspec$ according to a simulation policy, and then (2) deciding its time constraints.

5.5.1 Deriving Nets

In step (1), from given *Sspec* and Alloc(p, *Sspec*), actions and their execution order at each protocol entity are decided uniquely based on the following simulation policy (*e.g.* PE_1 executes the I/O event "$a?x$" of t_1 and then sends messages "$Mn_{1.12}$" and "$Mn_{1.13}$"). According to the policy, $Pspec^{\langle 1,p \rangle}$ without time constraints is constructed.

[Simulation Policy]

(a) For each t_j in *Sspec*, the protocol entity which has the I/O gate of the I/O event of t_j executes the I/O event. Such a protocol entity is called a *responsible protocol entity* of t_j and denoted by RPE(t_j). RPE(t_j) receives messages from all of the protocol entities, each of them has executed the I/O event of a previous transition of t_j. These messages are called *I/O completion notification messages.* Also RPE(t_j) sends a message to each protocol entity which executes the substitution statement for a register R_q of t_j. This is explained in (b).

(b) Each protocol entity executes the substitution statement for a register R_q of t_j if it has the register R_q. If it receives a message (called an *input value transfer message*), which includes the value of input variables, from RPE(t_j), then it calculates the new value of the register and stores it to R_q. Then it sends the new value to all protocol entities which may need it to execute future I/O events and substitution statements of registers, or to evaluate guard expressions of future transitions. These messages are called *register value transfer messages.*

In Fig. 5.2, the components MAIN-1, MAIN-2 and MAIN-3 are derived according to the policy (a). Each of the components is derived by replacing a transition in *Sspec* with a subnet. For example, PE_2 and PE_3 execute the I/O events of t_2 and t_3 ("$b?y$" and "$c?z$") and send I/O completion notification messages "$Mn_{2.21}$" and "$Mn_{3.31}$"[1] to PE_1, respectively. In this case, in PE_2,

[1] A message ID is defined as "$Mt_{i.xy}$", where t is its message type (n, i and r denote I/O

the transition t_2 in *Sspec* is replaced by a subnet which has two transitions. One executes the I/O event of t_2 and the another sends the message "$\mathrm{Mn}_{2.21}$" after that. PE_1 can know that the execution of the I/O events of t_2 and t_3 has been finished. The rest of the components in Fig. 5.2 are derived according to the policy (b). They are derived by adding some transitions or sub-nets. PE_2 sends an input value transfer message "$\mathrm{Mi}_{2.23}$" to PE_3, which includes the value of input variable y. In this case, a transition sending the message "$\mathrm{Mi}_{2.23}$" is added to the net MAIN-2 in PE_2. PE_3 receives the message, calculates the new value of R_3, and sends PE_1 the register value transfer message "$\mathrm{Mr}_{2.31}$", which includes the new value of R_3. Also in this case, in PE_3, the net which is a sequence of three transitions (a receiving transition, a transition executing the substitution statement and a sending transition) is added. PE_1 receives the message and it can know the latest value of R_3, which is necessary for executing the I/O event of t_4. PE_1 has a transition receiving the message which forms a self-loop.

5.5.2 Deciding Time Constraints

In step (2), time constraints of $Pspec^{\langle 1,p \rangle}$ are decided. In $Pspec^{\langle 1,p \rangle}$ derived from step (1), only the execution order of the I/O events in *Sspec* is implemented by exchanging I/O completion notification messages. Therefore, in order to obtain a correct $Pspec^{\langle 1,p \rangle}$, the following three issues must be guaranteed in step (2).
(a) Every time interval between a pair of successive I/O events in $Pspec^{\langle 1,p \rangle}$ must satisfy the corresponding time interval in *Sspec*.
(b) The input/output values of I/O event sequences in $Pspec^{\langle 1,p \rangle}$ must be equivalent to those in *Sspec*. In our simulation policy in Section 5.5.1, the execution of an I/O event which needs the latest value of a register does not wait for the arrival of the register value transfer message (as a result, the I/O event may be executed with an old value). Therefore, we need guarantee that each register value transfer message has always arrived before the I/O event becomes executable.

completion notification, input value transfer and register value transfer, respectively), i is the transition name concerned with the message, and $\mathrm{PE}_x/\mathrm{PE}_y$ are the source/destination protocol entities, respectively.

(c) Considering the case discussed in Section 5.4, selectable transitions in each choice structure in $Sspec$ must be also selectable in $Pspec^{\langle 1,p \rangle}$. □

However, since there are message delays, such time constraints of $Pspec^{\langle 1,p \rangle}$ that satisfy all the above (a), (b) and (c) may not exist. For example, the message "$\mathrm{Mr}_{3.31}$" includes the value of R_3 necessary for the execution of the I/O event of t_4 in PE_1. Although it is sent immediately after the execution of the substitution statement of R_3, it may not arrive at PE_1 in time (therefore, it may not be able to satisfy the condition (b)). On the other hand, if we delay the earliest executable time of the I/O event of t_4, the message may be able to arrive at PE_1 before I/O event of t_4 becomes executable.

In step (2), the time constraints of $Pspec^{\langle 1,p \rangle}$ derived from step (1) are represented by some non-negative variables and the conditions (a), (b) and (c) are represented as linear inequalities over those variables. If there exists a solution which satisfies all of the inequalities, the time constraints of $Pspec^{\langle 1,p \rangle}$ will satisfy the above conditions (a), (b) and (c). Using a procedure to solve linear programming problems, a solution can be obtained where the total sum of ranges of time constraints in $Pspec^{\langle 1,p \rangle}$ is maximized.

[Introduced Variables]

• For each transition in $Pspec^{\langle 1,p \rangle}$ executing the I/O event of t_i, we introduce two non-negative variables, and represent the time constraint of the transition as $[ETmin(t_i), ETmax(t_i)]$. $ETmin(t_i)$ and $ETmax(t_i)$ represent the minimum and maximum time from the time when it receives all the I/O completion notification messages to the time when it executes the I/O event of t_i, respectively.

• For each transition in $Pspec^{\langle 1,p \rangle}$ executing the substitution statement of register R_q in t_i, we also introduce two non-negative variables, and represent the time constraint of the transition as $[RTmin(R_q, t_i), RTmax(R_q, t_i)]$. $RTmin(R_q, t_i)$ and $RTmax(R_q, t_i)$ represent the minimum and maximum time from the time when it receives the input value transfer message to the time when it calculates the new value of R_q, respectively.

• We assume that the time constraint of each sending/receiving transition is **[0,** 0]. That is, we assume every sending/receiving event is executed immediately after it becomes executable, as explained in Section 5.3.

[Linear Inequalities]

For Condition (a):

- For each pair of a transition t_j and its previous transition t_i :

$$ETmin(t_j) \le ETmax(t_j) \tag{5.1}$$

$$\mathrm{Eft}(t_j) \le Dmin_{uv}{+}ETmin(t_j) \le Dmax_{uv}{+}ETmax(t_j) \le \mathrm{Lft}(t_j) \tag{5.2}$$

  where we assume that RPE(t_i) and RPE(t_j) are protocol entities PE$_u$ and PE$_v$, respectively.

For Condition (b):

- For each transition t_j in *Sspec* and each register R_q whose value is substituted in t_j :

$$RTmin(R_q, t_j) \le RTmax(R_q, t_j) \tag{5.3}$$

- For each transition t_i in *Sspec* and each register R_q whose value is substituted in t_i, suppose that the new value of R_q is used for the execution of the I/O event of a future transition t_j in *Sspec*. Assume that RPE(t_i) and RPE(t_j) are protocol entities PE$_u$ and PE$_w$ respectively, and that a protocol entity PE$_v$ has the register R_q. Then the following must hold :

$$min\_seq(t_i, t_j)_h \ge Dmax_{uv} + RTmax(R_q, t_i) + Dmax_{vw} \tag{5.4}$$

  where each $min\_seq(t_i, t_j)_h$ $(h = 1, 2, ..., r)$ is the minimum time from the I/O event of t_i has been executed to the I/O event of t_j becomes executable in $Pspec^{\langle 1,p \rangle}$ (the minimum execution time of I/O event sequences in $Pspec^{\langle 1,p \rangle}$).

Similarly, if the new value of R_q is used for the evaluation of the guard expression's value of a future transition t_j in $Sspec$, the following must hold.

$$min\_seq(t_i, t_j)_h - ETmin(t_j) \geq Dmax_{uv} + RTmax(R_q, t_i) + Dmax_{vw} \tag{5.5}$$

If the new value of R_q is used for the execution of the register R_s's value substitution statement of a future transition t_j in $Sspec$, the followings must hold :

$$\begin{aligned} & min\_seq(t_i, t_j)_h + Dmin_{wz} + RTmin(R_s, t_j) \\ & \quad \geq \; Dmax_{uv} + RTmax(R_q, t_i) + Dmax_{vw} \qquad (5.6) \\ & Dmax_{wz} + RTmax(R_s, t_j) \\ & \quad \leq \; min\_seq(t_j, t_i)_k + Dmin_{uv} + RTmin(R_q, t_i) + Dmin_{vw} \qquad (5.7) \end{aligned}$$

where PE$_z$ holds the register R_s.

For Condition (c):

- For each pair of transitions t_{j_a} and t_{j_b} which share an input place p and a transition t_i which has p as an output place, assume that RPE(t_i) is a protocol entity PE$_u$ and RPE(t_{j_a}) and RPE(t_{j_b}) are the same protocol entity PE$_v$ (See Restriction 4 in Section 5.3). If Eft(t_{j_a}) $\leq$ Lft(t_{j_b}) and Eft(t_{j_b}) $\leq$ Lft(t_{j_a}) hold.

$$\begin{aligned} ETmin(t_{j_a}) & \leq ETmax(t_{j_b}) \\ ETmin(t_{j_b}) & \leq ETmax(t_{j_a}) \end{aligned} \tag{5.8}$$

Objective Function:

$$OBJ = \sum_{t_i} (\ ETmax(t_i) - ETmin(t_i)\ ) + \sum_{R_q} \sum_{t_j} (\ RTmax(R_q, t_j) - RTmin(R_q, t_j)\ )$$

For the condition (a), two types of inequalities are given. Inequality (5.1) is obviously necessary from the definition of those variables. Inequality (5.2) guarantees that even if there is the delay of an I/O completion notification message between two successive I/O events, the time interval between the I/O events in $Pspec^{\langle 1,p\rangle}$ must be within that in $Sspec$.

For the condition (b), two types of inequalities are also given. Inequality (5.3) is also necessary from the definition of those variables like Inequality (5.1). Inequality (5.4) guarantees that the new value of R_q necessary for the execution of an I/O event of t_j must arrive at PE$_w$ before the I/O event becomes executable. We consider the time when the I/O event of t_i has been executed as the base time (denoted by T). The earliest time when the I/O event of t_j becomes executable can be represented as $T + min\_sequence(t_i, t_j)_h$ $(h = 1, 2, ..., r)$. On the other hand, the latest time when the value of R_q arrives at PE$_w$ can be represented as $T + Dmax_{uv} + RTmax(R_q, t_i) + Dmax_{vw}$. Therefore if Inequality (5.4) holds, the new value of R_q necessary for the execution of an I/O event of t_j is in time for the executable time of the I/O event of t_j. This is the main idea of the register value transfer. For the similar reason, Inequalities (5.5) and (5.6) are necessary.

Here, if the value of R_q is used for the execution of the I/O event or the evaluation of the guard expression, we can guarantee that the next new value of R_q on t_i arrives at PE$_w$ after the I/O event (guard-expression) of t_j has already been executed (evaluated), since there is a message sequence between the I/O event (guard-expression) of t_j and the substitution of R_q's value on t_i. However, there may be no dependency between the substitution of R_s on t_j and the substitution of R_q on t_i. For example, suppose that t_i and t_j form a loop. t_i and t_j substitute the values of R_q and R_s, respectively. Also suppose that PE$_v$ and PE$_z$ have R_q and R_s, respectively, and PE$_z$ needs the value of R_q for the substitution of the value of R_s. In this case, PE$_z$ receives the new value of R_q and substitutes the value of R_s. However, any protocol entity does not wait for the finish of the substitution, therefore, PE$_v$ may start the next substitution of R_q before the finish of the substitution of R_s and may send the new value to PE$_z$. So the substitution of R_s may be done with next new value. Here, in order to guarantee the execution order of the substitution, we assume Inequality (5.7).

For the condition (c), Inequality (5.8) is given. $\mathrm{Eft}(t_{j_a}) \leq \mathrm{Lft}(t_{j_b})$ and $\mathrm{Eft}(t_{j_b}) \leq \mathrm{Lft}(t_{j_a})$ represent that both t_{j_a} and t_{j_b} are selectable in *Sspec*. Inequality (5.8) represents the condition to guarantee that t_{j_a} and t_{j_b} are selectable in $Pspec^{\langle 1,p \rangle}$.

Finally, we get a solution which satisfies all of Inequalities (5.1) - (5.8) using a procedure for solving linear programming problems. If such a solution exists, we regard that the time constraints in *Sspec* also hold in $Pspec^{\langle 1,p \rangle}$. Here, we would like to get elastic time constraints whose ranges are possibly wide (this is our evaluation basis for optimization). Therefore, we represent the total sum of the ranges of time constraints in $Pspec^{\langle 1,p \rangle}$ as the objective function OBJ, where the first and second terms represent the time range of the I/O events and the register value substitutions, respectively. Then we get an optimal solution to maximize the objective function OBJ.

Example

The following example will help readers to understand the detail of the algorithm.

For the transitions executing the I/O events of t_1, t_2, t_3 and t_4, the following inequalities must hold according to Inequalities (5.1) and (5.2).

$$\begin{aligned}
\mathrm{Eft}(t_1) \leq Dmin_{11} + ETmin(t_1) &\leq Dmax_{11} + ETmax(t_1) \leq \mathrm{Lft}(t_1) \\
\mathrm{Eft}(t_2) \leq Dmin_{12} + ETmin(t_2) &\leq Dmax_{12} + ETmax(t_2) \leq \mathrm{Lft}(t_2) \\
\mathrm{Eft}(t_3) \leq Dmin_{13} + ETmin(t_3) &\leq Dmax_{13} + ETmax(t_3) \leq \mathrm{Lft}(t_3) \\
\mathrm{Eft}(t_4) \leq Dmin_{21} + ETmin(t_4) &\leq Dmax_{21} + ETmax(t_4) \leq \mathrm{Lft}(t_4) \\
\mathrm{Eft}(t_4) \leq Dmin_{31} + ETmin(t_4) &\leq Dmax_{31} + ETmax(t_4) \leq \mathrm{Lft}(t_4)
\end{aligned}$$

As a result, the following inequalities can be obtained.

$$\begin{aligned}
0 \leq ETmin(t_1) &\leq ETmax(t_1) \leq 5 \\
0 \leq ETmin(t_2) &\leq ETmax(t_2) \leq 3 \\
3 \leq ETmin(t_3) &\leq ETmax(t_3) \leq 3 \\
2 \leq ETmin(t_4) &\leq ETmax(t_4) \leq 4 \\
1 \leq ETmin(t_4) &\leq ETmax(t_4) \leq 5
\end{aligned}$$

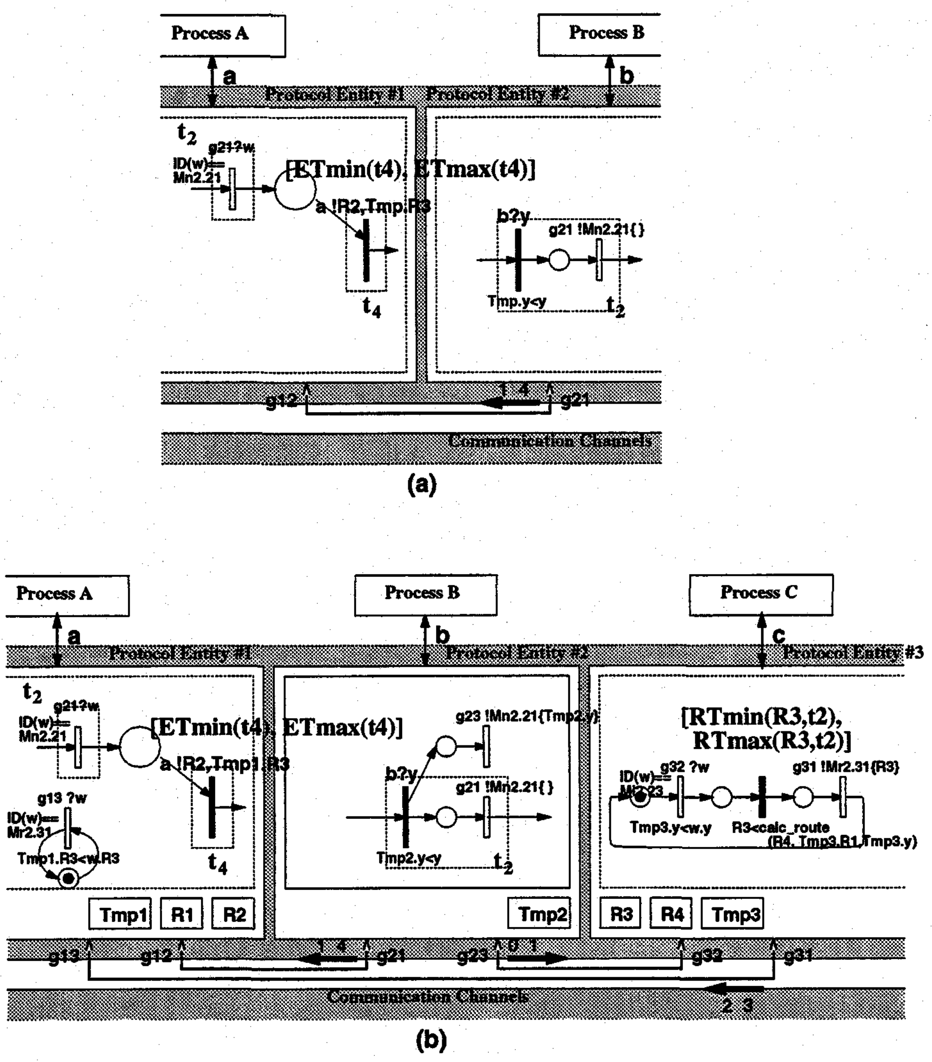


Figure 5.5: Example.

Then, for the transition t_1 and register R_1, the calculated value of R_1 is used for the register value substitution statement of t_2. Also for the transition t_2 and register R_3, the calculated value of R_3 is used for the I/O event of t_4. Similarly, for the transition t_3 and register R_2, the calculated value of R_2 is used for the I/O event of t_4. Therefore, according to Inequalities (5.3) and (5.4) , the following inequalities must hold.

$$0 \leq RTmin(R_1, t_1) \leq RTmax(R_1, t_1)$$

$$0 \leq RTmin(R_3, t_2) \leq RTmax(R_3, t_2)$$
$$0 \leq RTmin(R_2, t_3) \leq RTmax(R_2, t_3)$$

$$\begin{aligned}
&min\_seq(t_1, t_2)_1 + Dmin_{23} + RTmin(R_3, t_2) \\
&\quad \geq \; Dmax_{11} + RTmax(R_1, t_1) + Dmax_{13} \\
&min\_seq(t_2, t_4)_1 \\
&\quad \geq \; Dmax_{23} + RTmax(R_3, t_2) + Dmax_{31} \\
&min\_seq(t_3, t_4)_1 \\
&\quad \geq \; Dmax_{21} + RTmax(R_3, t_2) + Dmax_{11} \\
&min\_seq(t_4, t_2)_1 \\
&\quad \geq \; Dmax_{31} + RTmax(R_3, t_2) + Dmax_{11}
\end{aligned}$$

For the second inequality, $min\_seq(t_2, t_4)_1 = Dmin_{21} + ETmin(t_4)$. The left-side expression represents the earliest time when the I/O event of t_4 becomes executable, from the time T (when the I/O event of t_2 has been executed). On the other hand, the right-side expression represents the latest time when the register value transfer message "Mr$_{2.31}$" arrives at PE$_1$, from the same time T. As the result of the inequalities, the following inequalities can be obtained.

$$\begin{aligned}
&Dmin_{12} + ETmin(t_2) + Dmin_{23} + RTmin(R_3, t_2) \\
&\quad \geq \; Dmax_{11} + RTmax(R_1, t_1) + Dmax_{13} \\
&Dmin_{23} + ETmin(t_4) \\
&\quad \geq \; Dmax_{23} + RTmax(R_3, t_2) + Dmax_{31} \\
&Dmin_{31} + ETmin(t_4) \\
&\quad \geq \; Dmax_{31} + RTmax(R_2, t_3) + Dmax_{11}
\end{aligned}$$

That is,

$$\begin{aligned}
ETmin(t_2) + RTmin(R_3, t_2) - RTmax(R_1, t_1) &\geq -1 \\
ETmin(t_4) - RTmax(R_3, t_2) &\geq 3 \\
ETmin(t_4) - RTmax(R_2, t_3) &\geq 1
\end{aligned}$$

We illustrate how the time constraints of transitions are represented by variables in Fig 5.5.

Here, we give the following objective function and obtain a solution to minimize it.

$$\begin{aligned}
& OBJ \\
= \ & \sum_{i=1}^{4}(ETmax(t_i) - ETmin(t_i)) \\
+ \ & RTmax(R_1, t_1) - RTmin(R_1, t_1) \\
+ \ & RTmax(R_3, t_2) - RTmin(R_3, t_2) \\
+ \ & RTmax(R_2, t_3) - RTmin(R_2, t_3)
\end{aligned}$$

The solution is shown below.

$$\begin{aligned}
ETmin(t_1) &= 0, & ETmax(t_1) &= 5 \\
ETmin(t_2) &= 0, & ETmax(t_2) &= 3 \\
ETmin(t_3) &= 3, & ETmax(t_3) &= 3 \\
ETmin(t_4) &= 4, & ETmax(t_4) &= 4 \\
RTmin(R_1, t_1) &= 0, & RTmax(R_1, t_1) &= 1 \\
RTmin(R_3, t_2) &= 0, & RTmax(R_3, t_2) &= 0 \\
RTmin(R_2, t_3) &= 0, & RTmax(R_2, t_3) &= 3
\end{aligned}$$

5.6. Sketch of Correctness Proof

We show a sketch of correctness proof for our derivation algorithm in this section.

From the discussion of the correctness of protocol specifications in Section 5.4, we should show the following issues.

(1) The observable I/O event sequences (ignoring the time intervals between two successive events and the values of the sequences) of $Sspec$ and $Pspec^{\langle 1,p \rangle}$ are the same, except the sequences by the interleave of parallel transitions.

(2) The observable I/O values of the sequences are the same.

(3) Every time interval between two successive I/O events in $Pspec^{\langle 1,p\rangle}$ is within the corresponding time interval in $Sspec$.

We adopt a simulation policy where an I/O completion notification message is sent after the execution of each I/O event. Therefore, the execution order of I/O events of t_1,..., t_m in $Pspec^{\langle 1,p\rangle}$ is the same as that of t_1,..., t_m in $Sspec$, if we ignore the time intervals and I/O values. Here, the discussion of the correctness of removing ε-transitions is the same as that in Section 3.5. Therefore we omit the discussion. As a result, we can show that (1) holds.

Then in order to show that (2) holds, we should show that the following (2.1) and (2.2) hold.

(2.1) (a) the evaluation of the guard expression's value, (b) the execution of the I/O event and (c) the execution of register value substitution statements are done in this order.

(2.2) (a), (b) and (c) are done with correct values.

In the simulation policy, RPE(t_i) first evaluates the value of t_i's guard expression and executes t_i's I/O event. Each protocol entity which executes some of t_i's register value substitution statements cannot execute them without receiving an input value transfer message, which is sent after the execution of t_i's I/O event. Therefore, we can show that (2.1) holds.

In order to show that (2.2) holds, it is sufficient to show that both (2.2.1) and (2.2.2) hold.

(2.2.1) For each transition t_j which needs the value of a register R_q, a transition t_i which substitutes the value to the latest value is uniquely decided in $Sspec$.

(2.2.2) The value of the register R_q substituted on t_i has already arrived and the next new value has not arrived when the value is referred on t_j.

In our algorithm, we assume that there is no register conflict pair of transitions. Therefore, when we trace the Petri net graph of *Sspec* backward from t_j, we can find only one transition t_i where the value of R_q is substituted. Note that the net contains choice structures, so we may find more than one such a transition t_i. However, they never fire in parallel, therefore we can treat them as one transition t_i. As a result, we can show that (2.2.1) holds.

In our simulation policy, the protocol entity which holds the register R_q sends the substituted value of R_q to the protocol entity which uses the value for the simulation.of t_j. We restrict the time constraints of transitions in $Pspec^{\langle 1,p \rangle}$ to guarantee that the value always arrives before it is used. Inequalities (5.3)-(5.6) guarantee such a condition. As discussed in Section 5.5.2, the next new value of R_q must not arrive before the previous value is used. Inequality (5.7) guarantee such a condition. As a result of the above discussion, we can show that (2.2.2) holds.

Finally, for any pair of two successive I/O events, Inequality (5.2) guarantees that the time interval between them in $Pspec^{\langle 1,p \rangle}$ is within that in *Sspec*. However, when we consider time intervals, we must consider the special case of choice as discussed in Section 5.4. Here, Inequality (5.8) guarantees that the selectability of transitions in a choice structure in $Pspec^{\langle 1,p \rangle}$. We do not have to consider the interleaves of parallel I/O events due to the definition of correctness. Therefore we can show that (3) holds.

5.7. Discussion

We propose two different protocol synthesis methods although TPNR model is a superset of PNR model, because the available classes of service specifications are different. Also simulation policies are quite different.

Table 5.3 shows the comparison of two methods described in Chapter 3 and in this chapter.

The method in this chapter restricts the class of the Petri nets of service specifications as live and safe Free-Choice net. This is because choice structures in general Petri nets may make it difficult to guarantee the correctness of pro-

Table 5.3: Comparison.

	Method in Chapter 3	Method in this chapter
The Class of Petri Nets	**live and safe Petri net**	live and safe Free-Choice net
Register Conflict	**considered**	not considered
Time Constraint	not considered	**considered**

tocol specifications (for details, see Section 5.4). Also in the method, avoiding the register conflict problem is not considered, since it is not practical to adopt distributed mutual exclusion mechanisms. The distributed mutual exclusion mechanisms need additional messages and we must consider the delays of those messages in the derivation algorithm. Instead of those restrictions, we adopt a simulation policy where the values are sent as soon as possible for the future execution of actions. Such a simulation policy needs to extract data dependencies in service specifications.

On the other hand, the method in Chapter 3 only assumes liveness and safeness for the class of Petri nets. Also it considers the control of register conflict. Instead of those advantages, it adopts a simulation policy where each transition is simulated independently of the others. It facilitates to guarantee the correctness of protocol specifications.

5.8. Conclusion

In this chapter, a method to derive a correct protocol specification from a given service specification in TPNR model, a resource allocation and maximum/minimum communication delays among protocol entities has been proposed.

As briefly discussed, our concept is to provide a synthesis approach for distributed environments where (a) a global clock is not available and (b) state variables are considered. As a solution for (a), we use a time Petri net based model. In time Petri nets, time constraints can be specified between only successive actions, therefore it facilitates to implement service specifications in such a distributed environment. Nevertheless, it is commonly used and powerful enough for formal specifications of real-time systems. As a solution for (b), we extend time Petri nets, so that state variables can be treated formally. Also we adopt an implementation policy where data values are sent as early as possible (sent in advance) for the efficient implementation of service specifications. This concept is very common in real-time distributed systems.

For practical use, many system-specific things must be considered. For example, in distributed real-time databases, data with large capacity may be treated. Therefore the execution time of actions, such as the retrieval time of data, may not seem to be zero, while we assume it is zero in our method. To allow for such a case, we only change Inequality (5.4) to :

$$\begin{aligned} min\_seq(t_i, t_j)_h &> Dmax_{uv} + RTmax(R_q, t_i) \\ &\quad + max\_exec(R_q, t_i) + Dmax_{vw} \end{aligned}$$

where $max\_exec(R_q, t_i)$ represents the maximum execution time of the substitution statement of R_q. Our concept itself can be applied to many real-time distributed systems by slightly modifying the inequalities even though there are some system-specific aspects practically.

Chapter 6

Conclusion

In this thesis, the following three issues have been described.

1. A protocol synthesis method from service specifications written in an extended Petri net model has been proposed. The method provides how to simulate a service specification including parallelism and the calculation of state variables in a distributed environment, without inconsistency. In order to make the correctness of protocol specifications easy to prove, a simulation policy, where the behavior of each transition is simulated independently, is adopted. The number of messages exchanged for simulating the behavior of each transition based on the simulation policy is minimized.

2. An application of the proposed method has been shown. From a whole process description of cooperative work in PNR model and an allocation of tasks to workers, a set of process descriptions of workers is automatically derived. The derivation system which derives a set of process descriptions of workers and the execution system which interprets a process description of worker have been developed. Using the system, the efficiency of the synthesis method for cooperative work support is shown.

3. A protocol synthesis method from service specifications written in an extended time Petri net model has been proposed. The method provides how to simulate a service specification with time constraints in a distributed environment under the presence of communication delays. Therefore the

different cost measure, urgency of time, is introduced and an adequate simulation policy suitable for the cost measure should be adopted. In this method, a simulation policy, where each protocol entity holding the latest data sends it as soon as possible to the protocol entities which may need it in future is adopted even though it may lead unnecessary messages. By doing so, the total sum of the possible ranges of the executable time of the actions in a protocol specification is maximized.

In the first method, it has been shown that service specifications with parallelism and the calculation of state variables can be implemented correctly. The simulation policy implements each transition independently and it may make the correctness proof of the algorithm easy. In the third method, it has been also shown that service specifications with time constraints between successive actions can be implemented efficiently in a distributed environment where a global time is not available. In order to implement such service specifications efficiently, a simulation policy suitable for real-time properties is adopted. These methods are based on different concepts, therefore, the simulation policy and cost measure of one method are quite different from those of the another one.

The result of these researches may spread the class of service specifications in protocol synthesis methods and may be helpful to design reliable distributed computing systems. As one of future work, in order to treat more general class of service specifications, a method to derive protocol specifications, from service specifications where a number of the same services can be provided, will be considered. In practical, systems may have to provide a number of the same services, for example, several similar projects progress concurrently in cooperative work. For modeling a number of the same services simply, coloured Petri nets can be considered as a description model.

References

[1] Merlin, P.M. and Farber, D.J.: "Recoverability of Communication Protocols Implications of a Theoretical Study," *IEEE Trans. on Communications*, Vol. COM-24, pp. 1036–1043, 1976.

[2] Kosaraju, S.R.: "Decidability of Reachability in Vector Addition Systems," *Proc. of 14th Annual ACM Symp. on Theory Computing*, pp. 267–281, 1982.

[3] Mayr, E. W.: "An Algorithm for the General Petri Net Reachability Problem," *SIAM, J. Comput.*, Vol. 13, No. 3, pp. 441–460, Aug. 1984.

[4] Murata, T.: "Petri Nets: Properties, Analysis and Applications," *Proc. of IEEE*, Vol. 77, No. 4, pp. 541–580, 1989.

[5] Barkaoui, K. and Minoux, M.: "A Polynomial-Time Graph Algorithm to Decide Liveness of Some Basic Classes of Bounded Petri Nets," *Proc. of Int. Conf. on Application and Theory of Petri Nets 1992, LNCS*, Vol. 616, pp. 62–75, 1992.

[6] Kemper, P. and Bause, F.: "An Efficient Polynomial-Time Algorithm to Decide Liveness and Boundedness of Free-Choice Nets," *Proc. of Int. Conf. on Application and Theory of Petri Nets 1992, LNCS*, Vol. 616, pp. 263–278, 1992.

[7] Bucci, G. and Vicario, E.: "Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets," *IEEE Trans. on Software Engineering*, Vol. 21, No. 12, pp. 969–992, 1995.

[8] Bruno, G. and Balsamo, A.: "Petri Net-Based Object Oriented Modelling of Distributed Systems," *Proc. of ACM OOPSLA '86*, pp. 284–293, 1986.

[9] Berthomieu, B. and Diaz, M.: "Modeling and Verification of Time Dependent Systems Using Time Petri Nets," *IEEE Trans. on Software Engineering*, Vol. 17, No. 3, pp. 259–273, 1991.

[10] Probert, R. and Saleh, K.: "Synthesis of Communication Protocols: Survey and Assessment," *IEEE Trans. on Computers*, Vol. 40, No. 4, pp. 468–476, 1991.

[11] Saleh, K.: "Synthesis of Communication Protocols: an Annotated Bibliography," *ACM SIGCOMM Computer Communication Review*, Vol. 26, No. 5, pp. 40–59, 1996.

[12] Lamport, L.: "Time, Clocks, and Ordering of Events in a Distributed

System," *Communication of ACM*, Vol. 21, No. 7, pp. 558–564, 1978.

[13] Maekawa, M.: "A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems," *ACM Trans. on Computer Systems*, Vol. 3, No. 2, pp. 145–159, 1985.

[14] Lamport, L. and Lynch, N.A.: "Distributed Computing : Modes and Methods," *Hand Book of Theoretical Computer Science, B: Formal Models Semantics*, The MIT Press/Elsevier, pp. 1157–1200, 1990.

[15] Park, D.: "Concurrency and Automata on Infinite Sequences," *Theoretical Computer Science*, Vol. 104, pp. 167–183, 1981.

[16] Milner, R.: "Communication and Concurrency," *Prentice-Hall*, 1989.

[17] Bochmann, G.V. and Gotzhein, R.: "Deriving Protocol Specifications from Service Specifications," *Proc. of ACM SIGCOMM '86*, pp. 148–156, 1986.

[18] Khendek, F., Bochmann, G.V. and Kant, C.: "New Results on Deriving Protocol Specifications from Service Specifications," *Proc. of ACM SIGCOMM '86*, pp. 136–145, 1989.

[19] Gotzhein, R. and Bochmann, G.V.: "Deriving Protocol Specifications from Service Specifications Including Parameters," *ACM Trans. on Computer Systems*, Vol. 8, No. 4, pp. 255–283, 1990.

[20] Langerak, R.: "Decomposition of Functionality; a Correctness-Preserving LOTOS Transformation," *Proc. of 10th IFIP WG6.1 Symp. on Protocol Specification, Testing and Verification (PSTV-10)*, pp. 229–242, 1990.

[21] Higashino, T.: "Service Specification and Its Protocol Specifications in LOTOS," *IEICE Trans. on Fundamentals*, Vol. E75-A, No. 3, pp. 330–338, 1992.

[22] Kant, C., Higashino, T. and Bochmann, G.V.: "Deriving Protocol Specifications from Service Specifications Written in LOTOS," *Distributed Computing*, Vol. 10, No. 1, pp. 29–47, 1996.

[23] Chu, P.-Y.M. and Liu, M.T.: "Synthesizing Protocol Specifications from Service Specifications in FSM Model," *Computer Networking Symp. '88*, pp. 173–182, 1988.

[24] Chu, P.-Y.M. and Liu, M.T.: "Protocol Synthesis in a State-Transition Model," *Proc. of COMPSAC '88*, pp. 505–512, 1988.

[25] Higashino, T., Okano, K., Imajo, H. and Taniguchi, K.: "Deriving Protocol Specifications from Service Specifications in Extended FSM Models," *Proc. of 13th Int. Conf. on Distributed Computing Systems (ICDCS-13)*, pp. 141–148, 1993.

[26] Kakuda, Y., Igarashi, H. and Kikuno, T.: "Automated Synthesis of Protocol Specifications with Message Collisions and Verification of Timeliness," *Proc. of 1994 Int. Conf. on Network Protocols (ICNP'94)*, 1994.

[27] Osterweil, L.J.: "Software Processes Are Software Too," *Proc. of 9th Int. Conf. on Software Engineering (ICSE-9)*, pp. 2–13, 1987.

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

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

[30] Saeki, M., Kaneko, T. and Sakamoto, M.: "A Method for Software Process Modeling and Description using LOTOS," *Proc. of 1st Int. Conf. on Software Process (ICSP-1)*, pp. 90–104, 1991.

[31] Curtis, B., Kellner, M. and Over, J.: "Process Modeling," *Communication of ACM*, Vol. 35, No. 9, pp. 75–90, 1992.

[32] Iida, H., Mimura, K., Inoue, K. and Torii, K.: "Hakoniwa: Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model," *Proc. of 2nd Int. Conf. on Software Process (ICSP-2)*, pp. 64–74, 1993.

[33] Sutton, S., Heimbigner, D. and Osterweil, L. J.: "Language Constructs for Managing Change in Process Centered Environments," *Proc. of 4th SIGSOFT Symposium on Software Development Environments, Software Eng. Notes*, Vol. 15, No. 6, pp. 206–217, 1990.

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

[35] Huff, K.E. and Lessor, V.R. : "A Plan-based Intelligent Assistant that Supports the Software Development Process," *Proc. of 3rd Software Engineering Symposium on Practical Software Development Environments, Software Eng. Notes*, Vol. 13, No. 5, pp. 97–106, 1989.

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

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

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

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

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

[41] Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, M. and Picco, G.P.: "Modeling and Improving an Industrial Software Process," *IEEE Trans. on Software Engineering*, Vol. 21, No. 5, pp. 440–454, 1995.

[42] Leonhardt, U., Kramer, J., Nuseibeh, B. and Finkelstein, A.: "Decentralized Process Enactment in a Multi-Perspective Development Environment," *Proc. of 17th Int. Conf. on Software Engineering (ICSE-17)*, pp. 255–264, 1995.

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

[44] Yasumoto, K., Higashino, T. and Taniguchi, K.: "Software Process Description Using LOTOS and its Enaction," *Proc. of 16th Int. Conf. on Software Engineering (ICSE-16)*, pp. 169–179, 1994.

[45] Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K.: "Software Process Description in a Petri Net Model and Its Distributed Execution," *Technical Report of IEICE of Japan*, 94–SS–38, Vol. 94, No. 334, pp. 25–32, Nov. 1994.

[46] Koler, M.-K.: "Deriving Protocol Specifications from Service Specifications with Heterogeneous Timing Requirements," *Proc. of 1991 Int. Conf. on Software Engineering for Real Time Systems*, pp. 266–270, 1991.

[47] Khoumsi, A., Bochmann, G.V. and Dssouli, R.: "On Specifying Services and Synthesizing Protocols for Real-time Applications," *Proc. of 14th IFIP WG6.1 Symp. on Protocol Specification, Testing and Verification (PSTV-14)*, pp. 185–200, 1994.

[48] Nakata, A., Higashino, T. and Taniguchi, K.: "Protocol Synthesis from Timed and Structured Specifications," *Proc. of 1995 Int. Conf. on Network Protocols (ICNP'95)*, pp. 74–81, 1995.

[49] Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K.: Protocol Synthesis from Time Petri Net Based Service Specifications, *Proc. of 1997 Int. Conf. on Parallel and Distributed Systems (ICPADS'97)*, pp. 236–243,

Dec. 1997.

[50] Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K.: "Synthesis of Protocol Specifications from Service Specifications of Distributed Systems in a Marked Graph Model," *IEICE Trans. on Fundamentals*, Vol. E77-A, No. 10, pp. 1623–1633, 1994.

[51] Chao, D.Y. and Wang, D.T.: "A Synthesis Technique of General Petri Nets," *Journal of System Integration*, Vol. 4, pp. 67–102, 1994.

[52] Yamaguchi, H., Okano, K., Higashino, T. and Taniguchi, K.: "Synthesis of Protocol Entities' Specifications from Service Specifications in a Petri Net Model with Registers," *Proc. of 15th Int. Conf. on Distributed Computing Systems (ICDCS-15)*, pp. 510–517, 1995.

[53] Kahlouche, H. and Girardot, J.J.: "A Stepwise Requirement Based Approach for Synthesizing Protocol Specifications in an Interpreted Petri Net Model," *Proc. of INFOCOM'96*, pp. 1165–1173, 1996.

Appendix

In Chapter 4, we have applied our derivation method in Chapter 3 to cooperative work support. In this appendix, we will show an another application.

We model a work-flow of a manufacturing system [8]. We describe the overall work-flow of the system in PNR model and regard it as a service specification. Then we derive a set of protocol entity specifications of six computers.

The system consists of six computers PE_1, ..., PE_6 (Fig. 1). For each pair of PE_i and PE_j, they are connected by a communication channel and both end points of the channel are represented as I/O gates g_{ij} (PE_i's side) and g_{ji} (PE_i's side). There are three processing machines 1, 2 and 3, an operator, a stockroom operator and a carrier. The operator uses the computer PE_4 and indicates the system to process the parts of a product. Also it watches the behavior of the processing machines and carrier. These are done through the I/O gate OP on PE_4. The stockroom operator uses the computer PE_5 and checks whether the required parts are in a stockroom or not. If so, it takes the parts out from the stockroom. These are done through the I/O gate STK on PE_5. The carrier carries the parts from the stockroom to one of three processing machines. PE_6 controls the carrier through the I/O gate CR on PE_6. The processing machines 1, 2 and 3 process the parts carried from the stockroom. PE_1, PE_2 and PE_3 control the processing machines 1, 2 and 3 through I/O gates $MC1$, $MC2$ and $MC3$ on PE_1, PE_2 and PE_3, respectively.

The system has two types of databases. One keeps a stock list and the another keeps a parts information list. They are represented as registers R_{sdb} and R_{cdb}, respectively. Each entry in the stock list consists of a parts name, the number of the parts in the stockroom and the type of the parts. Also each entry in the parts information list consists of a parts name and the information

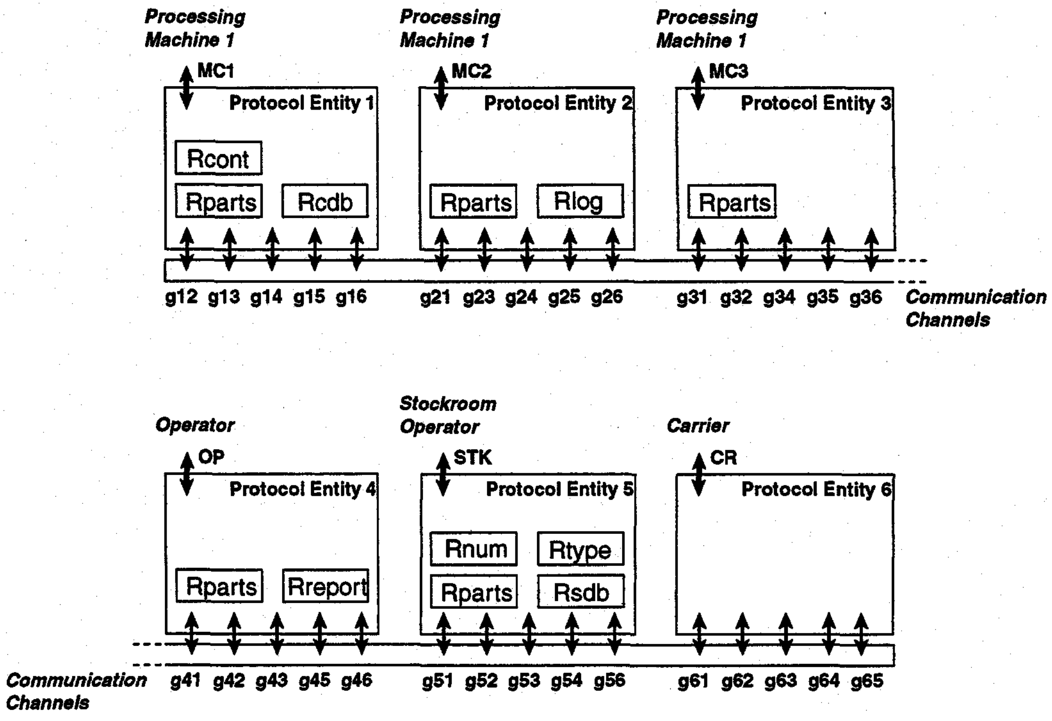


Figure 1: Computers and System Resources.

of the parts. The register R_{log} keeps the system logs. The registers R_{parts}, R_{num}, R_{report} and R_{cont} are used as variables to hold input data, computation results, and so on. Each register is placed on at least one computer.

First, we assume that all of the I/O gates and registers can be used and all the communication channels and their end points are hidden. Then we describe the whole behavior of the system in PNR model. It is shown in Figures 2 and 3.

The system receives the parts name to be processed, from the operator (through the I/O gate OP) on the transition t_1. Also it retrieves the entry of the parts from the parts list R_{sdb} and the result of the retrieval is stored to the register R_{num}. If the number of the parts in the retrieved entry (the value of R_{num}) is equal to zero, the system tells the stockroom operator to supply the parts on the transition t_2, then tells the operator that the parts are out of stock on the transition t_3 and then goes back to the initial state. If the value of R_{num}

is less than zero, the inconsistency has occurred. In this case, the system tells the stockroom operator to correct the inconsistency on the transition t_4, then tells the operator that the parts are out of stock on the transition t_5 and goes back to the initial state. Otherwise the system lets the stockroom operator to check whether the parts are really in stock or not. If the system is told that the parts are out of stock on the transition t_7, the system tells the operator that the inconsistency occurs, then lets the stockroom operator to supply the parts, and goes back to the initial state. Otherwise the system tells the stockroom operator to carry the parts out from the stockroom on the transition t_{11}, and receives the signal of the finish on the transition t_{12}. At this time, the contents of the database R_{sdb} is updated and the type of the parts is retrieved from R_{sdb}. The result of the retrieval is stored to the register R_{type}. After that the system lets the carrier to carry the parts from the stockroom to one of three processing machines according to the type of the parts (the value of R_{type}). It is done by the choice of the transitions t_{14}, t_{15} or t_{16}. Then the system receives the signal of the finish on the transition t_{18}. Now we assume that the processing machine 1 receives the parts. For the processing machine 1, the system lets the machine to mount the parts on the transition t_{20}, receives the signal of the finish on the transition t_{24}, lets it to process the parts on the transition t_{27} and receives the signal of the finish on the transition t_{30}. Here, on the transition t_{30}, the system receives not only the signal but also the report of the processing. Then the system checks whether the processing has been successfully completed or not. It is done by the choice of t_{34} or t_{35}. If not, the system lets the machine to reset itself on the transition t_{39}, tells the operator that the process has not been completed, and then goes to the initial state. Otherwise the system lets the machine to unmount the parts on the transition t_{36} and tells the operator that the process has been completed. Then the system goes back to the initial state. Here, the system records the behavior of the machines and carrier onto the log-file R_{log}. These are done on the transitions t_{13}, t_{17}, t_{19}, t_{23} and so on, in parallel with the main work-flow of processing.

Then we regard the description in Figures 2 and 3 as a service specification, and derive a protocol specification on the six computers in Figure 1. The derived protocol entity specifications are shown in Figures 4 and 5 ($Pspec_1$), Figures 6 and 7 ($Pspec_2$), Figure 8 ($Pspec_3$), Figures 9 and 10 ($Pspec_4$), Figures 11 and 12 ($Pspec_5$), and Figure 13 ($Pspec_6$).

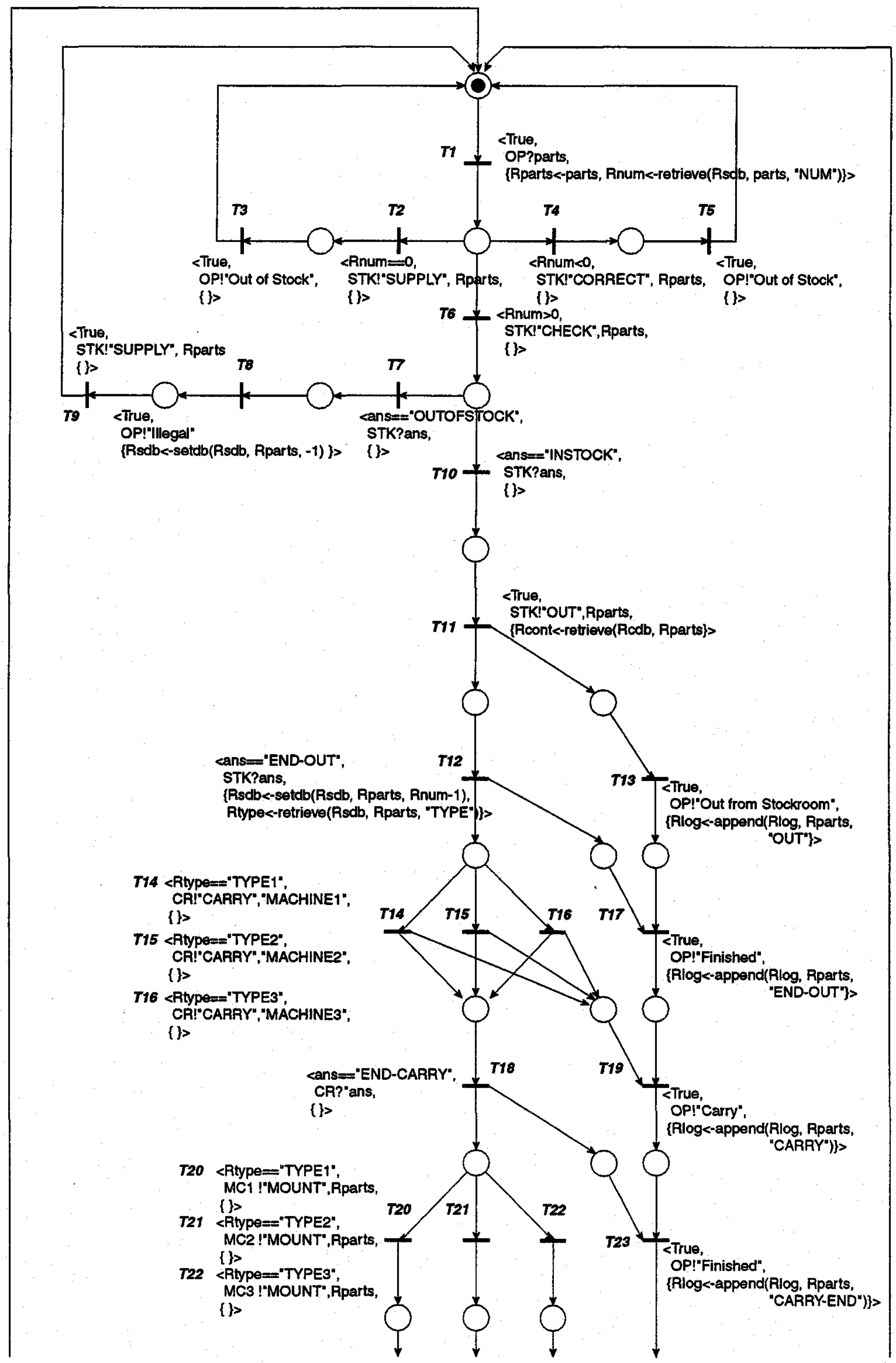

T1 <True, OP?parts, {Rparts<-parts, Rnum<-retrieve(Rsdb, parts, "NUM")}>
T3 <True, OP!"Out of Stock", { }>
T2 <Rnum==0, STK!"SUPPLY", Rparts, { }>
T4 <Rnum<0, STK!"CORRECT", Rparts, { }>
T5 <True, OP!"Out of Stock", { }>
T6 <Rnum>0, STK!"CHECK",Rparts, { }>
<True, STK!"SUPPLY", Rparts { }>
T8
T7
T9 <True, OP!"Illegal" {Rsdb<-setdb(Rsdb, Rparts, -1) }>
<ans=="OUTOFSTOCK", STK?ans, { }>
T10 <ans=="INSTOCK", STK?ans, { }>
T11 <True, STK!"OUT",Rparts, {Rcont<-retrieve(Rcdb, Rparts}>
<ans=="END-OUT", STK?ans, {Rsdb<-setdb(Rsdb, Rparts, Rnum-1), Rtype<-retrieve(Rsdb, Rparts, "TYPE")}>
T12
T13 <True, OP!"Out from Stockroom", {Rlog<-append(Rlog, Rparts, "OUT"}>
T14 <Rtype=="TYPE1", CR!"CARRY","MACHINE1", { }>
T15 <Rtype=="TYPE2", CR!"CARRY","MACHINE2", { }>
T16 <Rtype=="TYPE3", CR!"CARRY","MACHINE3", { }>
T14
T15
T16
T17 <True, OP!"Finished", {Rlog<-append(Rlog, Rparts, "END-OUT"}>
<ans=="END-CARRY", CR?"ans, { }>
T18
T19 <True, OP!"Carry", {Rlog<-append(Rlog, Rparts, "CARRY")}>
T20 <Rtype=="TYPE1", MC1 !"MOUNT",Rparts, { }>
T21 <Rtype=="TYPE2", MC2 !"MOUNT",Rparts, { }>
T22 <Rtype=="TYPE3", MC3 !"MOUNT",Rparts, { }>
T20
T21
T22
T23 <True, OP!"Finished", {Rlog<-append(Rlog, Rparts, "CARRY-END")}>

Figure 2: Specification of Manufacturing System. (1 of 2)

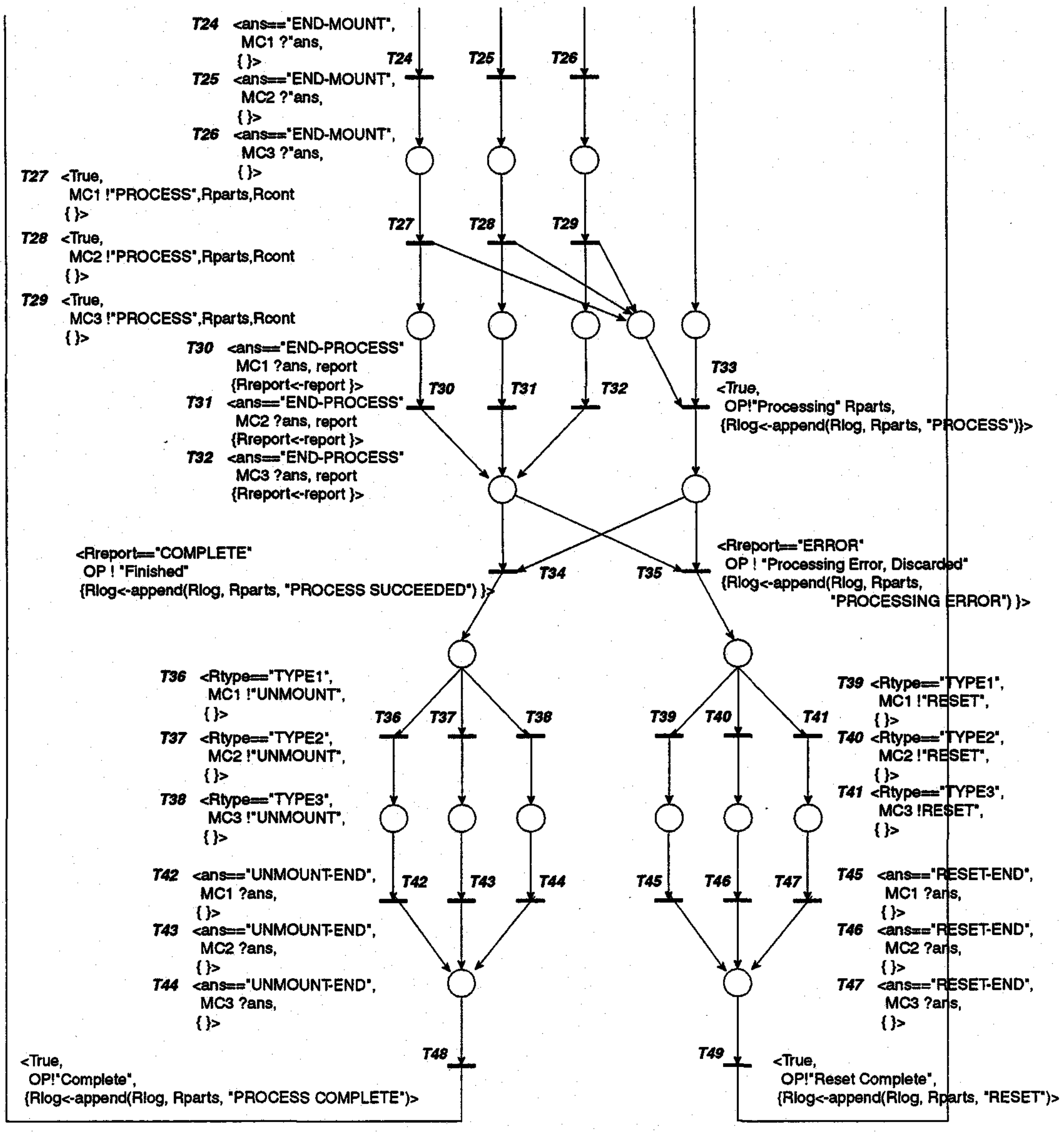

T24 <ans=="END-MOUNT", MC1 ?"ans, { }>
T25 <ans=="END-MOUNT", MC2 ?"ans, { }>
T26 <ans=="END-MOUNT", MC3 ?"ans, { }>
T27 <True, MC1 !"PROCESS",Rparts,Rcont { }>
T28 <True, MC2 !"PROCESS",Rparts,Rcont { }>
T29 <True, MC3 !"PROCESS",Rparts,Rcont { }>
T30 <ans=="END-PROCESS" MC1 ?ans, report {Rreport<-report }>
T31 <ans=="END-PROCESS" MC2 ?ans, report {Rreport<-report }>
T32 <ans=="END-PROCESS" MC3 ?ans, report {Rreport<-report }>
T33 <True, OP!"Processing" Rparts, {Rlog<-append(Rlog, Rparts, "PROCESS")}>
<Rreport=="COMPLETE" OP ! "Finished" {Rlog<-append(Rlog, Rparts, "PROCESS SUCCEEDED") }>
<Rreport=="ERROR" OP ! "Processing Error, Discarded" {Rlog<-append(Rlog, Rparts, "PROCESSING ERROR") }>
T34
T35
T36 <Rtype=="TYPE1", MC1 !"UNMOUNT", { }>
T37 <Rtype=="TYPE2", MC2 !"UNMOUNT", { }>
T38 <Rtype=="TYPE3", MC3 !"UNMOUNT", { }>
T39 <Rtype=="TYPE1", MC1 !"RESET", { }>
T40 <Rtype=="TYPE2", MC2 !"RESET", { }>
T41 <Rtype=="TYPE3", MC3 !RESET", { }>
T42 <ans=="UNMOUNT-END", MC1 ?ans, { }>
T43 <ans=="UNMOUNT-END", MC2 ?ans, { }>
T44 <ans=="UNMOUNT-END", MC3 ?ans, { }>
T45 <ans=="RESET-END", MC1 ?ans, { }>
T46 <ans=="RESET-END", MC2 ?ans, { }>
T47 <ans=="RESET-END", MC3 ?ans, { }>
T48
<True, OP!"Complete", {Rlog<-append(Rlog, Rparts, "PROCESS COMPLETE")}>
T49
<True, OP!"Reset Complete", {Rlog<-append(Rlog, Rparts, "RESET")}>

Figure 3: Specification of Manufacturing System. (2 of 2)

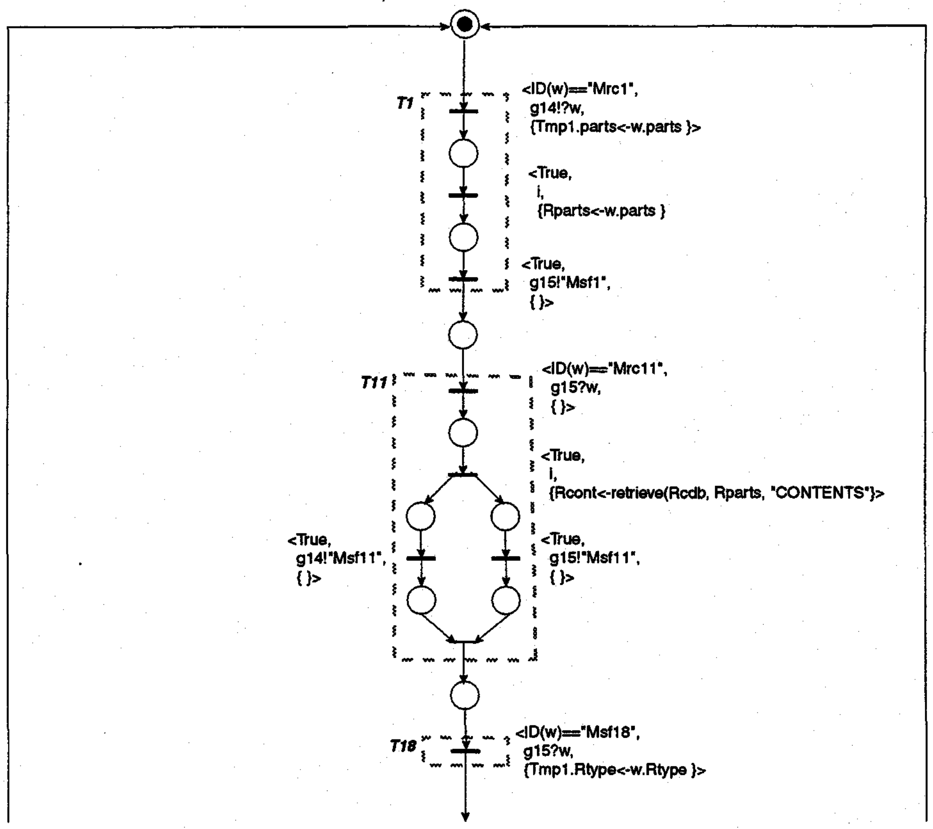
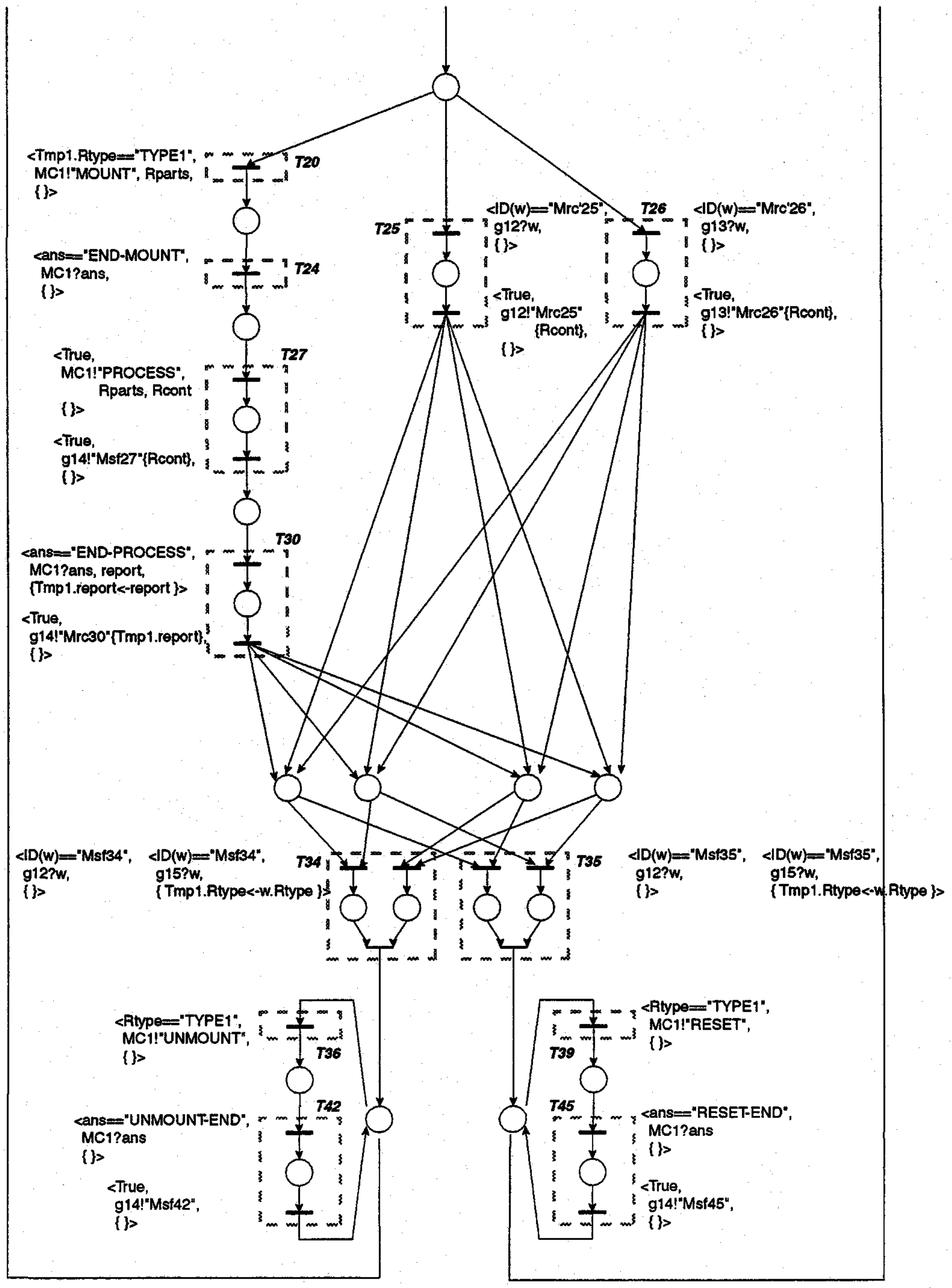


Figure 4: Specification for PE_1. (1 of 2)



<Tmp1.Rtype=="TYPE1", MC1!"MOUNT", Rparts, { }>
T20
<ans=="END-MOUNT", MC1?ans, { }>
T24
<True, MC1!"PROCESS", Rparts, Rcont { }>
T27
<True, g14!"Msf27"{Rcont}, { }>
<ans=="END-PROCESS", MC1?ans, report, {Tmp1.report<-report }>
T30
<True, g14!"Mrc30"{Tmp1.report}, { }>
T25
<ID(w)=="Mrc25", g12?w, { }>
<True, g12!"Mrc25" {Rcont}, { }>
T26
<ID(w)=="Mrc26", g13?w, { }>
<True, g13!"Mrc26"{Rcont}, { }>
<ID(w)=="Msf34", g12?w, { }>
<ID(w)=="Msf34", g15?w, { Tmp1.Rtype<-w.Rtype }>
T34
T35
<ID(w)=="Msf35", g12?w, { }>
<ID(w)=="Msf35", g15?w, { Tmp1.Rtype<-w.Rtype }>
<Rtype=="TYPE1", MC1!"UNMOUNT", { }>
T36
<ans=="UNMOUNT-END", MC1?ans { }>
T42
<True, g14!"Msf42", { }>
<Rtype=="TYPE1", MC1!"RESET", { }>
T39
<ans=="RESET-END", MC1?ans { }>
T45
<True, g14!"Msf45", { }>

Figure 5: Specification for PE_1. (2 of 2)

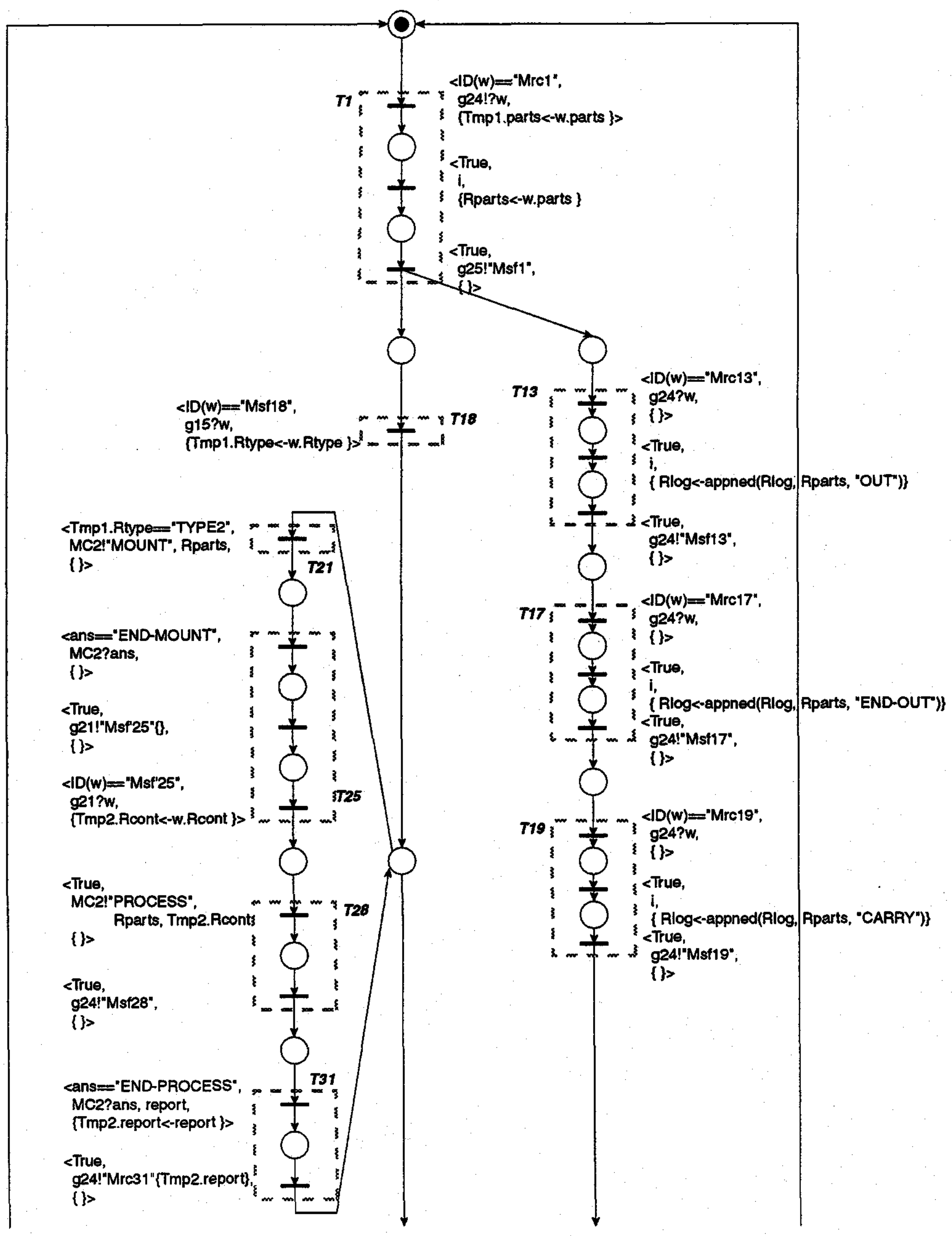


Figure 6: Specification for PE_2. (1 of 2)

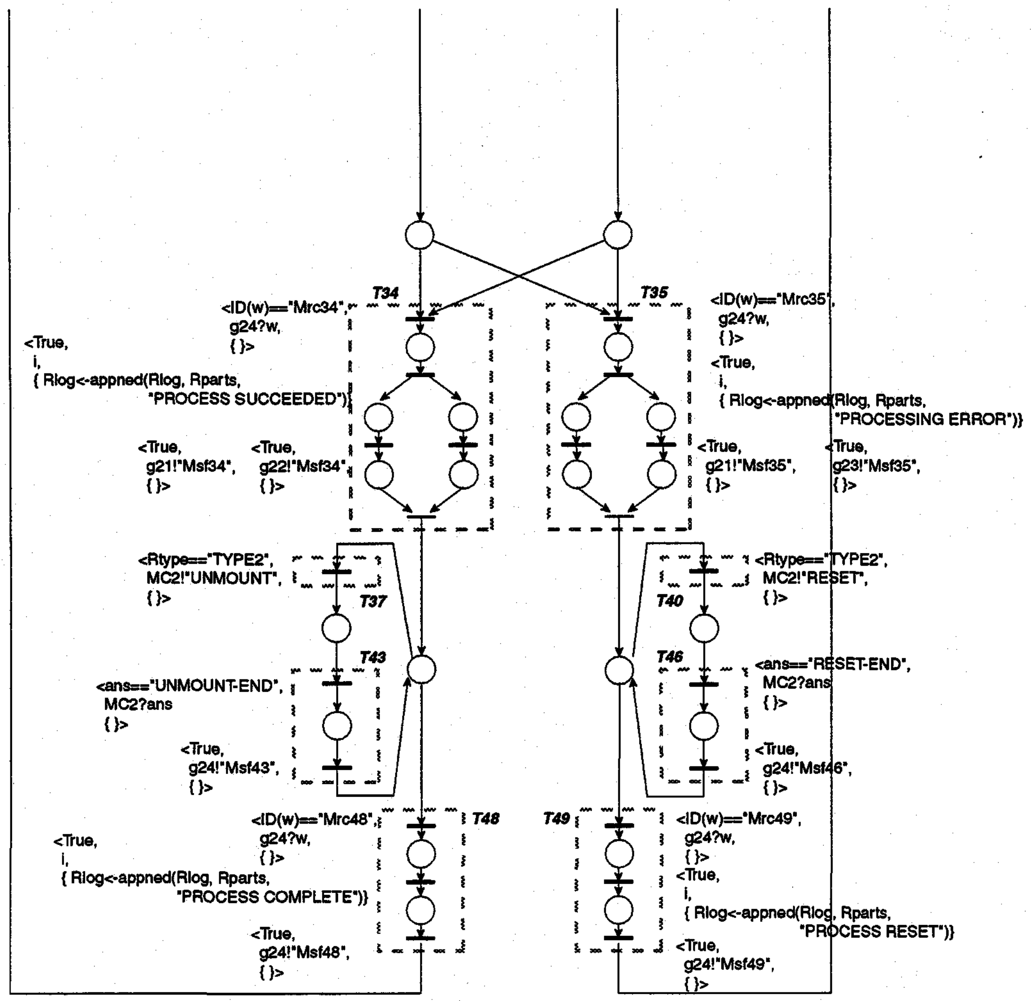

T34
T35
<ID(w)=="Mrc34", g24?w, { }>
<True, i, { Rlog<-appned(Rlog, Rparts, "PROCESS SUCCEEDED")}
<True, g21!"Msf34", { }>
<True, g22!"Msf34", { }>
<ID(w)=="Mrc35", g24?w, { }>
<True, i, { Rlog<-appned(Rlog, Rparts, "PROCESSING ERROR")}
<True, g21!"Msf35", { }>
<True, g23!"Msf35", { }>
<Rtype=="TYPE2", MC2!"UNMOUNT", { }>
T37
T43
<ans=="UNMOUNT-END", MC2?ans { }>
<True, g24!"Msf43", { }>
<Rtype=="TYPE2", MC2!"RESET", { }>
T40
T46
<ans=="RESET-END", MC2?ans { }>
<True, g24!"Msf46", { }>
<ID(w)=="Mrc48", g24?w, { }>
T48
<True, i, { Rlog<-appned(Rlog, Rparts, "PROCESS COMPLETE")}
<True, g24!"Msf48", { }>
T49
<ID(w)=="Mrc49", g24?w, { }>
<True, i, { Rlog<-appned(Rlog, Rparts, "PROCESS RESET")}
<True, g24!"Msf49", { }>

Figure 7: Specification for PE_2. (2 of 2)

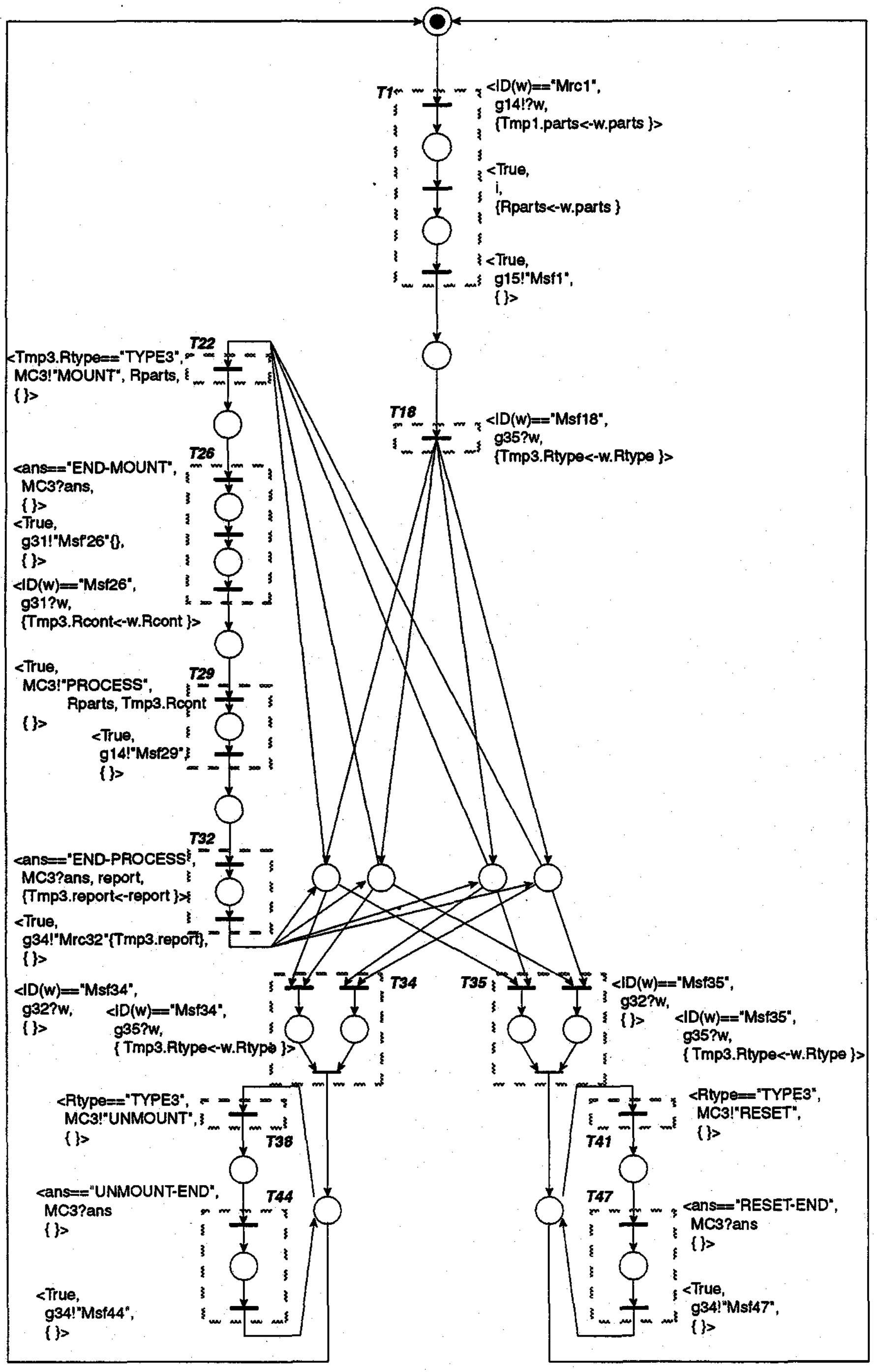
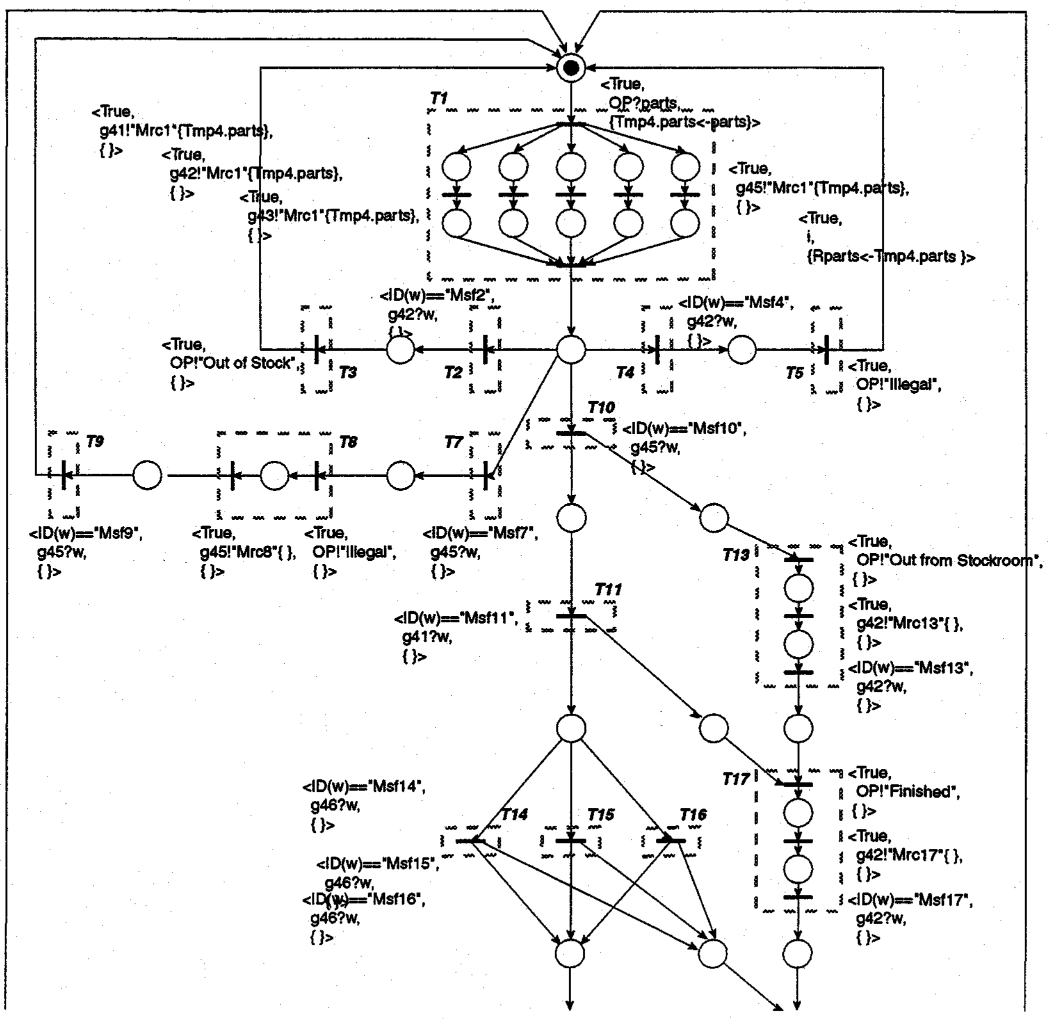
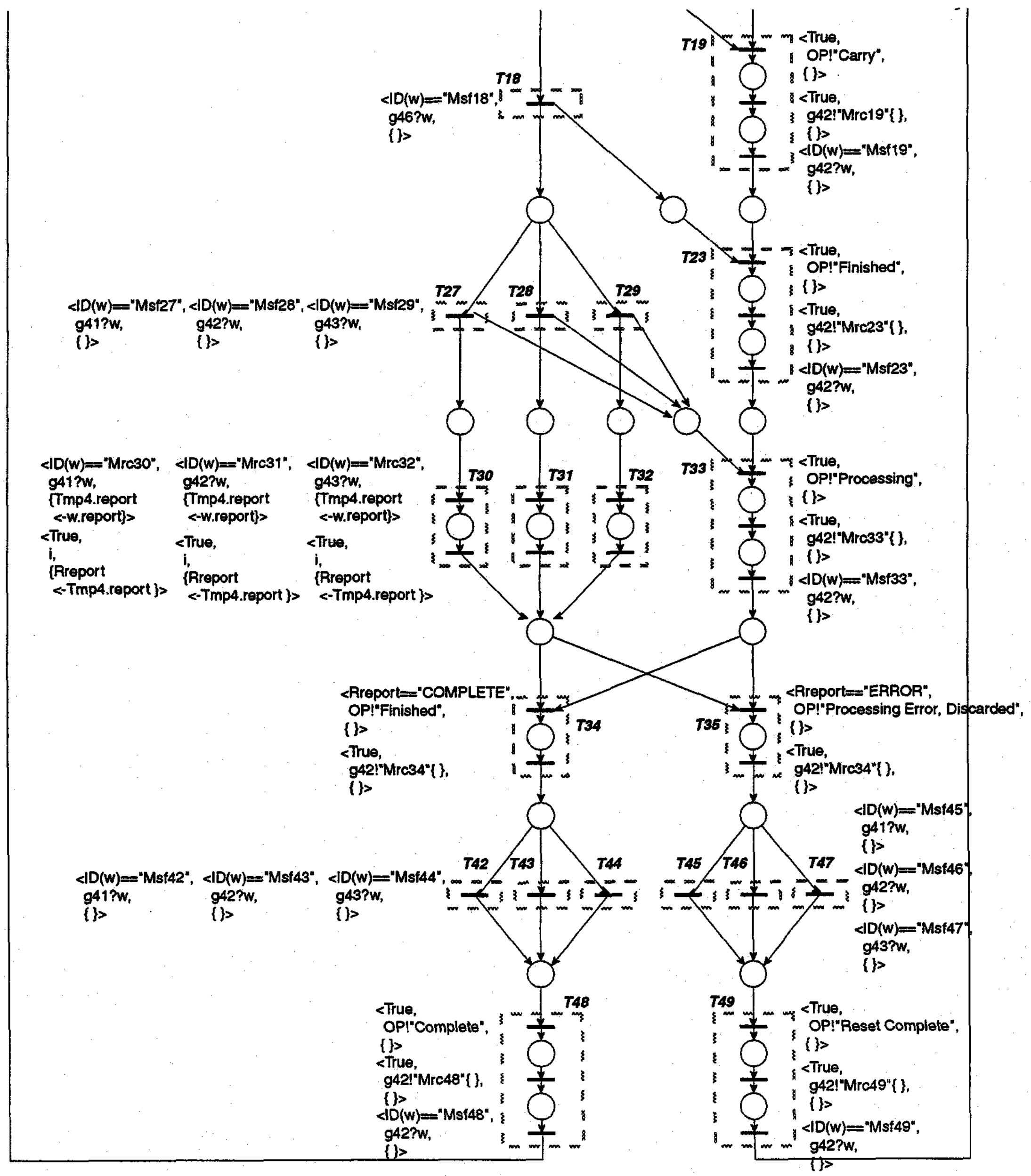


Figure 8: Specification for PE$_3$.



T1
<True, OP?parts, {Tmp4.parts<-parts}>
<True, g41!"Mrc1"{Tmp4.parts}, { }>
<True, g42!"Mrc1"{Tmp4.parts}, { }>
<True, g43!"Mrc1"{Tmp4.parts}, { }>
<True, g45!"Mrc1"{Tmp4.parts}, { }>
<True, i, {Rparts<-Tmp4.parts }>
<ID(w)=="Msf2", g42?w, { }>
<ID(w)=="Msf4", g42?w, { }>
<True, OP!"Out of Stock", { }>
T3
T2
T4
T5
<True, OP!"Illegal", { }>
T10
<ID(w)=="Msf10", g45?w, { }>
T9
T8
T7
<ID(w)=="Msf9", g45?w, { }>
<True, g45!"Mrc8"{ }, { }>
<True, OP!"Illegal", { }>
<ID(w)=="Msf7", g45?w, { }>
T13
<True, OP!"Out from Stockroom", { }>
<True, g42!"Mrc13"{ }, { }>
<ID(w)=="Msf13", g42?w, { }>
T11
<ID(w)=="Msf11", g41?w, { }>
T17
<True, OP!"Finished", { }>
<True, g42!"Mrc17"{ }, { }>
<ID(w)=="Msf17", g42?w, { }>
<ID(w)=="Msf14", g46?w, { }>
T14
T15
T16
<ID(w)=="Msf15", g46?w, { }>
<ID(w)=="Msf16", g46?w, { }>

Figure 9: Specification for PE_4. (1 of 2)



T18
<ID(w)=="Msf18", g46?w, { }>
T19
<True, OP!"Carry", { }>
<True, g42!"Mrc19"{ }, { }>
<ID(w)=="Msf19", g42?w, { }>
T23
<True, OP!"Finished", { }>
<True, g42!"Mrc23"{ }, { }>
<ID(w)=="Msf23", g42?w, { }>
T27
T28
T29
<ID(w)=="Msf27", g41?w, { }>
<ID(w)=="Msf28", g42?w, { }>
<ID(w)=="Msf29", g43?w, { }>
T30
T31
T32
T33
<ID(w)=="Mrc30", g41?w, {Tmp4.report <-w.report}>
<True, i, {Rreport <-Tmp4.report }>
<ID(w)=="Mrc31", g42?w, {Tmp4.report <-w.report}>
<True, i, {Rreport <-Tmp4.report }>
<ID(w)=="Mrc32", g43?w, {Tmp4.report <-w.report}>
<True, i, {Rreport <-Tmp4.report }>
<True, OP!"Processing", { }>
<True, g42!"Mrc33"{ }, { }>
<ID(w)=="Msf33", g42?w, { }>
T34
<Rreport=="COMPLETE", OP!"Finished", { }>
<True, g42!"Mrc34"{ }, { }>
T35
<Rreport=="ERROR", OP!"Processing Error, Discarded", { }>
<True, g42!"Mrc34"{ }, { }>
T42
T43
T44
<ID(w)=="Msf42", g41?w, { }>
<ID(w)=="Msf43", g42?w, { }>
<ID(w)=="Msf44", g43?w, { }>
T45
T46
T47
<ID(w)=="Msf45", g41?w, { }>
<ID(w)=="Msf46", g42?w, { }>
<ID(w)=="Msf47", g43?w, { }>
T48
<True, OP!"Complete", { }>
<True, g42!"Mrc48"{ }, { }>
<ID(w)=="Msf48", g42?w, { }>
T49
<True, OP!"Reset Complete", { }>
<True, g42!"Mrc49"{ }, { }>
<ID(w)=="Msf49", g42?w, { }>

Figure 10: Specification for PE$_4$. (2 of 2)

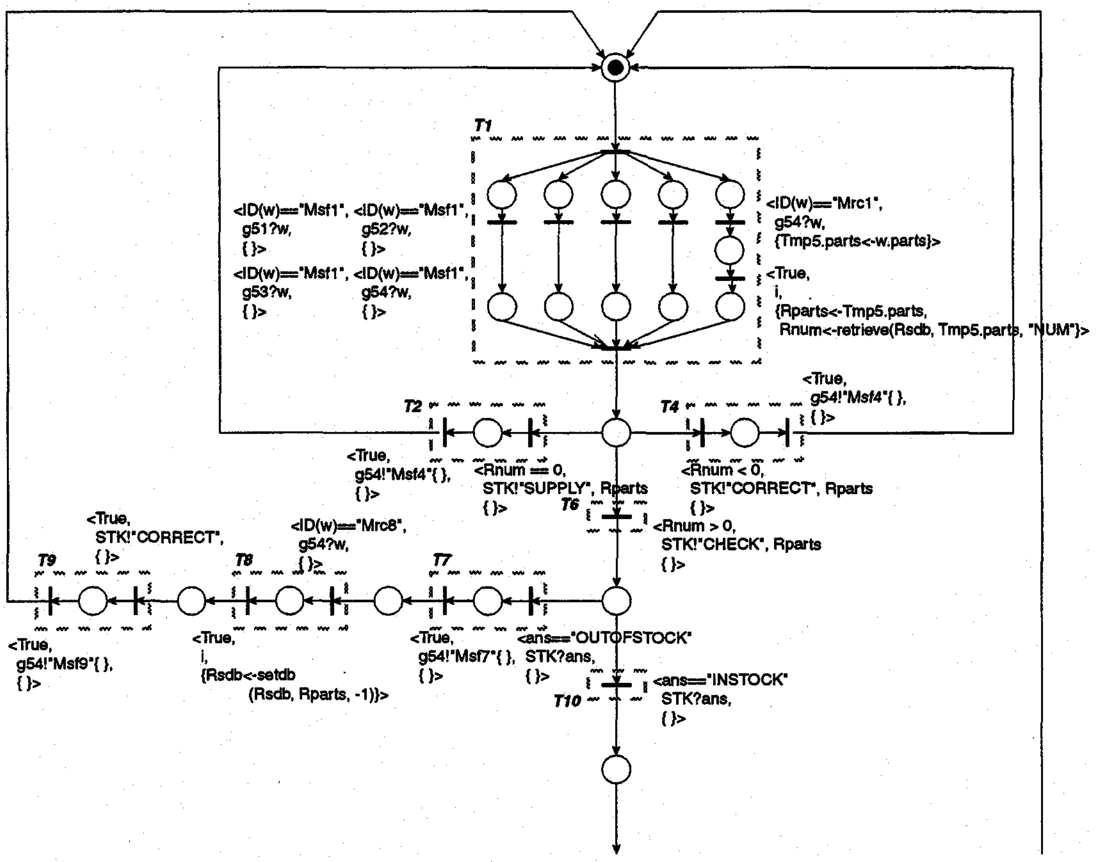

T1
<ID(w)=="Msf1", g51?w, { }>
<ID(w)=="Msf1", g52?w, { }>
<ID(w)=="Msf1", g53?w, { }>
<ID(w)=="Msf1", g54?w, { }>
<ID(w)=="Mrc1", g54?w, {Tmp5.parts<-w.parts}>
<True, i, {Rparts<-Tmp5.parts, Rnum<-retrieve(Rsdb, Tmp5.parts, "NUM"}>
T2
<True, g54!"Msf4"{ }, { }>
<Rnum == 0, STK!"SUPPLY", Rparts { }>
T4
<True, g54!"Msf4"{ }, { }>
<Rnum < 0, STK!"CORRECT", Rparts { }>
T6
<Rnum > 0, STK!"CHECK", Rparts { }>
T9
<True, STK!"CORRECT", { }>
<True, g54!"Msf9"{ }, { }>
T8
<ID(w)=="Mrc8", g54?w, { }>
<True, i, {Rsdb<-setdb (Rsdb, Rparts, -1)}>
T7
<True, g54!"Msf7"{ }, { }>
<ans=="OUTOFSTOCK" STK?ans, { }>
T10
<ans=="INSTOCK" STK?ans, { }>

Figure 11: Specification for PE$_5$. (1 of 2)

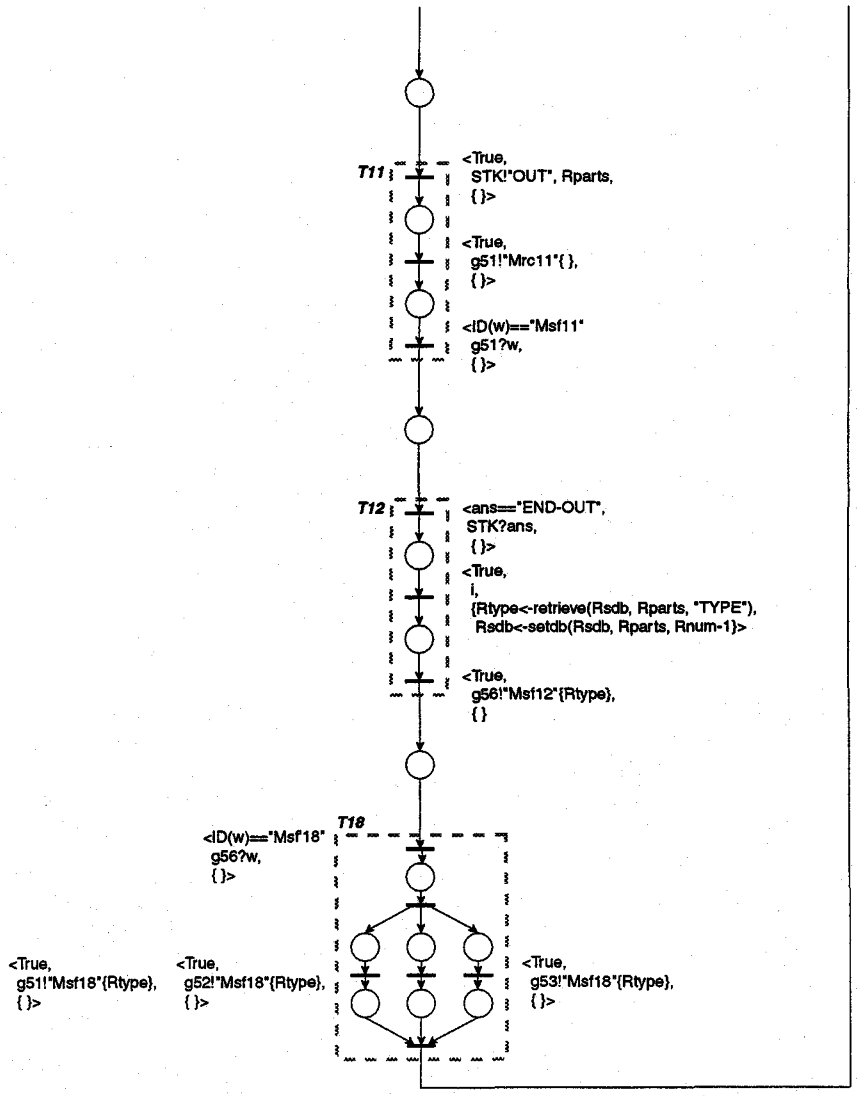


Figure 12: Specification for PE_5. (2 of 2)

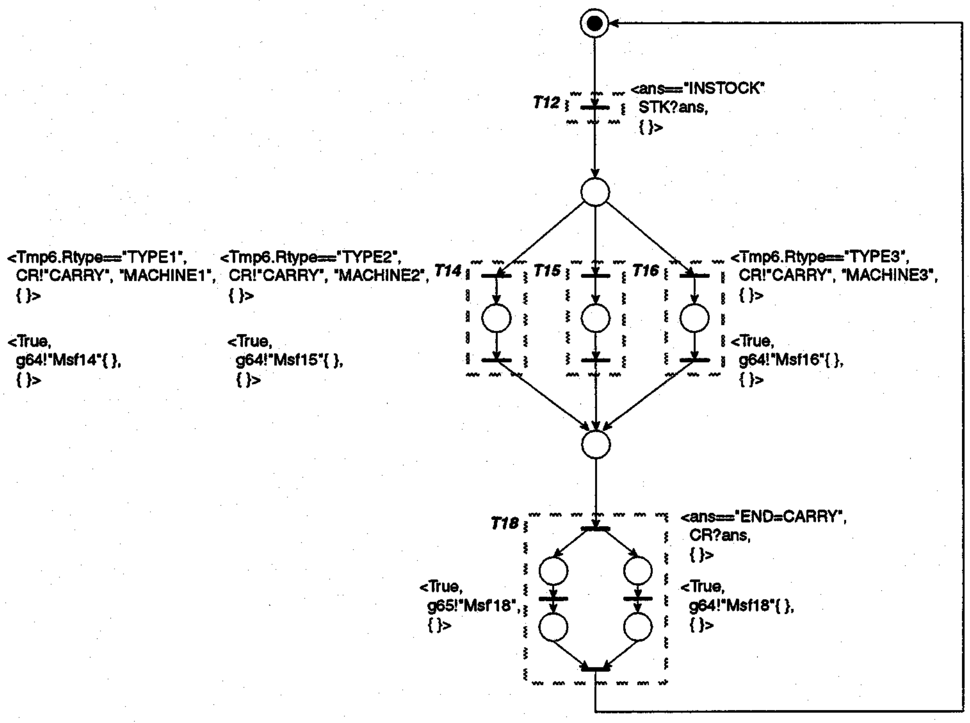


Figure 13: Specification for PE_6.

List of Figures

List of Tables