A Petri Net Based Method for Deriving Distributed Specification with Optimal Allocation of Resources

Hirozumi Yamaguchi
Graduate School of Eng. Sci.
Osaka University
Toyonaka, Osaka, 560-8531, Japan
h-yamagu@ics.es.osaka-u.ac.jp

Khaled El-Fakih
School of Info. Tech. and Eng.
University of Ottawa
Ottawa, Ontario, K1N 6N5, Canada
kelfakih@site.uottawa.ca

Gregor v. Bochmann
School of Info. Tech. and Eng.
University of Ottawa
Ottawa, Ontario, K1N 6N5, Canada
bochmann@site.uottawa.ca

Teruo Higashino
Graduate School of Eng. Sci.
Osaka University
Toyonaka, Osaka, 560-8531, Japan
higashino@ics.es.osaka-u.ac.jp

Abstract

In this paper, we present a method for the synthesis of extended Petri net based distributed specification. Although a lot of synthesis methods have been proposed, only a few synthesis methods have treated resources (computational data) such as databases and files. In contrast to previous methods that assume some fixed resource allocation, our method finds an optimal resource allocation that optimizes the derived distributed specification, based on some reasonable communication cost criteria. The method starts by identifying the set of rules for deriving a protocol specification from a given service specification. Based on these rules, an optimal resource allocation problem is formulated using an integer linear programming model. An example application is discussed.

1. Introduction

Synthesis methods have been used (for surveys see [2]) to derive a specification of a distributed system (hereafter called *protocol specification*) automatically from a given specification of the service to be provided by the distributed system to its users (called *service specification*). The service specification is written like a program of a centralized system, and does not contain any specification of the message exchange between different physical locations. However, the protocol specification contains the specification of communications between protocol entities (PE's) at the different locations.

A number of existing protocol synthesis strategies have been described in the literature. The first strategy, [3, 6], aims at implementing complex control-flows using several computational models such as LOTOS and Petri nets. The second strategy, [7], aims at satisfying the timing constraints specified by a given service specification in the derived protocol specification. This strategy deals with real-time distributed systems. The last strategy, [8, 4, 5], deals with the management of distributed resources such as files and databases. The objective here, is to determine how the values of these distributed resources are updated or exchanged between PE's for a given fixed resource allocation on different physical locations.

Some methods in the last strategy have tried to derive a protocol specification with minimum communication costs. Especially, the methods presented in our previous research work [4, 8], minimize the number of messages exchanged between PE's for a given fixed resource allocation. However, in the context of distributed applications, one also has to decide on an optimal allocation of these resources, since this allocation significantly affects the communication costs of the derived PE's.

As an example, we consider a Computer Supported Cooperative Work (CSCW) software development process. This process is distributed among engineers (developers, designers, managers and others). Each engineer has his own machine (PE) and participates in the development process using distributed resources (drafts, source codes, object codes, multimedia video and audio files, and others) placed on different machines. Considering the need for using these resources between different computers, we would like to derive, using a protocol synthesis method, all engineers' sub-processes (protocol specification) knowing the whole software development process (service specification)

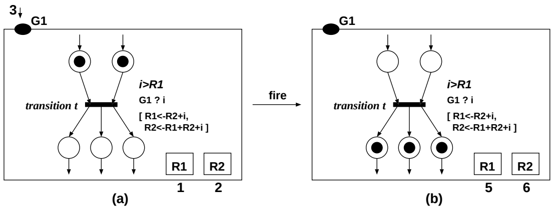


Figure 1. Register Values and Token Locations before and after Firing of Transition in PNR

and decide on an allocation of the resources that would minimize the communications costs.

In this paper, we propose a new method to derive a protocol specification with an optimal allocation of resources from a given service specification. Both service and protocol specifications are described using extended Petri nets. The method starts by identifying a set of rules for deriving a protocol specification. Based on these rules, an optimal resource allocation problem is formulated using an integer linear programming (ILP) model. This problem is about determining an optimal allocation of resources that minimizes the communication costs of the protocol specification. Our ILP model can also treat several reasonable cost criteria that could be used in various application areas for deriving protocol specifications. Particularly, we have considered the following cost criteria: (a) communication channel costs, (b) the size of messages to be exchanged between different PE's, (c) the number of messages based on frequency of executions, and (d) resource placement costs.

2. Service Specification and Protocol Specification

2.1. Petri Net Model with Registers

We use an extended Petri net model called a *Petri Net with Registers* (*PNR* in short) to describe both service and protocol specifications of a distributed system.

Each transition t in PNR has a label $\langle \mathcal{C}(t), \mathcal{E}(t), \mathcal{S}(t) \rangle$, where $\mathcal{C}(t)$ is a pre-condition statement (one of the firing conditions of t), $\mathcal{E}(t)$ is an event expression (which represents I/O) and $\mathcal{S}(t)$ is a set of substitution statements (which represents parallel updates of data values). Consider, for example, transition t of Fig. 1 where $\mathcal{C}(t)$ ="$i > R_1$", $\mathcal{E}(t)$ ="$G_1?i$" and $\mathcal{S}(t)$ ="$R_1 \leftarrow R_2 + i, R_2 \leftarrow R_1 + R_2 + i$". i is an input variable, which keeps an input value and its value is referred by only the transition t. R_1 and R_2 are registers, which keep assigned values until new values are assigned, and their values may be referred and updated by all the transitions in PNR (that is, global variables). G_1 is a gate, a service access point (interaction point) between users and the system. Note that "?" in $\mathcal{E}(t)$ means that $\mathcal{E}(t)$ is an input event.

A transition may fire if (a) each its input place has one token, (b) the value of $\mathcal{C}(t)$ is true and (c) an input value is given through the gate in $\mathcal{E}(t)$ (if $\mathcal{E}(t)$ is an input event). Assume that an integer of value *three* has been given through gate G_1, and the current values of registers R_1 and R_2 are 1 and 2, respectively. In this case the value of "$i > R_1$" is true and the transition may fire. If it fires, the event "$G_1?i$" is executed and the input value *three* is assigned to input variable i. Then "$R_1 \leftarrow R_2 + i$" and "$R_2 \leftarrow R_1 + R_2 + i$" are executed in parallel. Therefore after the firing, the tokens are moved and the values of registers R_1 and R_2 are changed to five ($= 2+3$) and six ($= 1+2+3$), respectively (Fig. 1(b)).

Formally, $\mathcal{E}(t)$ is one of the following three events: "$G_s\ !exp$", "$G_s\ ?iv$", or "τ". "$G_s\ !exp$" is an output event and it means that the value of expression "exp", whose arguments are registers, is output through gate G_s. "$G_s\ ?iv$" is an input event and it means that the value given through G_s is assigned to the input variable "iv". "τ" is an internal event, which is unobservable from the users. $\mathcal{S}(t)$ is a set of substitution statements, each of the form "$R_w \leftarrow exp_w$", where R_w is a register and exp_w is an expression whose arguments are from the input variable in $\mathcal{E}(t)$ and registers. If t fires, $\mathcal{E}(t)$ is executed followed by the parallel execution of statements in $\mathcal{S}(t)$.

2.2. Service Specification

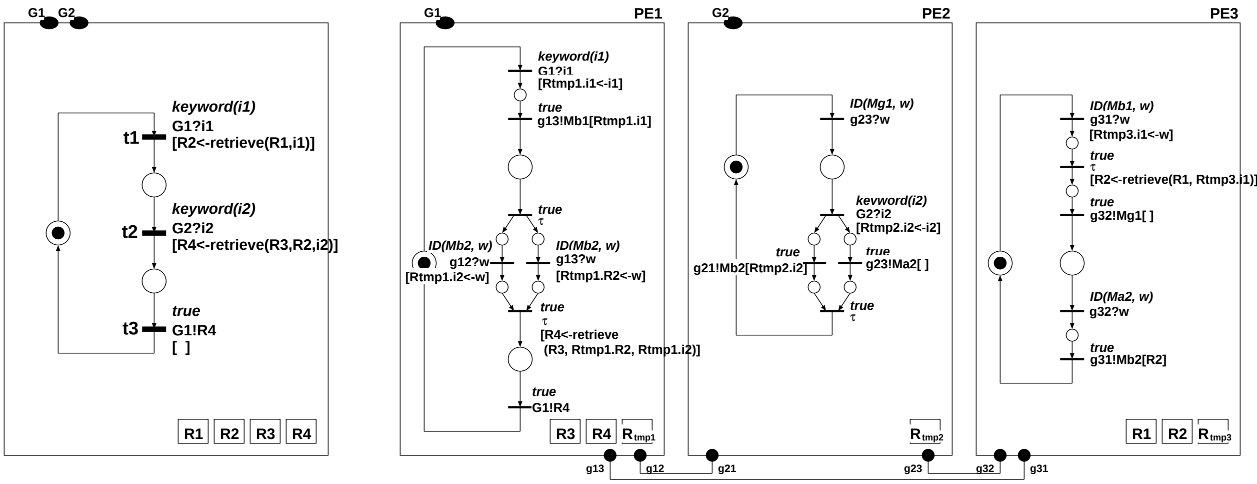
At a highly abstracted level, a distributed system is regarded as a centralized system which works and provides services as a single "virtual" machine. The number of actual PE's and communication channels among them are hidden. The specification of the distributed system at this level is called a *service specification* and denoted by $Sspec$.

Actual resources of a distributed system may be located on some physical machines, called protocol entities. However, only one virtual machine is assumed at this level.

Fig. 2(a) shows $Sspec$ of a simple database system which has only three transitions. The system receives a keyword (input variable i_1) through gate G_1, retrieves an entry corresponding to the keyword from a database (register R_1), and stores the result to register R_2. This is done on transition t_1. Then the system receives another keyword (input variable i_2) through gate G_2, retrieves an entry corresponding to the keyword and the retrieved entry (register R_2) from another database (register R_3), and stores the result to register R_4. This is done on transition t_2. Finally the system outputs the second result (the value of register R_4) through G_1 and returns to the initial state.

2.3. Protocol Specification

A distributed system is a communication system which consists of p protocol entities PE_1, PE_2, ... and PE_p. We



(a) Service Specification

(b) Protocol Specification

Figure 2. Service Specification and Protocol Specification

assume a duplex and reliable communication channel with infinite capacity buffers at both ends, between any pair of PE_i and PE_j. The PE_i (PE_j) side of the communication channel is represented as gate g_{ij} (g_{ji}). Moreover, we assume that some resources (registers and gates) are allocated to certain PE's of the distributed system.

Two PE's communicate with each other by exchanging messages. If PE_i executes an output event "$g_{ij}!M[R_w]$", the value of register R_w located on PE_i is sent to PE_j through the communication channel between them and put into the buffer at PE_j's end. M is an identifier to distinguish several values which may exist at the same time on the same channel. PE_j can take the value identified by M from the buffer, by executing an input event "$g_{ji}?w$" with a pre-condition $ID(M,w)$. $ID(M,w)$ is a predicate whose value is true *iff* the identifier in input variable w is M. Note that more than one register's or input variable's value can be sent at a time. If a received data contains multiple values, they are distinguished by suffix such as $w.R_1$ and $w.i$. A set of an identifier and register/input values is called a message. A message may contain no value and sending such a message is represented as an output event "$g_{ij}!M[\,]$".

In order to implement a distributed system which consists of p PE's, we must specify the behavior of these PE's. A specification of PE_k is called a *protocol entity specification* and denoted by $Pspec_k$. A set of p protocol entity specifications $\langle$ $Pspec_1$, ..., $Pspec_p$ $\rangle$ is called a *protocol specification* and denoted by $Pspec^{\langle 1,p\rangle}$. We need a protocol specification to implement the distributed system.

As an example, let us assume that there are three PE's PE_1, PE_2 and PE_3 in order to implement the service specification of Fig. 2(a). We also assume that an allocation of resources to these PE's has been fixed as follows. PE_1 has the gate G_1 and the registers R_3 and R_4, PE_2 has the gate G_2, and PE_3 has the registers R_1 and R_2. Note that in addition to these registers, we assume that each PE_i has another register $Rtmp_i$ to keep received values given through gates (inputs and message contents)[1]. Fig. 2(b) shows an example of $Pspec^{\langle 1,3\rangle}$, which provides the service of Fig. 2(a), based on this allocation of resources.

According to the specification of Fig. 2(b), PE_1 first receives an input (input variable i_1) through G_1 and stores it to $Rtmp_1.i_1$. Then it sends the value of $Rtmp_1.i_1$ to PE_3 as a message, since PE_3 needs the value of i_1 to change the value of R_2. PE_3 receives and stores the value to $Rtmp_3.i_1$. Then it changes the value of R_2 using its own value and the value of $Rtmp_3.i_1$, and sends a message to PE_2. When PE_2 receives the message, PE_2 knows that it can now check the value of $\mathcal{C}(t_2)$ and execute $\mathcal{E}(t_2)$. PE_2 receives an input (input variable i_2), stores it to $Rtmp_2.i_2$, and sends two messages. One is to send the value of i_2 to PE_1 and another is to incite PE_3 to send the value of R_2 to PE_1. PE_1 receives these values and stores them to $Rtmp_1.i_2$ and $Rtmp_1.R_2$, respectively. Then it changes the value of R_4. Finally, PE_1 outputs the value of R_4 and PE_1, PE_2 and PE_3 return to their initial states.

3. Protocol Derivation

A method for deriving a protocol specification from a given service specification is described in this section. It is based on the simulation of each transition $t_x = \langle \mathcal{C}(t_x), \mathcal{E}(t_x), \mathcal{S}(t_x)\rangle$ of the service specification by corre-

[1] $Rtmp_i$ can contain several values. The values can be distinguished by adding the name of the value as suffix, such as $Rtmp_1.R_3$. Here, we can realize such a register that contains several values, by using several registers. However, for simplicity of discussion, we use these registers.

sponding PE's in the protocol specification. The principle of the method introduced in this paper is as follows.

- The PE that has gate G_s used in $\mathcal{E}(t_x)$ (say PEstart(t_x)) checks the value of $\mathcal{C}(t_x)$ (pre-condition statement) and executes $\mathcal{E}(t_x)$ (event expression).
- After that, the PE sends messages called α-messages to the PE's which have the registers used in the arguments of $\mathcal{S}(t_x)$ (substitution statements).
- In response, these PE's send the register values to the PE's which have the registers to be updated in $\mathcal{S}(t_x)$ (PEsubst(t_x) denotes the set of those PE's) as messages called β-messages.
- The substitution statements are executed and notification messages called γ-messages are sent to those PE's which will start the execution of the next transitions.

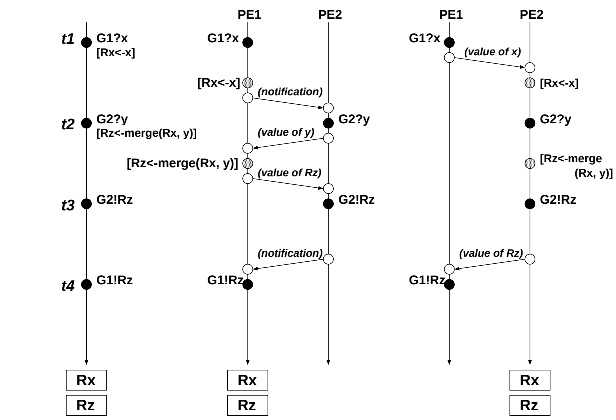
In Fig. 3, we present the details of our derivation method as a set of rules which specify how PE's execute each transition t_x of $Sspec$. These rules are further classified into action and message rules. Action rules specify which PE checks the pre-condition and executes the event and substitution statements of t_x. Message rules specify how the PE's exchange messages, and the contents and types of these messages.

Three types of messages are exchanged for the execution of t_x. (1) α-messages are sent by the PE that starts the execution of t_x (*i.e.* PEstart(t_x)) to inform those PE's who need to send their registers' values to other PE's, that they can go ahead and send these values. Thus, an α-message does not contain values of registers. (2) β-messages are sent in order to let each PE which executes some substitution statements of t_x (*i.e.* $PE_k \in$PEsubst(t_x)), know the timing and some values of registers' it needs for executing these statements. (3) γ-messages are sent to each $PE_m \in$PEstart($t_x \bullet \bullet$), note that $t_x \bullet \bullet$ is the set of each next transition of t_x, to let it know the timing and some values of registers it needs to start executing the next transitions (*i.e.* transitions in $t_x \bullet \bullet$).

4. Optimal Resource Allocation

4.1. On Optimal Resource Allocation

In our previous work[8], we have shown that the number of messages exchanged between different PE's for the execution of a transition in $Sspec$ may not be unique even for a given fixed allocation of resources. This is due to the fact that a resource may be allocated to more than one PE and several resource values may be sent in one message. However, due to the limitations of our assumption that the resources are fixed, more messages have to be exchanged between the derived PE's, if these resources are not already optimally allocated between different PE's.



(a) Service Specification (b) Protocol Specification (c) Protocol Specification with Optimal Allocation of Resources

Figure 4. Optimal Allocation

For illustration, we give a simple example which describes a process of merging two codes (input variables x and y) into one code (register R_z) developed by two engineers (assigned to gates G_1 and G_2) on different sites, PE_1 and PE_2. Fig. 4(a) shows the service specification, and its corresponding protocol specifications with different resource allocations are given in Fig. 4(b) and in Fig. 4(c). The protocol specification of Fig. 4(b) needs four messages, while that of Fig. 4(c) needs only two.

In most realistic applications, one may want to consider some other communication cost criteria along with the number of messages exchanged during protocol derivation. If we consider, for example, the different costs of placing resources on different PE's, then deciding on an optimal allocation of these resources would significantly affect the communication costs of a derived specification.

We consider communication costs as an important element in the development of distributed applications. To reduce these costs, an optimal allocation of resources that minimizes them has to be determined. In the following subsection, we build a model that decides on an optimal allocation that minimizes the number of messages exchanged between different PE's, then later in Section 5 we incorporate into this model some other cost criteria that we consider important for deriving distributed specifications with minimum communication costs.

4.2. Integer Linear Programming Model for Protocol Derivation with Minimum Communication Costs

We introduce the following 0-1 variables in order to determine, using our derivation method, an optimal resource allocation that minimizes the number of messages exchanged between different PE's.

We let $t_x = \langle \mathcal{C}(t_x), \mathcal{E}(t_x), \mathcal{S}(t_x) \rangle$ be a transition of $Sspec$.

[Action Rules]

(A_1) The PE which has the gate appearing in $\mathcal{E}(t_x)$ (denoted by G_s) checks that

(a) the value of $\mathcal{C}(t_x)$ is true,

(b) the execution of the previous transitions of t_x has been finished and

(c) an input has been given through G_s if $\mathcal{E}(t_x)$ is an input event.

Then the PE executes $\mathcal{E}(t_x)$. This PE is denoted by PEstart(t_x).

(A_2) After (A_1), the PE's which have at least one register whose value is changed in the substitution statements $\mathcal{S}(t_x)$ execute the corresponding statements in $\mathcal{S}(t_x)$. The set of these PE's is denoted by PEsubst(t_x).

[Message Rules]

($M_{\beta 1}$) Each $PE_k \in$PEsubst(t_x) must receive at least one β-message from some PE's (each called PE_j) in order to know the timing and values of registers it needs for executing its substitution statements (see ($M_{\beta 2}$)), except where PE_k=PEstart(t_x), in this case PE_k already knows the timing to start executing its substitution statements of t_x.

($M_{\beta 2}$) If $PE_k \in$PEsubst(t_x) needs the value of some register (say R_z) in order to execute its substitution statements, then PE_k must receive R_z through a β-message if R_z is not in PE_k.

($M_{\beta 3}$) Each PE_j that sends some values of registers to $PE_k \in$PEsubst(t_x) through a β-message, knows the timing to send these values by receiving an α-message from PEstart(t_x). Note, if PE_j=PEstart(t_x) then PE_j knows the timing to send these values without receiving an α-message.

(M_{α}) After (A_1), the only PE that can send α-messages to the PE's which need them is PEstart(t_x).

($M_{\gamma 1}$) Each $PE_m \in$PEstart($t_x \bullet\bullet$), where $t_x \bullet\bullet$ is the set of next transitions of t_x, must receive a γ-message from each $PE_k \in$PEsubst(t_x) after (A_2), except where $m = k$. This allows PE_m to know that the execution of the substitution statements of t_x had been finished.

($M_{\gamma 2}$) Each $PE_m \in$PEstart($t_x \bullet\bullet$) must receive at least one γ-message from some PE_l (where $m \neq l$) in order to know that the execution of t_x had been finished and/or to know some values of registers it needs to evaluate and execute its condition and event expression, respectively.

($M_{\gamma 3}$) Each PE_l that sends a γ-message to $PE_m \in$PEstart($t_x \bullet\bullet$) :

(a) must be in PEsubst(t_x) (see ($M_{\gamma 1}$)), or

(b) must receive an α-message from PEstart(t_x) to know the timing to send the γ-message to PE_m, or

(c) it is itself PEstart(t_x). In this case, PE_l sends the γ-message to let PE_m know the timing and/or some values of registers to start evaluating and executing its condition and event expressions.

($M_{\gamma 4}$) If $PE_m \in$PEstart($t_x \bullet\bullet$) needs the value of some register (say R_v) in order to evaluate and/or execute its substitution statements, then PE_m must receive R_v through a γ-message if R_z is not in PE_m.

Figure 3. Derivation Method in Detail

- Each of the following variables represent the fact that a message is sent from one PE to another.
  - $\alpha^x_{p,q}$ ($\beta^x_{p,q}$, $\gamma^x_{p,q}$): its value is one *iff* an α-message (β-, γ-) is sent from PE_p to PE_q in the execution of transition t_x; otherwise zero.
  - $\beta^x_{p,q}[R_w]$ ($\gamma^x_{p,q}[R_w]$): its value is one *iff* the β- (γ-) message sent from PE_p to PE_q contains the value of register R_w; otherwise zero.
- $ALC_p[G_s]$ ($ALC_p[R_w]$) : its value is one *iff* gate G_s (register R_w) is allocated to PE_p; otherwise zero.
- $PEstart^x_p$: its value is one *iff* PE_p starts the execution of t_x; otherwise zero.
- $PEsubst^x_p$: its value is one *iff* PE_p executes one or more substitution statements of t_x; otherwise zero.

Using the above variables, we determine an optimal resource allocation that minimizes the number of messages exchanged between different PE's by minimizing the following objective function, subject to constraints (1) to (16) described below.

Objective Function:

$$\text{Min}: \sum_x \sum_p \sum_q \left(\alpha^x_{p,q} + \beta^x_{p,q} + \gamma^x_{p,q} \right)$$

The following constraints are derived from the definition of their variables. According to Constraint (1), if a β-message is sent from PE_j to PE_k in the execution of t_x and it contains the value R_w, then this message should have been sent through a β-message. Moreover, in order for PE_j to send R_w, R_w should be allocated to it. The same reasoning applies to Constraint (2).

$$\beta^x_{j,k} + ALC_j[R_w] - 2\beta^x_{j,k}[R_w] \geq 0 \qquad (1)$$

$$\gamma^x_{l,m} + ALC_m[R_w] - 2\gamma^x_{l,m}[R_w] \geq 0 \quad (2)$$

According to rule (A_1), the PE that has the gate G_s appearing in event expression $\mathcal{E}(t_x)$ (say G_s) must be the one that executes this expression (*i.e.* PEstart(t_x)).

$$PEstart^x_i - ALC_i[G_s] = 0 \quad (3)$$

According to rule (A_2), each PE that has a register R_w whose value is changed in the set of substitution statements S(t_x), must be the one that executes this substitution statement.

$$PEsubst^x_k - ALC_k[R_w] \geq 0 \quad (4)$$

$$\sum_w ALC_k[R_w] - PEsubst^x_k \geq 0 \quad (5)$$

Constraints (6) to (13) directly correspond to message exchange rules ($M_{\beta 1}$) to ($M_{\gamma 4}$) of Fig. 3.

The following Constraint corresponds to rule ($M_{\beta 1}$). It means that at least one β-message should be sent to PE_k or PE_k =PEstart(t_x), if $PE_k \in$PEsubst(t_x).

$$\sum_j \beta^x_{j,k} + PEstart^x_k - PEsubst^x_k \geq 0 \quad (6)$$

Constraint (7) corresponds to rule ($M_{\beta 2}$).

$$\sum_j \beta^x_{j,k}[R_z] + ALC_k[R_z] - ALC_k[R_w] \geq 0 \quad (7)$$

Constraint (8) corresponds to rule ($M_{\beta 3}$).

$$PEstart^x_j + \sum_i \alpha^x_{i,j} - \beta^x_{j,k} \geq 0 \quad (8)$$

Constraint (9) corresponds to rule (M_α).

$$PEstart^x_i - \alpha^x_{i,j} \geq 0 \quad (9)$$

Constraint (10) corresponds to rule ($M_{\gamma 1}$).

$$\gamma^x_{k,m} - PEsubst^x_k - PEstart^x_m \geq -1 \quad (10)$$

Constraint (11) corresponds to rule ($M_{\gamma 2}$).

$$\sum_l \gamma^x_{l,m} + PEstart^x_m + PEsubst^x_m - PEstart^y_m \geq 0 \quad (11)$$

Constraint (12) corresponds to rule ($M_{\gamma 3}$).

$$PEstart^x_l + \sum_i \alpha^x_{i,l} + PEsubst^x_l - \gamma^x_{l,m} \geq 0 \quad (12)$$

Constraint (13) corresponds to rule ($M_{\gamma 4}$).

$$\sum_l \gamma^x_{l,m}[R_v] + ALC_m[R_v] \geq 1 \quad (13)$$

Constraints (14) and (15) restrict the possible number of PE's which have a gate G_s and a register R_w, respectively. Note, as described in Constraint (16), we use a register (called $Rtmp_i$) in $PEstart^x_i$ to save the input variable used in the event expression of t_x (say i^x).

$$\sum_i ALC_i[G_s] = 1 \quad (14)$$

$$\sum_i ALC_i[R_w] \geq 1 \quad (15)$$

$$ALC_i[Rtmp_i.i^x] = PEstart^x_i \quad (16)$$

5. Other Cost Criteria

In this section, we present and incorporate into our ILP model some cost criteria that could be used in minimizing the communication costs of the derived protocol specification. One may select or combine these criteria according to the application area and its underlying network architecture.

Considering Communication Channels Costs For application areas that use communication channels with different channel costs, we let $ChannelCost_{p,q}$ denote the cost to send a message from PE_p to PE_q. Then we incorporate these costs into our ILP model as follows:

$$\text{Min}: \sum_x \sum_p \sum_q ChannelCost_{p,q} * \big(\alpha^x_{p,q} + \beta^x_{p,q} + \gamma^x_{p,q}\big)$$

Considering Size of Messages In most application areas, the size of resources exchanged between different PE's plays an important factor in determining their communication costs. We let $Size[R_w]$ denote the size of resource R_w, and reformulate our ILP model objective function as shown below.

$$\text{Min}: \sum_x \sum_p \sum_q \Big(\alpha^x_{p,q} + \beta^x_{p,q} + \gamma^x_{p,q} + \sum_w Size[R_w] * \big(\beta^x_{p,q}[R_w] + \gamma^x_{p,q}[R_w]\big) \Big)$$

Note that we consider the size of messages that do not contain values of registers relatively small (*i.e.* equal to one).

Considering Execution Frequencies of Transitions In some application areas, the structure of the service specification includes many loops and each loop includes many transitions. Consequently, in such areas, one might want to consider the frequencies of transitions execution during the protocol derivation. In general, this is a dynamic property of the system, however, an approximation of the firing

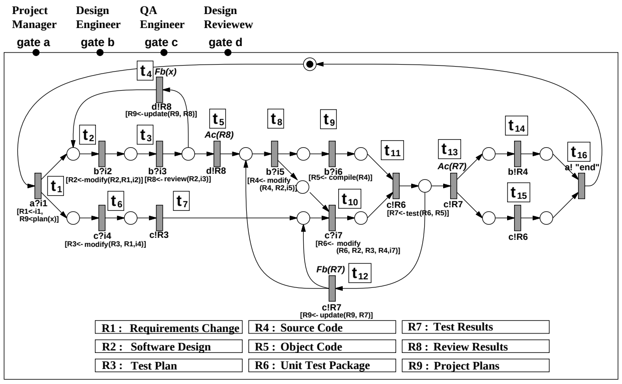


Figure 5. A Workflow of Software Development

frequency may be derived by firing vector analysis or simulation of Petri nets [1], which has been investigated extensively.

Let F^x denote the (approximate) firing frequency of a transition t_x. We incorporate F^x into our ILP model as shown below.

$$\text{Min}: \sum_x F^x * \sum_p \sum_q \left(\alpha^x_{p,q} + \beta^x_{p,q} + \gamma^x_{p,q}\right)$$

Considering Resource Placement Costs In application areas where there are major differences in the costs of placing resources on different physical locations (PE's), one might want to consider these differences during protocol derivation. We let $PlaceCost_p[R_w]$ denote the cost of placing resource R_w on PE$_p$, and we formulate our ILP model objective function as follows:

$$\begin{aligned}\text{Min}: & \sum_x \sum_p \sum_q \left(\alpha^x_{p,q} + \beta^x_{p,q} + \gamma^x_{p,q}\right) \\ & + \sum_p \sum_w PlaceCost_p[R_w] * ALC_p[R_w]\end{aligned}$$

6. Application and Experimental Results

Protocol synthesis methods have been applied to many applications such as communication protocols, factory manufacturing systems , distributed cooperative work management and so on. In this section, we apply our derivation method to the distributed development of software which involves four engineers (project manager, design engineer, quality assurance engineer, and design reviewer), in three different connected development sites (site$_1$, site$_2$, and site$_3$, respectively). The software development process includes modification and compilation of source code, test of

	site$_1$	site$_2$	site$_3$
site$_1$		1	10
site$_2$	1		5
site$_3$	10	5	

Table 2. Channel Costs

R_1	R_2	R_3	R_4	R_5	R_6	R_7	R_8	R_9
10	50	20	100	200	30	10	5	5

Table 3. Sizes of Resources

F^1	F^2	F^3	F^4	F^5	F^6	F^7	F^8
1	4	4	3	1	1	1	10
F^9	F^{10}	F^{11}	F^{12}	F^{13}	F^{14}	F^{15}	F^{16}
10	10	10	9	1	1	1	1

Table 4. Firing Frequencies of Transitions

Resource / Engineer	site$_1$	site$_2$	site$_3$
R_1	1	10	10
R_2	1	1	3
R_3	10	8	1
R_4	7	1	1
R_5	7	14	1
R_6	15	10	2
R_7	1	1	1
R_8	1	20	20
R_9	1	10	10
G_a	20	10	3
G_b	4	8	9
G_c	1	5	8
G_d	1	5	5

Table 5. Resource Placement Costs

the generated object code, and its review. The engineers cooperate with each other to finish these sub-sequential tasks.

Fig. 5 shows a workflow model of the above development process using PNR, where the engineers and the resources needed to accomplish the tasks are indicated. The reader may refer to [9] for a detailed description of the modeling concept.

We regard this workflow as a service specification and we derive the corresponding protocol specifications with minimum communication costs using the different cost criteria presented in the previous section. The specification for each PE in the derived protocol specification will correspond to the workflow in one site. Of course each engineer could be assigned only to one site.

We have developed an automated system to generate the ILP model and its constraints from the given specification in order to decide on an optimal resource allocation that min-

	$site_1$	$site_2$	$site_3$	Time (second)
(a)	G_d	G_a	G_b, G_c	157
		R_1	R_2,R_3,R_4,R_5,R_6,R_7,R_8,R_9	
(b)	G_b, G_c	G_a	G_d	359
	R_2,R_3,R_4,R_5,R_6,R_7	R_1,R_8,R_9		
(c)	G_a	G_d	G_b,G_c	28
	R_8	R_1	R_2,R_3,R_4,R_5,R_6,R_7,R_9	
(d)	G_d	G_a	G_b,G_c	12
	R_3	R_1	R_2,R_4,R_5,R_6,R_7,R_8,R_9	
(e)	G_d	G_a	G_b,G_c	30
	R_3	R_1	R_2,R_4,R_5,R_6,R_7,R_8,R_9	

Table 1. Optimal Resource Allocation and Derivation Time Using (a) the Number of Messages Costs, (b) the Channel Costs, (c) the Size of Message Costs, (d) the Execution Frequencies of Transitions Costs and (e) the Resource Placement Costs

imizes the communication costs of the derived PE's. Then we have used the program "lp_solve" on a Compaq XP1000 with Alpha 21264, to solve the optimization problem for the different cost criteria discussed above.

Table 1 contains the optimal resource allocation of the given specification and the time to decide them. The optimized costs are (a) the number of messages as in our ILP model of Section 4.2, (b) the channel costs depicted in Table 2, (c) the sizes of resources depicted in Table 3, (d) the execution frequencies of transitions depicted in Table 4 and (e) the resource placement costs depicted in Table 5. These experimental results show that our method can decide optimal resource allocations for various cost criteria in reasonable time.

7. Conclusion

In this paper, we have proposed a Petri net based method for deriving a protocol specification (distributed specification) from a given service specification, with an optimal allocation of resources that minimizes communication costs. The resource allocation problem is formulated using an integer linear programming model that can also use several reasonable cost criteria for deriving protocol specifications. We have also given an example application.

Our future work is to develop a distributed environment supporting our derivation method.

References

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

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

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

[4] H. Yamaguchi, K. Okano, T. Higashino and K. Taniguchi, "Synthesis of Protocol Entities' Specifications from Service Specifications in a Perti Net Model with Registers," *Proc. ICDCS-15*, pp. 510–517, 1995.

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

[6] A. Al-Dallal and K. Saleh, "Protocol Synthesis Using the Petri Net Model," *Prof. PDCS'97*, 1997.

[7] M. Kapus-Koler, "Deriving Protocol Specifications from Service Specifications with Heterogeneous Timing Requirements," *Proc. 1991 Int. Conf. on Soft. Eng. for Real Time Systems*, pp. 266–270, 1991.

[8] K. El-Fakih, H. Yamaguchi and G.v. Bochmann, "A Method and a Genetic Algorithm for Deriving Protocols for Distributed Applications with Minimum Communication Cost," *Proc. PDCS'99*, 1999.

[9] Kellner, M. et al. : "ISPW-6 Software Process Example," Proc. 1st Int. Conf. on Software Process, pp. 176–186, 1991.