Automatic Derivation of Petri Net Based Distributed Specification with Optimal Allocation of Resources

Khaled El-Fakih[†] Hirozumi Yamaguchi[‡] Gregor v. Bochmann[†] Teruo Higashino[‡]

[†]University of Ottawa
School of Information Technology and Engineering
Ottawa, Ontario K1N 6N5, CANADA
{kelfakih, bochmann}@site.uottawa.ca

[‡]Osaka University
Graduate School of Engineering Science
Toyonaka, Osaka 560-8531, JAPAN
{h-yamagu, 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. Our method finds an optimal allocation of resources (computational data) that optimizes the derived distributed specification, based on some reasonable communication cost criteria.

1 Introduction

Synthesis methods[1] have been used to derive a specification of a distributed system (*protocol specification*) automatically from a given specification of the service to be provided by the distributed system to its users (*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.

Some methods have tried to derive a protocol specification with minimum communication costs. Especially, the method in our previous research work [3] minimizes 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 the allocation significantly affects the communication costs of the derived PE's.

In this paper, we propose a new method to derive a protocol specification with an optimal allocation of resources from a given service specification. 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.

This work was partially funded by Communications and Information Technology Ontario (CITO).

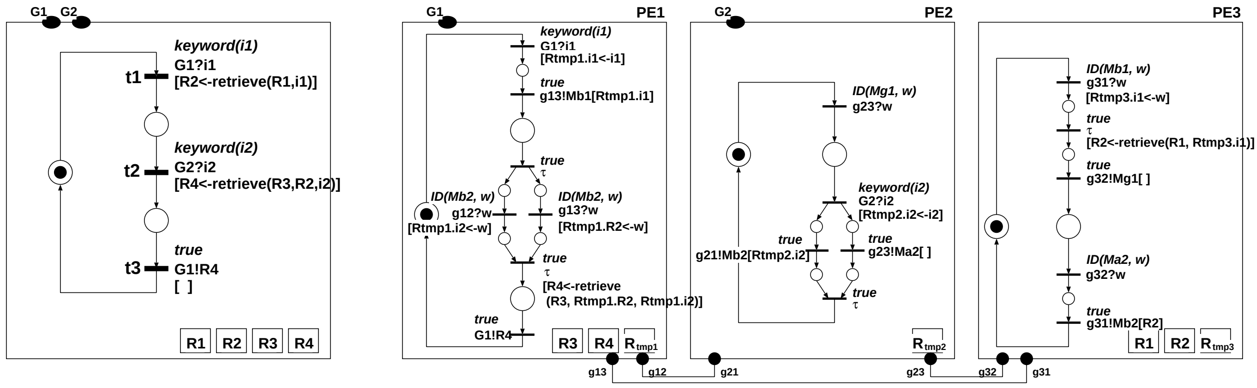
2 Service and Protocol Specifications

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 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). If t fires, $\mathcal{E}(t)$ is executed followed by the parallel execution of statements in $\mathcal{S}(t)$.

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 (PE's). However, only one virtual machine is assumed at this level.

Fig. 1(a) shows $Sspec$ of a simple database system



(a) Service Specification

(b) Protocol Specification

Figure 1. Service Specification and Protocol Specification

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 (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 (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.

Protocol Specification A distributed system is a communication system which consists of p protocol entities PE$_1$, PE$_2$, ... and PE$_p$. We 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 the resources (registers and gates) are allocated to certain PE's of the distributed system[1]. In order to implement the distributed system, 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.

Fig. 1(b) shows an example of $Pspec^{\langle 1,3\rangle}$, which provides the service of Fig. 1(a), based on this allocation of resources: 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. According to the specification of Fig. 1(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[2], 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

Our method for deriving protocol specification from a given service specification 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 corresponding PE's in the protocol specification.

The principle of the method is as follows. After the execution of all the previous transitions of t_x, the PE which has the gate in the event expression $\mathcal{E}(t_x)$ (say PEstart(t_x)) checks the value of the pre-condition statement $\mathcal{C}(t_x)$ and executes $\mathcal{E}(t_x)$. Then each PE which has at least one register whose value is changed in the substitution statements $\mathcal{S}(t_x)$ (say PE$_k$) changes the values of these registers. The values necessary for the change are sent from the PE's which have them. These PE's receive notification messages from PEstart(t_x) and send their values to PE$_k$. Using these values, PE$_k$ then can change the values of its register(s). After that, PE$_k$ sends notification messages to the PE's (called PEstart($t_x \bullet\bullet$)) which have the gates specified in $\mathcal{E}(t_x \bullet\bullet)$, where $t_x \bullet\bullet$ is the set of each next transitions of t_x, in order to indicate that the execution of t_x is completed.

[1]We assume that each PE$_i$ has another register $Rtmp_i$ to keep received values given through gates (inputs and message contents). $Rtmp_i$ can contain several values. The values can be distinguished by adding the name of the value as suffix, such as $Rtmp_1.R_3$.

[2]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$_j$ and put into the buffer at PE$_j$'s end. M is an identifier to distinguish several values 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 $ID(M, w)$. The value of $ID(M, w)$ is true *iff* the identifier in input variable w is M.

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$_1$) PE$_u$ 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 $\mathcal{E}(t_x)$. PE$_u$ 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 set of 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$) in order to know the timing and values of registers (see (M$_{\beta 2}$)) it needs for executing its substitution statements, except where PE$_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$_{\alpha 1}$) After (A$_1$), the only PE that can send α-message to the PE's which need it 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 β-message if R_z is not in PE$_m$.

Figure 2. *Derivation Method in Detail*

In Fig. 2, 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. Three types of messages are exchanged for the execution of t_x. α-messages are sent by the PE that starts the execution of t_x (*i.e.* PE$_u$=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. β-messages are sent in order to let each PE (say PE$_k$) which executes some substitution statements of t_x, (*i.e.* PE$_k$$\in$PEsubst($t_x$)) (i) know the timing and some values of registers' it needs for executing these statements and (ii) inform each PE that belongs to the set of next transitions of t_x that the execution of its substitution statements has been finished. γ-messages are sent to each PE$_m$$\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 its corresponding transition (*i.e.* start evaluating and executing its condition and event expressions).

4 Optimal Resource Allocation

In this section, we build an Integer Linear Programming (ILP) model that decides on an optimal allocation that minimizes the number of messages exchanged between different PE's, then we incorporate into this model some other cost criteria that we consider important for deriving distributed specifications with minimum communication costs.

Integer Linear Programming Model for Protocol Derivation with Minimum Communication Costs We introduce the following 0-1 variables.

- $\alpha^x_{u,q}$: its value is one *iff* an α-message is sent from PE$_u$=PEstart(t_x) to PE$_q$ in the execution of t_x; Otherwise zero.
- $\beta^x_{p,q}$ ($\gamma^x_{p,q}$): its value is one *iff* a β-message (γ-message) 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.
- $ALC_p[R_w]$: its value is one *iff* register R_w is allocated to PE$_p$; Otherwise zero.
- $PEstart^x_i$: its value is one *iff* PE$_i$ 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

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

subject to constraints (1) to (13) described below.

The following constraints are driven from to 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).

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

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

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.

$$PEsubst^x_k - ALC_k[R_w] \geq 0 \qquad (3)$$

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

Constraints (5), (6) and (7) correspond to rules (M$_{\beta 1}$), (M$_{\beta 2}$) and (M$_{\beta 3}$) of Fig. 2, respectively.

$$\sum_j \beta^x_{j,k} - PEsubst^x_k \geq 0 \qquad (5)$$

$$\sum_j \beta^x_{j,k}[R_z] + ALC_k[R_z] - ALC_k[R_w] \geq 0 \qquad (6)$$

$$\alpha^x_{u,j} - \beta^x_{j,k} \geq 0 \qquad (7)$$

Constraints (8), (9), (10) and (11) correspond to rules (M$_{\gamma 1}$), (M$_{\gamma 2}$), (M$_{\gamma 3}$) and (M$_{\gamma 4}$), respectively.

$$\gamma^x_{k,m} - PEsubst^x_k \geq 0 \qquad (8)$$

$$\sum_l \gamma^x_{l,m} + PEsubst^x_m \geq 1 \qquad (9)$$

$$\alpha^x_{u,l} + PEsubst^x_l - \gamma^x_{l,m} \geq 0 \qquad (10)$$

$$\sum_l \gamma^x_{l,m}[R_v] + ALC_m[R_v] \geq 1 \qquad (11)$$

Constraints (12) and (13) restrict the number of PE's which have registers R_w and $Rtmp_p$, respectively. The register $Rtmp$ is used only in $PEStart^x_p$ to save the input variable used in the event expression of t_x (say i^x).

$$\sum_p ALC_p[R_w] \geq 1 \qquad (12)$$

$$ALC_p[Rtmp_p.i^x] = 1 \text{ if } p = u; \text{ Otherwise } 0 \qquad (13)$$

Other Cost Criteria The following objective functions can be incorporated into our ILP model to minimize the communication costs in different cost criteria. Note that we let $Sz[R_w]$, F^x and $Pl_p[R_w]$ denote the size of resource R_w, the (approximate) firing frequency of a transition t_x and the cost of placing resource R_w on PE$_p$, respectively.

• Considering Size of Messages:

$$\text{Min}: \sum_x \left(\sum_q \alpha^x_{u,q} + \sum_p \sum_q \left(\beta^x_{p,q} + \gamma^x_{p,q} + \sum_w Sz[R_w] * \left(\beta^x_{p,q}[R_w] + \gamma^x_{p,q}[R_w] \right) \right) \right)$$

• Considering Execution Frequencies of Transitions:

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

• Considering Resource Placement Costs:

$$\text{Min}: \sum_x \left(\sum_q \alpha^x_{u,q} + \sum_p \sum_q \left(\beta^x_{p,q} + \gamma^x_{p,q} \right) \right) + \sum_p \sum_w Pl_p[R_w]$$

5 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.

We have applied our synthesis method to the distributed development of software that involves five engineers, given in [4]. We have modeled the workflow as a service specification (34 transitions and 20 registers) and derived 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 of one engineer. It took less than 143 seconds on PC with Athlon 750MHz to solve those optimization problems.

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

References

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

[2] A. Khoumsi and K. Saleh, "Two Formal Methods for the Synthesis of Discrete Event Systems," *Comp. Net. and ISDN Syst.*, Vol. 29, No. 7, pp. 759–780, 1997.

[3] 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.

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