Petri Net Based Protocol Synthesis with Minimum Communication Costs

Khaled El-Fakih[†]
Department of Computer Science
American University of Sharjah
P.O. Box 26666, Sharjah, United Arab Emirates

Hirozumi Yamaguchi
Graduate School of Engineering Science
Osaka University
1-3 Machikaneyamacho, Toyonaka, Osaka 560-8531, JAPAN

Gregor v. Bochmann
School of Information Technology and Engineering
University of Ottawa
800 King Edward Str., P.O.Box 450 Stn A, Ottawa, Ontario, K1N 6N5, Canada

Teruo Higashino
Graduate School of Engineering Science
Osaka University
1-3 Machikaneyamacho, Toyonaka, Osaka 560-8531, JAPAN

[†] Contact Person
E-mail: kelfakih@aus.ac.ae
FAX: +971-6-5155950

Abstract

Protocol synthesis is used to derive a *protocol specification*, that is, the specification of a set of application components running in a distributed system of networked computers, from a specification of services (called the *service specification*) to be provided by the distributed application to its users. Protocol synthesis reduces design costs and errors by specifying the message exchanges between the application components, as defined by the protocol specifications. In this paper, we propose a new synthesis method that generates optimized protocol specification. Both service and protocol specifications are described using extended Petri nets. Particularly, we propose two integer linear programming models that derive distributed applications with minimum communication costs. The first model determines an optimal allocation of resouces that minimizes communication costs and the second model minimizes the communication costs based on a given fixed allocation of resources. Moreover, our models can treat several reasonable cost criteria that could be used in various related application areas. Particularly, we have considered the following cost criteria:(a) the number of messages exchanged between different distributed applications, (b) the size of messages, (c) the number of messages based on frequency of executions,(d) communication channel costs, and (e) resource placement costs. An application example is given along with some experimental results.

Keywords

Distributed system, protocol synthesis, integer linear programming, Petri net

1 Introduction

Protocol synthesis is used to derive a *protocol specification*, that is, the specification of a set of application components running in a distributed system of networked computers, from a specification of services (called the *service specification*) to be provided by the distributed application to its users. The service specification is written as a program of a centralized system, and does not contain any message exchange between different physical locations. However, the implementation level's specification of the cooperating programs, called *protocol entities* (*PE's*), includes the message exchanges between these entities. Therefore, protocol synthesis methods have been used to specify and derive such complex message exchanges automatically in order to reduce the design costs and errors that may occur when manual methods are used.

There are a number of aspects to protocol synthesis that have been addressed in the literature. The aspect deals with implementing complex control-flows using different computational models such as CCS based models [6, 7], LOTOS [9, 10], Petri nets [15, 16, 19] and FSM/EFSM [11, 13]. The second aspect,

[20, 21, 22, 24, 25, 26], deals with satisfying the timing constraints specified by a given service specification in the derived protocol specification. This is important for real-time distributed systems.

The last aspect, [28, 27, 8, 12, 17, 18, 23], deals with the management of distributed resources such as files and databases. The objective is to determine how these distributed resources are read and updated by the different PE's in the context of a given resource allocation.

Some methods related to the last aspect, especially in our previous research work [28, 12, 17], consider a given service specification and derive a corresponding protocol specification assuming a given fixed resource allocation. However, in the context of distributed applications,there may be a large number of possible resource allocation, and their choice may have an important impact on the performance of the resulting system. It is therefore desirable to find an optimized resource allocation and protocol.

As an example, we considered a software development process using a Computer Supported Cooperative Work (CSCW) environment. This process is carried out cooperatively by multiple engineers (developers, designers, managers and others). Each engineer has his/her own workstation (PE) and participates in the development process using specific distributed resources (*e.g.* drafts, source codes, object codes,multimedia video and audio files, and others) which may be placed on different computers. Considering the need for managing such a process in the distributed environment, we describe the whole software development process (service specification) and derive the set of all the engineers' sub-processes (protocol specification). We also determine an optimal allocation of resources that would minimize the communication costs (such as file transfer costs).

In this paper, we extend our synthesis method presented in [28] and propose a new synthesis method that derives optimized distributed applications. Both service and protocol specifications are described using extended Petri nets. Particularly, based on a set of rules [28] for deriving a protocol specification, we formulate two Integer Linear Programming (ILP) models that minimize the communication costs of the protocol specification. The first model (henceforth denote as the *resource-allocation* model) determines an optimal allocation of resources that minimizes the communication costs and the second model (hence-

forth denoted as the *fixed-allocation* model) minimizes the communications costs based on a given fixed allocation of resources. Our ILP models can also treat several other reasonable cost criteria that could be used in various application areas for deriving protocol specifications. Particularly, we have considered the following cost criteria: (a) the number of messages to be exchanged between different PE's, (b) the size of messages to be exchanged between different PE's, (c) the number of messages based on frequency of executions, (d) communication channel costs, and (e) resource placement costs.

For a given cost function, based on the solution provided by an ILP model, our synthesis algorithm determines how, when, and which entities should send messages in order to obtain optimized distributed applications.

This paper is organized as follows. Section 2 gives examples of service specifications and protocol specifications. Section 3 provides an overview of our protocol synthesis method. Based on this method, we present in Section 4 a formulation of the two integer linear programming models that can be used to derive distributed applications with minimum communication costs. Moreover, in Section 5 we discuss the related communication cost criteria. In Section 6 we give an application example, and in Section 7 we conclude this paper and include our insights for future research.

2 Service Specifications and Protocol Specifications

2.1 Petri Net Model with Registers

We use an extended Petri net model called *Petri Net with Registers* (*PNR* in short) [17, 28] to describe both service specifications and protocol specifications of distributed systems. In this model, the service access points between the users and the system are modeled as gates, and the variables used inside the system, such as databases and files, are modeled as registers. Each transition t in a PNR has a label $\langle \mathcal{C}(t), \mathcal{E}(t), \mathcal{S}(t) \rangle$, where $\mathcal{C}(t)$ is a pre-condition (the firing condition of t), $\mathcal{E}(t)$ is an I/O event and $\mathcal{S}(t)$ is a set of assignment statements (which represent parallel updates of register values).

A transition t may fire if (a) each of its input places has a 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, the corresponding I/O

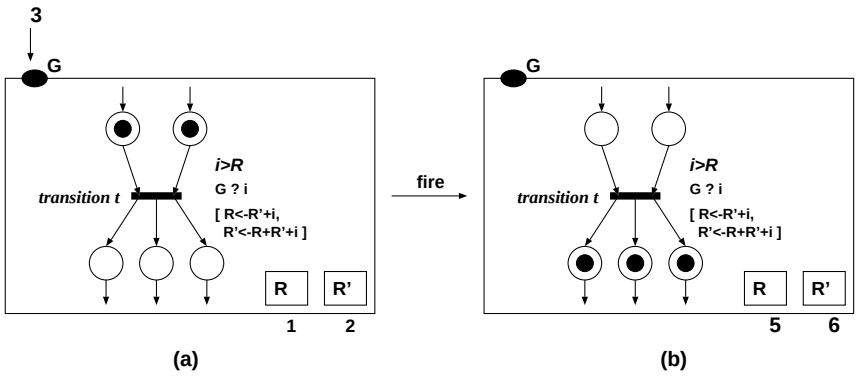

Figure 1: Register Values and Token Location before and after Firing Transition in PNR

event is executed, and the new values of registers are calculated and substituted in parallel as defined by $\mathcal{S}(t)$.

Consider, for example, transition t of Fig. 1(a) where $\mathcal{C}(t)$ ="$i > R$", $\mathcal{E}(t)$ ="$G?i$" and $\mathcal{S}(t)$ ="$\{R \leftarrow R' + i, R' \leftarrow R + R' + i\}$". Here, i denotes an input variable which holds an input value. The input value can be referred to only in this transition t. R and R' denote registers which contain values, and their values may be used and updated by all the transitions of the PNR. This means that registers are treated like global variables. G is a gate, that is, a service access point (interaction point) between users and the system. Note that "?" or "!" in $\mathcal{E}(t)$ indicate that $\mathcal{E}(t)$ is an input or output event, respectively.

Assume that an integer of value 3 has been given as input through gate G, and the current values of the registers R and R' are 1 and 2, respectively. In this case, since the value of the pre-condition "$i > R$" is true, the transition may fire. If it fires, event "$G?i$" is executed and the input value 3 is assigned to the input variable i. Then the assignments "$R \leftarrow R' + i$" and "$R' \leftarrow R + R' + i$" are executed in parallel. After the firing, the tokens are moved, and the values of the registers R and R' are 5 $(= 2 + 3)$ and 6 $(= 1 + 2 + 3)$, respectively (Fig. 1(b)).

Formally, an I/O event $\mathcal{E}(t)$ is one of the following types: "$G\ !exp$", "$G\ ?i$", or "τ". "$G\ !exp$" is an output event, and it means that the value of the expression "exp" is output through the gate G (all the

arguments in "exp" must be registers). "$G\ ?i$" is an input event and it means that the value given through G is assigned to the input variable "i". The event "τ" means that no external I/O event is associated with this transition. $\mathcal{S}(t)$ is a set of assignment statements, each of which has the form "$R \leftarrow exp$" where R is a register and exp is an expression whose arguments may be the input variable of $\mathcal{E}(t)$ or registers.

PNR is defined as a tuple $\langle T, P, A, M_0, G, R, I, \mathcal{C}, \mathcal{E}, \mathcal{S}, R_0 \rangle$ where $\langle T, P, A, M_0 \rangle$ is a Petri net, G is a set of gates, R is a set of registers, and I is a set of input variables. $\mathcal{C}$, $\mathcal{E}$ and $\mathcal{S}$ define the labels of transitions as explained above, and R_0 defines the initial values of the registers.

2.2 Service Specifications

At an abstract level, a distributed system is regarded as a non-distributed system which provides services as a single "virtual" machine. The number of actual PE's and communication channels between them are hidden. A specification of a distributed system at this level is called a *service specification* and denoted by $Sspec$ in this paper. Although the actual resources of a distributed system may be located on different physical machines, called protocol entities, the service specification, at this level, considers only one virtual machine.

For better readability and understanding, hereafter, we use the simple example of $Sspec$ shown in Fig. 2(a). A larger practical example is given in Section 6. $Sspec$ in Fig. 2(a) uses two gates G_{in} and G_{out} and two registers R and R'. At the initial marking, one token is assigned to place P_1, and therefore T_1 can fire if an input is given through G_{in}. When T_1 fires, the system updates the values of the registers R and R' simultaneously using the current values of register R' and the input i, respectively[1]. Then T_2 fires, and the system outputs the updated values of R and R' through gate G_{out} and returns to the initial marking.

2.3 Protocol Specifications

A distributed system may be considered as a communication system which consists of n protocol entities PE_1, PE_2, ... and PE_n. We assume a duplex and reliable communication channel between any pair of

[1] At the first firing of T_1, the initial value of R' is used to update R.

PE's (PE_i and PE_j). The PE_i and PE_j sides of the communication channel are represented as gates g_{ij} and g_{ji}, respectively. Moreover, we assume that all the registers and the gates for communication with the users are allocated to certain PE's in the distributed system.

Two PE's communicate with each other asynchronously by exchanging messages. A message is denoted by "M[list of values]" where "M" is one of the following three message types (α, β or γ) explained later. We assume that if PE_i executes an output event "$g_{ij}!M$[list of values]" on a transition, this message is sent through gate g_{ij} to the peer protocol entity PE_j. On reception, it is written into PE_j's receive buffer. If PE_j executes an input event "$g_{ji}?w$" with pre-condition "$ID(w) == M$" on a transition, PE_j removes the received message from its buffer and the message is kept in the input variable w. Note that the i-th value of the list included in the received message w will be denoted by $\#i(w)$.

In order to implement a distributed system which consists of n PE's, we must specify the behavior of these PE's. A behavior specification of PE_k is called a *protocol entity specification* and denoted by $Pspec_k$. A set of n protocol entity specifications is called a *protocol specification* and denoted as $Pspec_{1..n}$. We need a protocol specification in order to implement a given service specification.

Let us assume that there are two PE's (PE_1 and PE_2) in order to implement the service specification of Fig. 2(a). We also assume that PE_1 has both user gates G_{in} and G_{out} and register R, while PE_2 has register R'. Fig. 2(b) shows an example of $Pspec_{1..2}$ which provides the services of Fig. 2(a), based on the above allocation of resources. Note that some additional registers, called *temporary registers*, are used in this protocol specification to temporary keep values received in messages. If PE_i receives the value of register R from some other PE via a message, we assume a temporary register "$\_R$" (represented as a dotted box in the figure) to keep the value on PE_i. In Fig. 2(b), for simplicity of notations, both g_{12} and g_{21} are denoted as g, and internal events "τ", pre-conditions "true" and empty sets of assignment statements are omitted.

According to the protocol specification of Fig. 2(b) and its corresponding timing charts in Fig. 3, PE_1 first receives an input through G_{in} and checks the values of the firing condition $\mathcal{C}(T_1)$ on t_{1a}. Since it is

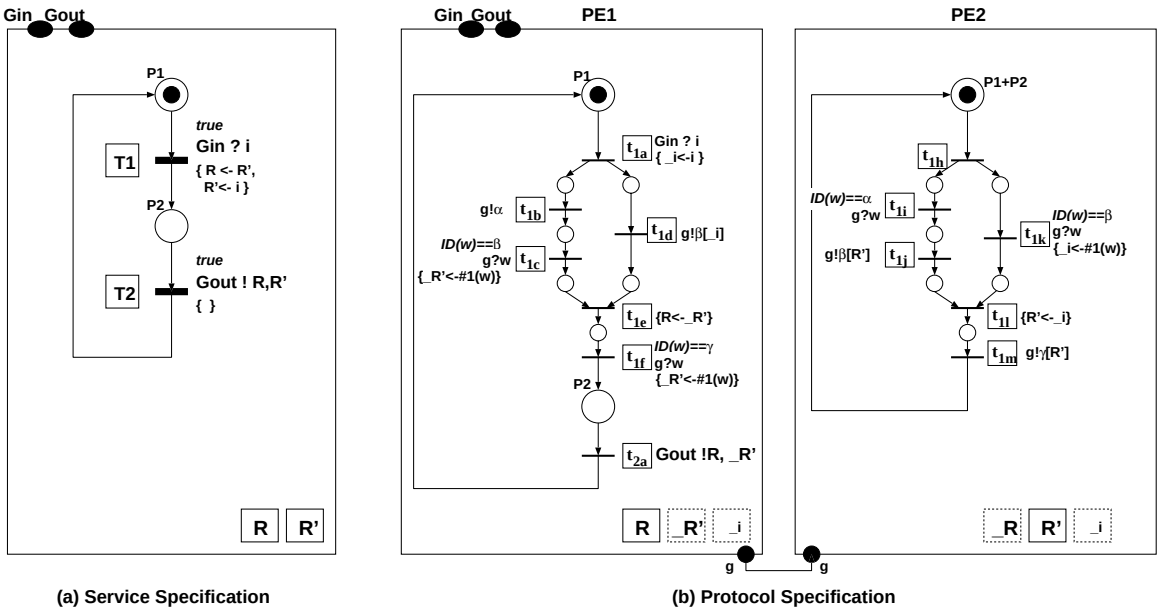

Figure 2: Service Specification and Protocol Specification

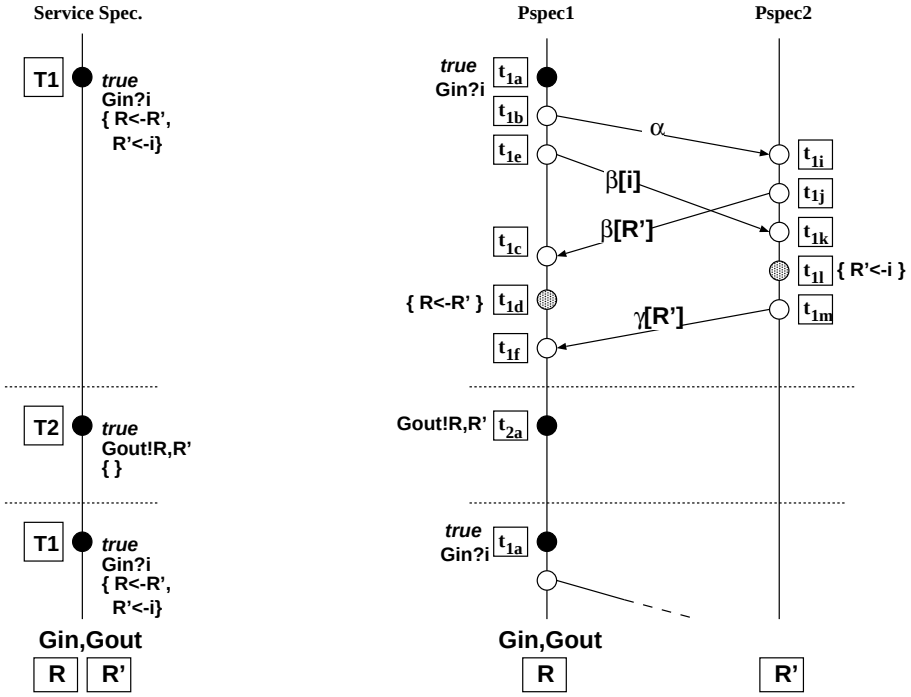

Figure 3: Service Specification and Protocol Specification : Timing Charts

always true, PE_1 executes $\mathcal{E}(T_1)$ on t_{1a} and keeps the received input i in the temporary register $\_i$. Then it sends a message "α" during transition t_{1b} in order to ask PE_2 to send the current value of R' which is necessary to update the value of R. PE_2 receives the message during transition t_{1i} and sends a message "$\beta[R']$" during transition t_{1j}. PE_1 receives the message during t_{1c} and now knows the value of R'. In parallel with the sending of the α-message, PE_1 sends the message "$\beta[\_i]$" during t_{1d} thereby sending the value of the input "i" to PE_2. PE_2 receives the message during t_{1k} and now knows the value of the input. After sending/receiving the β-messages, PE_1 and PE_2 know that now they can execute $\mathcal{S}(T_1)$ using the received values. They independently execute "$R \leftarrow R'$" and "$R' \leftarrow i$" during the transitions t_{1e} and t_{1l}, respectively. After the firing of t_{1e} and t_{1l}, the system is in a state where the service specification should check whether transition T_2 would be executed. For that purpose, PE_2 sends a message "$\gamma[R']$" in order to send the (updated) value of R' and let PE_1 know that the execution of "$R' \leftarrow i$" had been completed. When receiving the γ-message, PE_1 is ready to start the execution of T_2. After executing $\mathcal{E}(T_2)$ on t_{2a}, both PE_1 and PE_2 are back in their initial markings.

As shown in the above example, two PE's cooperate with each other in order to provide the same event sequences (including values) at the user gates G_{in} and G_{out} as specified in *Sspec*. Moreover, our synthesis method described below guarantees that the values of a register in *Sspec* and $Pspec_{1..2}$ are identical and the buffers of all the communication channels are empty at corresponding markings. For example, the marking of *Sspec* where place P_2 has a token and the marking of $Pspec_{1..2}$ where place P_1 of PE_1 and place "$P_1 + P_2$" of PE_2 have tokens are such corresponding markings. This is because our implementation never starts the execution of a transition unless the execution of all the previous transitions have been completed, and it allows us to easily keep consistency between *Sspec* and $Pspec_{1..2}$. It also allows us to use receive buffers of finite capacity for the communication channels, since we can determine the maximum number of messages that may be in transmit between any pair of protocol entities.

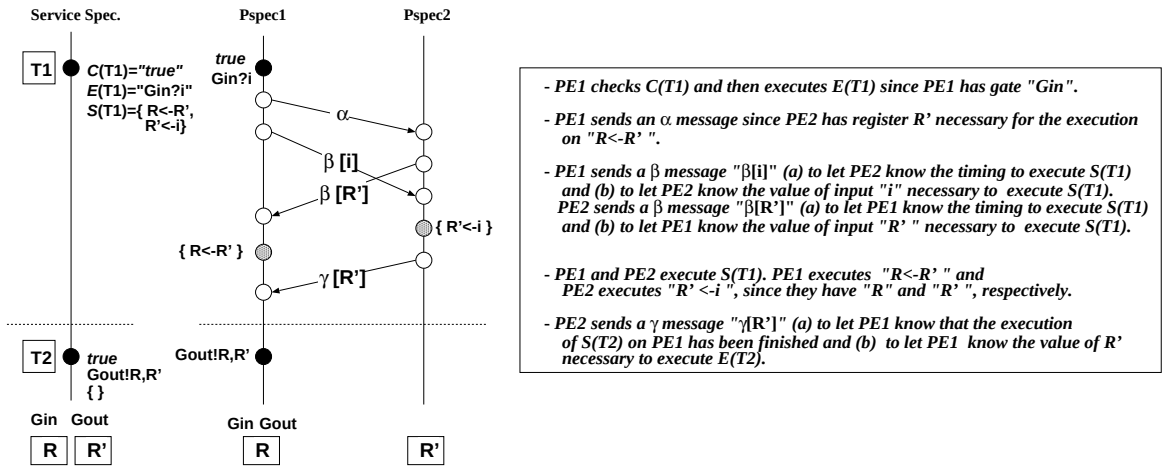

Figure 4: Applying Synthesis Rules to T_1

3 Synthesis Overview

We have presented in our previous work, especially in [28], the protocol synthesis method which is the basis for and thus highly relevant to the ILP model for optimal resource allocation presented in this paper. In order for the complete understanding of this model with the conviction of correctness, it is important to present our synthesis method designed to be suitable to the model.

The synthesis method derives a protocol specification from a given service specification and is based on a set of *synthesis rules* that specify how to execute each transition $T = \langle \mathcal{C}(T), \mathcal{E}(T), \mathcal{S}(T) \rangle$ of the service specification by the corresponding PE's in the protocol specification. Based on these rules, the behavior of all PE's and an optimal allocation of resources (registers and user gates) for minimum communication costs is determined. This leads to the specifications of all the PE's (protocol entity specifications) written in the same PNR model formalism.

3.1 Synthesis Rules

For executing a transition $T = \langle \mathcal{C}(T), \mathcal{E}(T), \mathcal{S}(T) \rangle$ of the service specification by a set of transitions of the PE's in the protocol specification, we use the following algorithm. Fig. 4 shows how our algorithm is

Table 1: Notation

Notation	
$\mathcal{C}(T)$, $\mathcal{E}(T)$, $\mathcal{S}(T)$	pre-condition, event and set of assignment statements of T
$PEstart(T)$	the PE where the gate used in $\mathcal{E}(T)$ is located
$PEsubst(T)$	the set of those PE's that contain a register updated by $\mathcal{S}(T)$
$PEstart(T \bullet\bullet)$	the set of the PE's that start the execution of the next transitions after T (*i.e.* $\bigcup_{T' \in T\bullet\bullet} PEstart(T')$ where $T \bullet\bullet$ is the set of the next transitions after T)

applied to transition T_1 of *Sspec* of Fig. 2(a). Note that the notation used in the following algorithm is summarized in Table 1.

- The PE that has the gate G used in $\mathcal{E}(T)$ (which we denote by $PEstart(T)$) decides to start the execution of T by checking the value of the pre-condition $\mathcal{C}(T)$. If it is true, $PEstart(T)$ executes the event $\mathcal{E}(T)$.

  In Fig. 4, $PEstart(T_1)$ is PE_1 since PE_1 has gate G_{in}. PE_1 checks the value of $\mathcal{C}(T_1)$="*true*" (always true) and then executes $\mathcal{E}(T_1)$="$G_{in}?i$".

- Then, $PEstart(T)$ sends synchronization messages called *α-messages* to those PE's that have the registers used to execute the assignment statements in $\mathcal{S}(T)$. On reception, those PE's send the register values to the PE's that execute assignment statements whose expressions require those register values. Note that *α-messages* are also sent to some other PE's. This is explained later.

  In Fig. 4, we assume that "$R \leftarrow R'$" and "$R' \leftarrow i$" are executed by PE_1 and PE_2 respectively, since PE_1 has R and PE_2 has R' (see Section 4 for the discussion of this allocation problem). For this execution, PE_1 needs the value of R' and PE_2 needs the value of i. Here, since PE_1 does not have "R'", the value must be sent from PE_2. An α-message is sent from $PEstart(T_1) = PE_1$ to PE_2 in order to let PE_2 send the value of R'. Also, since PE_2 does not know the value of i, it must be sent from PE_1 to PE_2. Here, since PE_1 is itself $PEstart(T_1)$, it knows the timing to send the value of "i" (just after the execution of $\mathcal{E}(T_1)$). Therefore no α-message is sent from PE_1 to itself.

- On the reception of an α-message, the PE sends the values of registers to those PE's that need those values for the execution of part of assignment statements in $\mathcal{S}(T)$ (the set of these PE's is denoted as $PEsubst(T)$). These messages are called *β-messages*. Using those register values, each PE in $PEsubst(T)$ executes the assignment statements. Note that if a PE in $PEsubst(T)$ does not need any register value for the execution of the assignment statements, $PEstart(T)$ directly sends a β-message to the PE in order to know that it can start executing its assignment statements.

  In the example of Fig. 4, we have $PEsubst(T_1) = \{PE_1, PE_2\}$ since PE_1 has register R and PE_2 has R'. PE_1 sends a β-message to PE_2 with the value of i, which is used by PE_2 to execute "$R' \leftarrow i$". Moreover, PE_2 sends a β-message to PE_1 with the value of R', which is used by PE_1 to execute "$R \leftarrow R'$". Then PE_1 and PE_2 execute "$R \leftarrow R'$" and "$R' \leftarrow i$", respectively.

- After all assignment statements in $\mathcal{S}(T)$ are executed, each PE in $PEsubst(T)$ sends so-called *γ-messages* to those PE's that will start the execution of the next transitions. The set of those PE's is denoted as $PEstart(T \bullet\bullet)$. These messages confirm the completion of the assignment statements and also contain the values of registers necessary to start the execution of next transitions. Note that the PE's that do not belong to $PEsubst(T)$ may also need to send some values of registers to the PE's in $PEstart(T \bullet\bullet)$. These values are also sent as γ-messages. In this case, α-messsages are sent to these PE's to initiate the sending of γ-messages.

  In our example, PE_2 sends a γ-message to $PEstart(T_1 \bullet\bullet) = \{PE_1\}$. PE_1 then knows the value of R' and the fact that the execution of $\mathcal{S}(T_1)$ on PE_2 has been completed.

The above algorithm is presented as a set of rules called *synthesis rules* (see Figure 5). These rules are classified into action rules and message rules. The action rules specify which PE's should check the pre-condition, execute the I/O event and assignment statements of T. The message rules specify which PE's should exchange messages. The contents and types of these messages are also specified.

Consequently, three types of messages are exchanged for the execution of a transition T:

- α-**messages** are sent from the PE that starts the execution of T (*i.e.* $PEstart(T)$) to the PE's that send β-messages (and PE's that send γ-messages and are not in $PEsubst(T)$). Their reception leads to the sending of β-messages and/or γ-messages. An α-message does not contain any register value.

- β-**messages** are sent from PE's that have registers to be used to execute assignment statements of $\mathcal{S}(T)$, to those PE's that execute these assignment statements. The latter PE's form the set $PEsubst(T)$. Note that for the PE's that need no register values for the execution of the assignment statements, β-messages are sent from $PEstart(T)$ for synchronization. The reception of β-messages leads to the execution of the assignment statements.

- γ-**messages** are sent from the PE's in $PEsubst(T)$ and PE's that have registers to be used to check/execute $\mathcal{C}(T')/\mathcal{E}(T')$ to PE's in $PEstart(T \bullet\bullet)$, where T' is a next transition after T. They let the PE's in $PEstart(T \bullet\bullet)$ know the values of registers required for $\mathcal{C}(T')/\mathcal{E}(T')$ and the timing for executing the next transition.

Our synthesis method assumes that the Petri net of the service specification is a live and safe *free-choice net*[2, 3]. A free-choice net is a sub-class of Petri nets which has simple choice structures. It is known that a live and safe free-choice net can be decomposed into a set of finite state machines [2, 3] and this property is used in our algorithm. In addition, we assume that for two transitions T and T' of $Sspec$ in a choice structure, $PEstart(T) = PEstart(T')$ (*i.e.* the gates in $\mathcal{E}(T)$ and $\mathcal{E}(T')$ are allocated to the same PE). This guarantees that a single PE makes the decision to select the next transition in the choice structure. Otherwise an agreement would be needed among several PE's to make this decision. This would be done by implementing a leader election algorithm as the one shown in [5]. Finally, it is assumed that for two transitions T and T' of $Sspec$ that may be executed in parallel, there is no register that is updated by one and referred or also updated by another. This assumption is used to prevent the inconsistency that may result in having multiple accesses to the same register. This assumption may also be relaxed by implementing a mutual exclusion algorithm (see for instance [5]).

Action Rules

($\mathbf{S}_{A1}$) PE_u that has the gate G used in $\mathcal{E}(T)$ checks that

(1) the value of $\mathcal{C}(T)$ is true,

(2) the execution of the previous transitions of T is completed

(3) an input has been given through G, if $\mathcal{E}(T)$ is an input event.

Then PE_u executes $\mathcal{E}(T)$. This PE_u is denoted $PEstart(T)$.

($\mathbf{S}_{A2}$) After ($\mathbf{S}_{A1}$), each PE (say PE_k) executes the subset of the assignment statements of $\mathcal{S}(T)$ that update the registers allocated to PE_k. The set of these PE's is denoted by $PEsubst(T)$. These assignment statements are executed when the corresponding (β-messages) are received (see below).

Message Rules

($\mathbf{S}_{M\alpha}$) After ($\mathbf{S}_{A1}$), $PEstart(T)$ only sends α-messages. The PE's to which α-messages are sent are determined in ($\mathbf{S}_{M\beta3}$) and ($\mathbf{S}_{M\gamma3}$).

($\mathbf{S}_{M\beta1}$) Each $PE_k \in PEsubst(T)$ must receive at least one β-message from some PE's (each called PE_j) in order to know the timing to execute $\mathcal{S}(T)$. This message also lets PE_k know the values of registers used in $\mathcal{S}(T)$ (see ($\mathbf{S}_{M\beta2}$)).

($\mathbf{S}_{M\beta2}$) For each register R_h that is used to execute $\mathcal{S}(T)$ by PE_k, PE_k must receive its value through a β-message if R_h is not allocated to PE_k.

($\mathbf{S}_{M\beta3}$) Each PE_j that sends a β-message to $PE_k \in PEsubst(T)$ knows the timing to send the message by receiving an α-message from $PEstart(T)$ unless PE_j is $PEstart(T)$.

($\mathbf{S}_{M\gamma1}$) Each $PE_m \in PEstart(T \bullet\bullet)$, where $T \bullet\bullet$ is the set of the next transitions after T, must receive a γ-message from each $PE_k \in PEsubst(T)$ after (A_{A2}), except where $m = k$. This lets PE_m know that the execution of $\mathcal{S}(T)$ had been completed on PE_k.

If $PEsubst(T)$ is empty, PE_m must receive at least one γ-message from any PE in order to know that the execution of T had been completed. γ-messages also let PE_m know the values of registers used in the pre-conditions and/or events of next transitions (see ($\mathbf{S}_{M\gamma2}$)).

($\mathbf{S}_{M\gamma2}$) For each register R_z used by PE_m to start the execution of the next transitions of T, PE_m must receive its value through a γ-message if R_z is not allocated to PE_m.

($\mathbf{S}_{M\gamma3}$) Each PE (say PE_l) that sends a γ-message to $PE_m \in PEstart(T \bullet\bullet)$ must be in $PEsubst(T)$ (see ($\mathbf{S}_{M\gamma1}$)), must receive an α-message from $PEstart(T)$ or must be $PEstart(T)$, in order to know the timing to send the γ-message to PE_m.

Figure 5: *Synthesis Rules in Detail*

3.2 Synthesis of Protocol Specifications

Based on the synthesis rules, a set of actions and message exchanges to be executed on each PE is defined for each transition T of $Sspec$. Then these actions and message exchanges are represented, for each PE, as a set of transitions where two transitions are connected through a place if a temporal ordering between the transitions is specified in the synthesis rules (*e.g.* an α-message must be received before the corresponding β-messages are sent). As a result, for each transition T in $Sspec$, a set of sub-PNR's $SPnet_1(T)$, ..., $SPnet_n(T)$ are produced for the n protocol entities. Afterwards, an intermediate protocol entity specification of PE_i (denoted as $\overline{Pspec_i}$) is derived by connecting all sub-PNR's $SPnet_i(T)$ in the same way as the transitions T are connected in $Sspec$. More specifically, $Pspec_i$ is obtained using the net structure of $Sspec$, by replacing each T with the corresponding sub-PNR $SPnet_i(T)$. Finally, a protocol entity specification of PE_i ($Pspec_i$) is derived by removing ε-transitions from $\overline{Pspec_i}$. The removing technique is based on the well-known technique to remove ε-moves in finite automata. In order to apply this technique to our PNR model containing parallel synchronization, we use the fact that a live and safe free-choice net can be decomposed into a set of live and safe finite state machines (FSM's) [2, 3]. The reader may refer to [28] for the details of the above mentioned steps.

4 Integer Linear Programming Models for Deriving Optimized Distributed Applications

As shown below (see also [28]), the above synthesis rules do not derive optimized distributed applications. That is, they do not derive PE's with minimum communication costs. We consider communication costs as a primary cost of building distributed applications. Accordingly, in the following two subsections, we extend our synthesis method [28] with two 0-1 ILP models that allow us to derive optimized applications. Moreover, in the following section, we present some other reasonable cost criteria that can be used in the derivation of optimized applications. The first model, called *resource-allocation model*, determines an optimal allocation of resources that minimizes communication costs and the second, called *fixed-allocation*

model, minimizes the communications costs based on a given fixed allocation of resources. The evaluation of fixed-allocation model consumes less memory resources and requires less computational time than the variable-allocation model since it takes into account that for a transition t_x of $Sspec$ the PE's that will execute the substitution statements (i.e. $PEsubst_p^x$) and the allocation of resources to PE's are known. For a given cost function and based on the solution provided by an ILP model, the synthesis algorithm determines which entitities should send the α, β, or γ messages, execute the substitution statement (for the resource-allocation model), and when to send these messages. We note that in our previous work published in [27], we have presented a preliminary version of the resource-allocation model, however, we did not include any application example nor experimental results.

4.1 An Integer Linear Programming Model for Optimal Resource Allocation

The communication costs (e.g. the number of messages) depend on the allocation of resources amongst different PE's. Therefore, we may carefully design this allocation in order to minimize the communication costs.

As a simple example, let us consider the timing charts in Fig. 6(b). This chart is similar to that in Fig. 4 obtained when R and R' are allocated to PE_1 and PE_2, respectively. If we use another allocation where both R and R' are allocated to PE_2, we obtain a different protocol specification whose timing chart is shown in Fig. 6(c). We note that the allocation of the user gates are usually fixed by the nature of the application, and therefore cannot be changed freely. These examples show that the resource allocation affects the communication costs of the protocol specifications and that it is not easy to find an optimal allocation, given the complex message exchanges between the PE's.

We formulate this optimal resource-allocation problem as an ILP problem. For this purpose, we introduce the following 0-1 integer (boolean) variables:

- Each of the following variables represent the fact that a message is sent from one PE to another.
  - $\alpha_{u,q}^x$: its value is one *iff* an α-message is sent from PE_u=PEstart(t_x) to PE_q in the execution

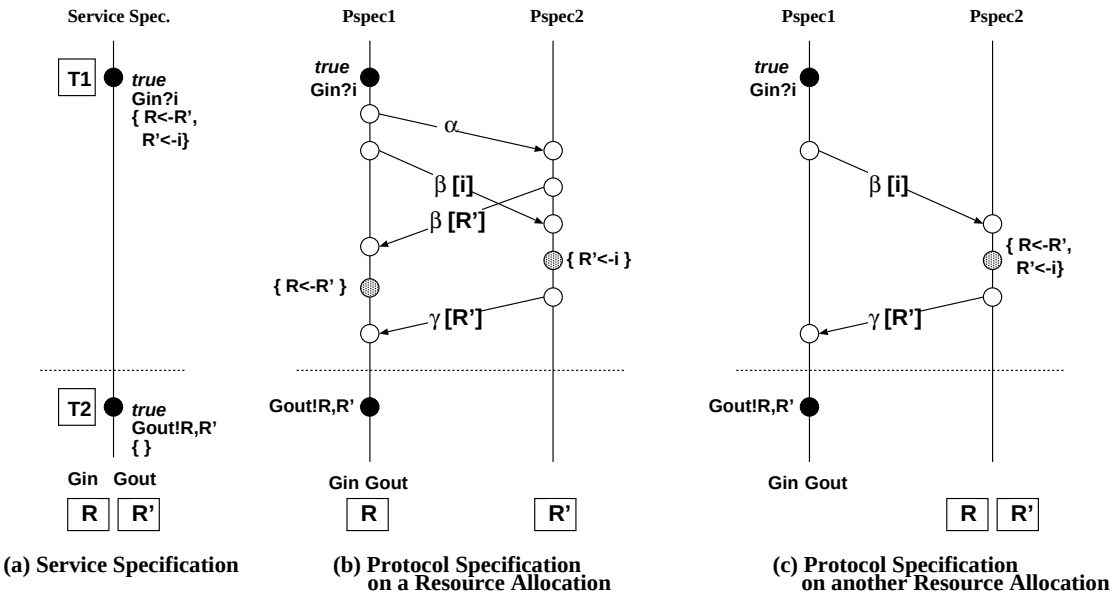

Figure 6: Two Protocol Specifications: they provide the same service as the service specification, however their resource allocations are different.

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.
- $PEsubst^x_p$: 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 (13) described below.

Objective Function:

$$\text{Min}: \sum_{x} \left(\sum_{q} \alpha^{x}_{u,q} + \sum_{p} \sum_{q} \left(\beta^{x}_{p,q} + \gamma^{x}_{p,q} \right) \right)$$

The following constraints are derived from the definition of the 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).

$$\beta^{x}_{j,k} + ALC_j[R_w] - 2\beta^{x}_{j,k}[R_w] \geq 0 \quad (1) \qquad \gamma^{x}_{l,m} + ALC_m[R_w] - 2\gamma^{x}_{l,m}[R_w] \geq 0 \quad (2)$$

According to rule (S_{A2}), each PE that has a register R_w whose value is changed in the set of substitution statements $\text{S}(t_x)$, must be the one that executes this substitution statement.

$$PEsubst^{x}_{k} - ALC_k[R_w] \geq 0 \quad (3)$$

$$\sum_{w} ALC_k[R_w] - PEsubst^{x}_{k} \geq 0 \quad (4)$$

The following three constraints correspond to rules ($\text{S}_{M\beta 1}$), ($\text{S}_{M\beta 2}$), and ($\text{S}_{M\beta 3}$), respectively.

$$\sum_{j} \beta^{x}_{j,k} - PEsubst^{x}_{k} \geq 0 \quad (5)$$

$$\sum_{j} \beta^{x}_{j,k}[R_h] + ALC_k[R_h] \geq 1 \quad (6)$$

$$\alpha^{x}_{u,j} - \beta^{x}_{j,k} \geq 0 \quad (7)$$

The following two constraints correspond to rule ($\text{S}_{M\gamma 1}$)

$$\gamma^{x}_{k,m} - PEsubst^{x}_{k} \geq 0 \quad (8)$$

$$\sum_{l} \gamma^{x}_{l,m} + PEsubst^{x}_{m} \geq 1 \quad (9)$$

Constraints (10) and (11) correspond to rules ($S_{M\gamma 2}$) and ($S_{M\gamma 3}$), respectively:

$$\sum_{l} \gamma^{x}_{l,m}[R_z] + ALC_m[R_z] \geq 1 \tag{10}$$

$$\alpha^{x}_{u,l} + PEsubst^{x}_{l} - \gamma^{x}_{l,m} \geq 0 \tag{11}$$

Constraints (12) and (13) restrict the number of PE's that have registers R_w and $\_R$, respectively. The register $\_R$ is used only in $PEStart^{x}_{p}$ to save the input variable used in the event expression of t_x (say i^x).

$$\sum_{p} ALC_p[R_w] \geq 1 \tag{12}$$

$$ALC_p[\_R_p.i^x] = 1 \text{ if } p = u; \text{ Otherwise } 0 \tag{13}$$

Note that a general 0-1 ILP problem is known to be a hard problem of exponential complexity, and therefore a solution to our optimization problem is not readily available for large-scale systems in terms of the numbers of transition, resources and PE's. There are some possibilities to tackle this complexity: (i) We may adopt a simpler model for applications with a given fixed allocation of resources. This is done in the following subsection in order to reduce computation time and memory resources. (ii) We may use heuristic algorithms, for instance genetic and simulated annealing algorithms as in [1]. This is part of our future work. However, it is worth mentioning that we have developed a genetic algorithm for the following fixed-allocation model and our preliminary experiments show that the algorithm determines a (near) optimal solution in a reasonable time.

4.2 An Integer Linear Programming Model for a Fixed Allocation of Resources

In some application areas, the allocation of resources to different PE's is given. Thus, in order to mimimize communication costs, it is worth to adopt a simpler model than that presented in the previous subsection.

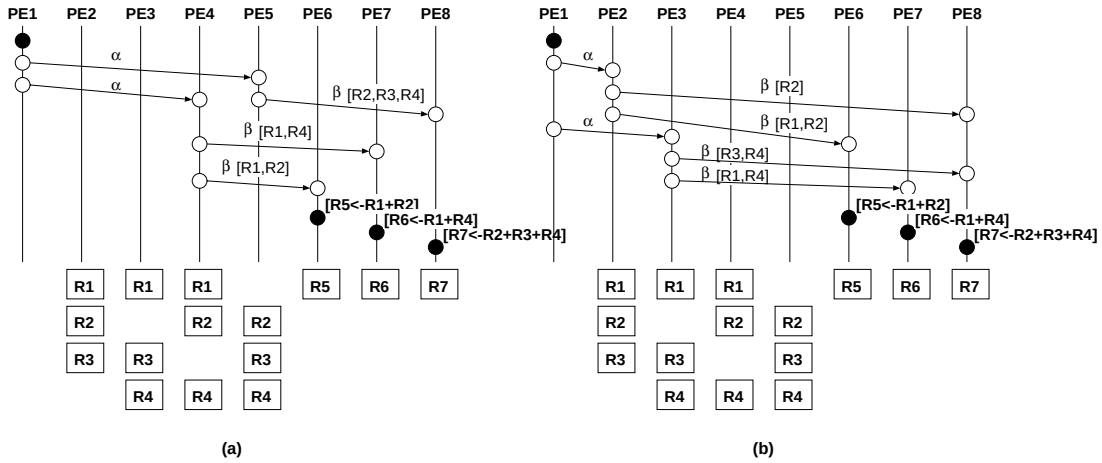

Figure 7: Two Protocol Specifications: they provide the same service as the service specification, however their number of messages are different, (a) has optimal and (b) has non-optimal message exchange

This model takes into account that for a transition t_x of $Sspec$ the PE's that will execute the substitution statements (i.e. $PEsubst_p^x$) and the allocation of resources to PE's are known.

As described in [28], here we note that even if the allocation of resources to different PE's is given, the number of messages exchanged between the entities for the execution of a transition in $Sspec$ may not be unique because a register may be allocated to more than one PE and several input/register values may be sent in one message. For example, let us assume that there are eight PE's as in Fig. 7(a). PE_6, PE_7 and PE_8 should change the values of registers R_5, R_6 and R_7, respectively. However, they do not have the registers necessary to change these values as PE_2, PE_3, PE_4 and PE_5 do. PE_1 is $PEstart(t)$. Fig. 7(a) shows the optimal way that uses five messages to send the necessary values to R_5, R_6 and R_7. However, there are many ways to send these values, Fig. 7(b) is one of them, and it uses six messages.

We consider the number of messages as a primary cost criteria for distributed systems. Moreover, if we assume that the size of a message is small, then the overhead for sending it as a packet throug a high-speed network is high. To reduce this cost, the minimum number of messages exchanged for simulating a transition t_x of $Sspec$ for a given resource allocation has to be determined.

Using the relevant variables defined in the previous subsection, for each transition t_x of $Sspec$, we mimimize the number of messages exchanged between different PE's for the execution of t_x by minimizing the following objective function, subject to constraints (14) to (22) given below.

Objective Function:

$$\text{Min}: \sum_q \alpha_{u,q} + \sum_p \sum_q (\beta_{p,q} + \gamma_{p,q})$$

The following constraints are drived from the definition of the variables. According to constraint (14), 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. The same reasoning applies to constraint (15).

$$\beta_{j,k} - \beta_{j,k}[R_w] \geq 0 \qquad (14) \qquad\qquad \gamma_{l,m} - \gamma_{l,m}[R_w] \geq 0 \qquad (15)$$

The following three constraints correspond to rules ($\text{S}_{M\beta 1}$), ($\text{S}_{M\beta 2}$), and ($\text{S}_{M\beta 3}$), respectively.

$$\sum_j \beta_{j,k} \geq 1 \qquad (16)$$

$$\sum_j \beta_{j,k}[R_h] \geq 1 \qquad (17)$$

$$\alpha_{u,j} - \beta_{j,k} \geq 0 \qquad (18)$$

The following two constraints correspond to rule ($\text{S}_{M\gamma 1}$)

$$\gamma_{k,m} - PEsubst_k \geq 1 \qquad (19)$$

$$\sum_l \gamma_{l,m} \geq 1 \qquad (20)$$

Constraints (21) and (22) correspond to rules ($\text{S}_{M\gamma 2}$) and ($\text{S}_{M\gamma 3}$), respectively:

$$\sum_l \gamma_{l,m}[R_z] \geq 1 \qquad (21)$$

$$\alpha_{u,l} - \gamma_{l,m} \geq 0 \qquad (22)$$

We note that different optimization criteria may be considered by adopting corresponding objective functions. For example, instead of considering the number of messages, we may also consider the size and/or the cost of sending messages over particular communication channels. In the following section, we consider some cost criteria that could be used in minimizing the communication costs of the derived protocol entities.

5 Other Cost Criteria

In this section, we incorporate into our resource-allocation model some other cost criteria that could be adopted during the minimization of the communication costs of the derived protocol specification. One may select a criterion according to the application area and the underlying network architecture. We note that all crieteria except the last one, can also be used by the fixed-allocation model since the resource placement costs are fixed in this model.

5.1 Considering Size of Messages

In most application areas, the size of resources exchanged between different PEs plays an important factor in determining their communication costs. We let $Size[R_w]$ denote the size of resource R_w, and reformulate our resource-allocation ILP model objective function as follows.

$$\text{Min}: \sum_{x} \left(\sum_{q} \alpha^{x}_{u,q} + \sum_{p} \sum_{q} \left(\beta^{x}_{p,q} + \gamma^{x}_{p,q} + \sum_{w} Size[R_w] * \left(\beta^{x}_{p,q}[R_w] + \gamma^{x}_{p,q}[R_w] \right) \right) \right)$$

5.2 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 cases, one might want to consider the frequencies of transition executions during the protocol derivation. In general, this is a dynamic property of the system, however, an approximation of the firing frequency may be derived by firing vector analysis or simulation of Petri nets, which has been investigated extensively [2].

Let F^x denote the (approximate) firing frequency of a transition t_x. We incorporate F^x into our resource-allocation ILP model as follows.

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

5.3 Considering Channel Costs

For application areas that use many communication channels with different channels costs,we let $ChannelCost_{i,j}$ denote the cost to send a message from PE_p to PE_q. Then we incorporate these costs into our resource-allocation ILP model as follows:

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

5.4 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 resource-allocation ILP model objective function as follows:

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

6 An Application Example and Experimental Results

Protocol synthesis methods have been applied to many applications such as communication protocols, factory manufacturing systems [15], distributed cooperative work management [14] and so on. In the following subsection we apply our synthesis method to the distributed development of software We derive for the specification of this application the corresponding protocol specifications with minimum communication costs using the different cost criteria presented in the previous section.

We have developed an automated system to generate for a given specification, the variable allocation) ILP model and its constraints. Then, we have used the tool lp_solve[30] on a PC with Athlon 750Hz to solve the ILP problem and determine the minimum communication costs of the derived PE's using the different cost criteria described in Section 5.

6.1 The ISPW-6 Example and its Experimental Results

In this subsection, we apply our synthesis method to the distributed development of software that involves five engineers (project manager, quality assurance, design, and two software engineers). Each engineer has his/her own machine connected through a network, and participates in the development through a gate (interfaces) of this machine, using distributed resources placed on these machines. This distributed development process includes tasks for scheduling and assigning tasks, design modification, design review, code modification, test plan modification, modification of unit test packages, unit testing, and progress monitoring. The engineers cooperate with each other to finish these sub-sequential tasks in a suitable order. The reader may refer to *ISPW-6 Core Problem* [29] for a complete description of this process, which was provided as an example to help the understanding and comparison of various approaches to process modeling.

Fig. 8 shows a workflow model of the above development process using *PNR*, where the engineers and resources needed to accomplish the tasks are indicated. We note that, for convenience, we do not show the progress monitoring tasks in Fig. 8.

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 of one engineer. Particularly, we have used our developed automated system and generated from the given specification the corresponding resource-allocation ILP problem and its constraints. Then, we have used lp_solve[30] to solve the problem using the different costs criteria. Table 2 contains the optimal resource allocation of the given problem and the execution times taken by lp_solve to solve the problem using the

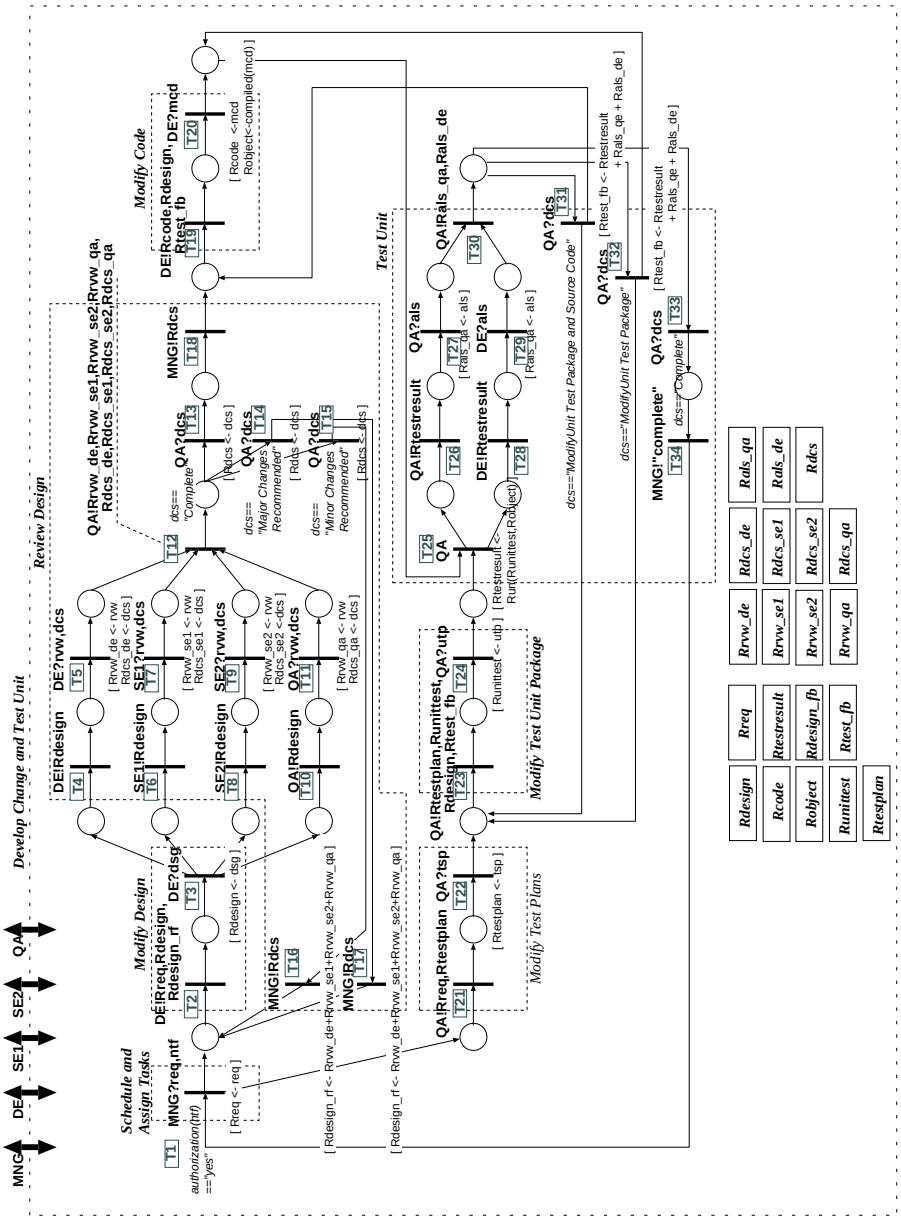

Figure 8: Modeling the ISPW-6 Core Problem

following cost criteria: (a) the number of messages as in our ILP model of Section 4, (b) the sizes of resources depicted in Table 3, (c) the execution frequencies of transitions depicted in Table 4, (d) the communication channel costs depicted in Table 5, and (e) the resource placement costs depicted in Table 6.

These experimental results show that for various cost criteria, we obtain different optimal allocations. Moreover, our method can determine optimal resource allocations for various cost criteria in a reasonable time.

7 Conclusion

In this paper, we have proposed a Petri-net based method for deriving an optimized protocol specification (distributed application) from a given service specification. In particular, we have presented two integer linear programming models that derive distributed applications with minimum communication costs. The first model determines an optimal allocation of resouces that minimizes communication costs and the second model minimizes the communication costs based on a given fixed allocation of resources. Furthermore, the models can treat several reasonable cost criteria that could be used in various related application areas. Specifically, we have considered the following cost criteria:(a) the number of messages exchanged between different distributed applications, (b) the size of messages, (c) the number of messages based on frequency of executions,(d) communication channel costs, and (d) cost of resource placement. An application example is given along with the experimental results.

In general, the solution to our integer linear programming optimization problems is not readily available for large systems (in terms of the numbers of transition, resources and PE's). Consequently, we are developing heuristic algorithms, such as genetic and simulated annealing algorithms, to solve effeciently the optimization problems. Our preliminary experiments with a genetic algorithm that we have developed for the fixed-allocation problem show that the algorithm finds (near) optimal solutions in a very reasonable time. Our future work is to develop an integrated development environment for distributed systems based on our proposed method, including a tool support for specifying service requirements (service

	PE_{mng}	PE_{de}	PE_{se1}	PE_{se2}	PE_{qa}	Time (second)
Gate (Fixed)	MNG	DE	$SE1$	$SE2$	QA	
(a)		R_{req} R_{design} R_{design_fb} R_{rvw_de} R_{dcs_de} R_{code} R_{test_fb} $R_{testplan}$	R_{rvw_se1} R_{dcs_se1} $R_{unittest}$ $R_{testresult}$	R_{rvw_se2} R_{dcs_se2}	R_{rvw_qa} R_{dcs_qa} R_{dcs} R_{object} R_{alc_qa} R_{als_de}	112
(b)	R_{req}	R_{rvw_de} R_{dcs_de} R_{design} R_{design_fb} R_{als_de}	R_{rvw_se1} R_{dcs_se1}	R_{rvw_se2} R_{dcs_se2}	$R_{testplan}$ $R_{unittest}$ $R_{testresult}$ R_{test_fb} R_{rvw_qa} R_{dcs_qa} R_{als_qa}	109
(c)	R_{req} R_{code} R_{object}	R_{rvw_de} R_{dcs_de} R_{design} R_{design_fb} R_{als_de}	R_{rvw_se1} R_{dcs_se1}	R_{rvw_se2} R_{dcs_se2}	$R_{testplan}$ $R_{unittest}$ $R_{testresult}$ R_{test_fb} R_{rvw_qa} R_{dcs_qa} R_{als_qa} R_{dcs}	142
(d)	R_{dcs}	R_{rvw_de} R_{dcs_de}	R_{rvw_se1} R_{dcs_se1}	R_{req} R_{design} R_{design_fb} R_{rvw_se2} R_{dcs_se2} R_{code} R_{test_fb} R_{object} $R_{testresult}$	R_{rvw_qa} R_{dcs_qa} $R_{testplan}$ $R_{unittest}$ R_{als_qa} R_{als_de}	135
(e)	R_{req} R_{code} R_{object} R_{dcs}	R_{rvw_de} R_{dcs_de} R_{design} R_{design_fb}	R_{rvw_se1} R_{dcs_se1}	R_{rvw_se2} R_{dcs_se2}	$R_{testplan}$ $R_{unittest}$ $R_{testresult}$ R_{test_fb} R_{rvw_qa} R_{dcs_qa} R_{als_qa} R_{als_de}	20

Table 2: Optimal Resource Allocation and Derivation Time Using as Cost Criterion (a) the Number of Messages, (b) the Size of Message, (c) the Execution Frequencies of Transitions, (d) the Communication Channel and (e) the Resource Placement Costs

specifications) through a graphical interface, synthesizing optimized protocol specifications, and generating Java code from the optimized protocol specifications.

References

[1] N. Mansour and K.El-Fakih, "Simulated Annealing and Genetic Algorithms for Optimal Regression Testing," *Journal of Software Maintenance*, 1(11), 19-34, 1999.

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

[3] E. Best, "Structure Theory of Petri Nets: the Free Choice Hiatus," *Advances in Petri Nets Part I: Petri Nets, Central Models and Their Properties, in Lecture Notes in Computer Science*, Vol. 254, pp. 168-205, 1986.

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

[5] N. A. Lynch, "Distributed Algorithm," *Morgan Kaufmann Publishers*, 1996.

[6] V. Carchiolo, A. Faro and D. Giordano, "Formal Description Techniques and Automated Protocol Synthesis," *Journal of Information and Software Technology*, Vol. 34, No. 8, pp. 513–421, 1992.

[7] H. Erdogmus and R. Johnston, "On the Specification and Synthesis of Communicating Processes," *IEEE Trans. on Software Engineering*, Vol. SE-16, No. 12, 1990.

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

[9] R. Langerak, "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.

[10] 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.

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

[12] T. Higashino, K. Okano, H. Imajo and K. Taniguchi, "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.

[13] M. Nakamura, Y. Kakuda and T. Kikuno, "Component-based Protocol Synthesis from Service Specifications," *Computer Communications Journal*, Vol. 19, No. 14, pp.1200-1215, Dec. 1996.

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

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

[16] F.-Y. Wang, K. Gildea, H. Jungnitz and D.D. Chen, "Protocol Design and Performance Analysis for Manufacturing Message Specification: A Petri Net Approach," *IEEE Trans. on Industrial Electronics*, Vol. 41, No. 6, pp. 641–653, 1994.

[17] H. Yamaguchi, K. Okano, T. Higashino and K. Taniguchi, "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.

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

[19] A. Al-Dallal and K. Saleh, "Protocol Synthesis Using the Petri Net Model," *Prof. of 9th Int. Conf. on Parallel and Distributed Computing and Systems (PDCS'97)*, 1997.

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

[21] M. Kapus-Koler, "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.

[22] A. Khoumsi, G. v. Bochmann and R. Dssouli, "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.

[23] A. Khoumsi and G. v. Bochmann, "Protocol Synthesis Using Basic LOTOS and Global Variables," *Proc. of 1995 Int. Conf. on Network Protocols (ICNP'95)*, 1995.

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

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

[26] J. -C. Park and R. E. Miller, "Synthesizing Protocol Specifications from Service Specifications in Timed Extended Finite State Machines," *Proc. of 17th Int. Conf. on Distributed Computing Systems (ICDCS-17)*, 1997.

[27] K. El-Fakih, H. Yamaguchi, G.v. Bochmann and T. Higashino, "Automatic Derivation of Petri Net Based Distributed Specification with Optimal Allocation of Resources," *Proc. of 15th IEEE Int. Conf. on Automated Software Engineering (ASE'2000)*, pp. 305–308, 2000.

[28] H. Yamaguchi, K. El-Fakih, G.v. Bochmann and T. Higashino, "Protocol Synthesis and Re-Synthesis with Optimal Allocation of Resources Based on Extended Petri Nets" *Journal of Distributed Computing*,16(1), 21-36, 2003.

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

[30] "lp_solve," `ftp://ftp.ics.ele.tue.nl/pub/lp_solve/`

R_{req}	R_{design}	R_{design_fb}	R_{rvw_de}	R_{dcs_de}	R_{code}	R_{test_fb}	$R_{testplan}$	R_{rvw_se1}	R_{dcs_se1}
1	10	5	1	1	30	5	10	1	1
$R_{unittest}$	$R_{testresult}$	R_{rvw_se2}	R_{dcs_se2}	R_{rvw_qa}	R_{dcs_qa}	R_{dcs}	R_{object}	R_{als_qa}	R_{als_de}
10	5	5	1	5	1	1	20	3	3

Table 3: Sizes of Resources

F^1	F^2	F^3	F^4	F^5	F^6	F^7	F^8	F^9	F^{10}	F^{11}	F^{12}	F^{13}	F^{14}	F^{15}	F^{16}	F^{17}
1	3	3	3	3	3	3	3	3	3	3	3	1	1	1	1	1
F^{18}	F^{19}	F^{20}	F^{21}	F^{22}	F^{23}	F^{24}	F^{25}	F^{26}	F^{27}	F^{28}	F^{29}	F^{30}	F^{31}	F^{32}	F^{33}	F^{34}
1	2	2	1	1	3	3	3	3	3	3	3	1	1	1	1	1

Table 4: Firing Frequencies of Transitions

	PE_{mng}	PE_{de}	PE_{se1}	PE_{se2}	PE_{qa}
PE_{mng}	–	10	5	1	5
PE_{de}	10	–	10	5	10
PE_{se1}	5	10	–	1	5
PE_{se2}	1	5	1	–	5
PE_{qa}	5	10	5	5	–

Table 5: Channel Costs

Resource	PE_{mng}	PE_{de}	PE_{se1}	PE_{se2}	PE_{qa}
R_{req}	1	2	2	2	2
R_{design}	4	2	3	3	4
R_{design_fb}	4	2	3	3	4
R_{code}	10	10	30	18	18
R_{object}	5	12	15	15	9
$R_{testplan}$	2	2	2	2	1
$R_{unittest}$	3	3	3	3	1
$R_{testresult}$	3	3	3	3	1
R_{test_fb}	3	3	3	3	1
R_{rvw_de}	5	1	5	5	5
R_{dcs_de}	5	1	5	5	5
R_{rvw_se1}	5	5	1	5	5
R_{dcs_se1}	5	5	1	5	5
R_{rvw_se2}	5	5	5	1	5
R_{dcs_se2}	5	5	5	1	5
R_{rvw_qa}	5	5	5	5	1
R_{dcs_qa}	5	5	5	5	1
R_{dcs}	1	5	5	5	5
R_{als_qa}	5	5	5	5	1
R_{als_de}	5	5	5	5	1

Table 6: Resource Placement Costs