Decomposing Service Definition in Predicate/Transition-Nets for Designing Distributed Systems

Hirozumi Yamaguchi[1], Gregor von Bochmann[2], and Teruo Higashino[1]

[1] Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyamacho, Toyonaka, Osaka 560-8531, Japan
{h-yamagu,higashino}@ist.osaka-u.ac.jp
[2] School of Information Technology and Engineering, University of Ottawa
800 King Edward Avenue, Ottawa, Ontario K1N 6N5, Canada
bochmann@site.uottawa.ca

Abstract. In this paper, we propose a new algorithm for the derivation of a protocol specification in Pr/T-nets, which is the specification of communicating N entities (N can be given), from a given service specification in Pr/T-nets and an allocation of the places of the service specification to the N entities. Our algorithm decomposes each transition of the service specification into a set of communicating Pr/T-subnets running on the N entities. Moreover, for the efficient control of conflict of shared resources, we present a timestamp-based mutual exclusion algorithm and incorporate it into the derivation algorithm.

1 Introduction

Designing highly reliable distributed systems is still a challenging task and a number of techniques have been proposed to reduce design costs and errors. Especially, for the specification phase in the design of distribute systems, there exists a useful design methodology called *protocol derivation* (or *protocol synthesis*, for surveys see [2]). The derivation methods have been used to derive the specification of a distributed algorithm (hereafter called *protocol specification*) automatically from a given specification of services to be provided by the distributed system to its users (called *service specification*). The service specification is written in the form of a centralized model, and does not contain any message exchanges between different physical locations. However, the protocol specification of the cooperating entities' programs, called *protocol entities* (*PE's*), includes the message exchanges between these entities. 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. Recently, many synthesis methods have been proposed which use CCS based models or LOTOS [3–5], FSM based models[6, 10] and Petri net based models [7–9] as service definition languages.

The most popular extension of Petri nets is known as coloured Petri nets (CPN) [1] and predicate/transition-nets (Pr/T-nets) [12] where tokens have values and the firability of transitions can be determined by those values. These models have enough modeling and analytical power to specify, verify and analyze large and practical software systems, communication protocols, control systems and so on [1] and many software tools are provided to help design of these systems using the models. They have been used to model large-scale distributed systems which often include multiple processes running concurrently, such as e-commerce systems where multiple customers may look at, purchase and sell items managed by distributed databases. So what is desired here is to enable designers to define services in these extended Petri nets, and it is also desirable that the protocol specifications are derived from the service definition automatically.

In this paper, we propose a new algorithm for the derivation of a protocol specification in Pr/T-nets, which is the specification of communicating N entities (N can be given), from a given service specification in Pr/T-nets and an allocation of the places of the service specification to the N entities. Our algorithm decomposes each transition of the service specification into a set of communicating Pr/T-subnets running on the N entities. Moreover, for the efficient control of conflict of shared resources, we present a timestamp-based mutual exclusion algorithm and incorporate it into the derivation algorithm. The method has been applied manually to an example specification of a distributed database management system[12] to show the applicability of our algorithm to practical applications. The result is presented in [14].

Our approach is very powerful in the sense that non-restricted Pr/T-nets are allowed to be used for specifying services. Since such Pr/T-nets include complex conflict structures made by choice places, synchronization transitions and multiple tokens with values, we have to consider how to implement those complex conflict structures by multiple entities. In our approach, a timestamp-based mutual exclusion algorithm is presented and elegantly incorporated into the derivation algorithm in order to implement such a structure. Moreover, in order to implement each transition of the service specification handling multiple tokens with values, a new protocol to exchange tokens between entities is introduced. Some existing synthesis methods also allow to treat variables (parameters) in their modeling languages like a CCS-based model with I/O parameters[4] and Petri nets with external variables[7, 9]. However, since these existing methods mainly focus on value exchanges between entities, only simple control flows are allowed (the combination of choices and synchronization involving parameters, which often represents resource conflict, is not allowed). Therefore, the class has been considerably extended from the existing work and as far as we know, no paper has presented synthesis approaches for a first order extension of Petri nets.

This paper is organized as follows. Section 2 gives the definition of Pr/T-nets which we use in this paper and examples of service and protocol specifications. In Section 3 our derivation algorithm is presented and Section 4 enhances the algorithm to handle timestamp-based mutual exclusion. Section 5 gives brief

discussion on the validation and applicability of the algorithm, and Section 6 concludes the paper.

2 Service and Protocol Specifications in Pr/T-Nets

2.1 Predicate/Transition-Nets

We use predicate/transition-nets (Pr/T-nets) for the specification of target systems. In Pr/T-nets, an arc from a place p to a transition t (denoted by (p, t)) has a finite multi-set $m(p, t)$ of tuples of variables. $m(p, t)$ is defined as $m(p, t) = \sum_i k_i V_i$ where k_i is a non-negative integer and V_i is a tuple of variables like $\langle v_1, v_2, ...v_n \rangle$. Each token has a tuple of values C_i, and we say that a token with a tuple C_i of values is *assignable* to V_i *iff* the type of each value in C_i matches that of the corresponding variable in V_i and $|V_i| = |C_i|$. A multi-set of tokens which can be assigned to $m(p, t)$ is called an *assignable set*. Moreover, a transition may have a predicate of variables from the multi-sets on input arcs, called a *condition*. The arc from the transition t to a place p' also has a multi-set (denoted by $m'(t, p')$) whose variables are from the multi-sets on the input arcs of t.

A transition t can fire *iff* there exists an assignable set in each input place and the token values in the assignable sets satisfy the condition of t. If t fires, new tokens are generated and put into the output places according to the multi-sets on the output arcs. For example, in Fig. 1(a), the input arc (P_1, T) has a multi-set "$< i_1, n_1 > + < i_2, n_2 >$" where i_1, n_1, i_2 and n_2 are variables. This means that two tokens which consist of 2-tuples of values are necessary in place P_1 for firing of T. Here, since the assignable sets "(1,"Steve")+(2,"John")" in P_1, "("Steve","Tokyo",10000)+("John","Tokyo",20000)" in P_2 and "2($'e$)" in P_3 satisfy the condition "$(n_1 == m_1)$ &$(n_2 == m_2)$ & $(b_1 == b_2 == Tokyo)$ & $(s == {}'e)$" for firing of T, T can fire by these sets. Note that "$'e$" is a normal token (*i.e.* a token which has no value). Such a token is represented as a black dot in the following figures and is called an *empty value token* hereafter. After the firing of T, new tokens are generated to the output places P_4 and P_5 using those token values. The marking after the firing of T is shown in Fig. 1(b).

2.2 Service Specification

Fig. 1(c) shows a service specification of an example system. The system works as follows. At the initial marking, transition T_u can fire, since there exists an assignable set in each input place of T_u and these assignable sets satisfy the condition of T_u. For example, "100" in P_1, "20" in P_2, "500" in P_3 and "30" in P_5 are such assignable sets that satisfy the condition "$x > y$". Let us assume that these tokens are used for the firing of T_u. If T_u fires, these tokens are removed and new tokens "650" and "$(100, 20, 150)$" are generated in the output places P_3 and P_4, respectively. After that, T_u can still fire using the remaining tokens in its input places. At the same time, T_v can fire now using the new token in P_4.

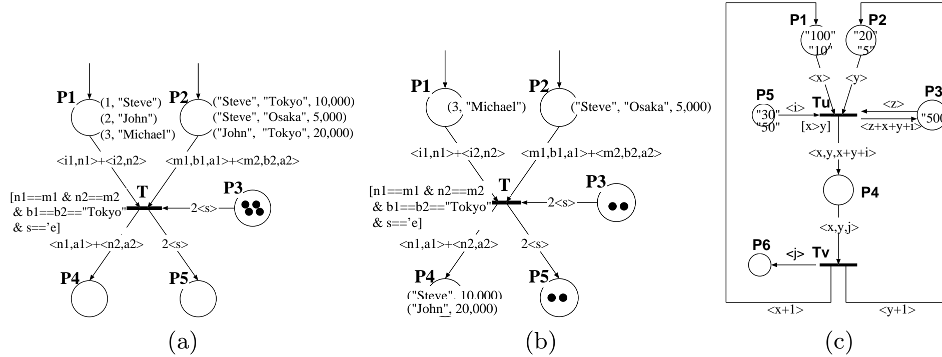

Fig. 1. (a) An example of Pr/T-nets. (b) After firing of transition T. (c) Service specification of an example system

2.3 Protocol Specification

Fig. 2 shows a protocol specification corresponding to the example system in Fig. 1(c). A protocol specification is a set of specifications of N entities communicating with each other asynchronously, called *sites* in this paper. In protocol specifications, we introduce places for modeling asynchronous (and reliable) communication channels (*i.e.* buffers), called *communication places*, like "fusion places" in coloured Petri nets[1]. We assume that two communication places with a common name "$X_{u.ij}$" (X=α or $X = \beta$, explained in the next section) in the Pr/T-nets of two different sites i and j represent the end points (send and receive buffers) of a reliable communication channel from site i to site j. If a token is put on "$X_{u.ij}$" at site i, the token is eventually removed and put onto "$X_{u.ij}$" at site j. Note that u means that these communication places are used with respect to the execution of transition T_u of the service specification. Communication places are represented as dotted circles in the following figures.

In distributed systems, computer resources such as databases are usually distributed over multiple sites. This means that the input and output places of each transition of the service specification may be located on different sites, and we need a protocol to collect/distribute tokens from/to these places to execute the transition in a distributed environment. We determine an efficient protocol in our derivation algorithm where the action of a transition T is implemented by multiple transition sequences on different sites. These transitions are categorized as shown in Table 1.

Now let us consider the firing of T_u in Fig. 1(c) to explain how our protocol works. In the corresponding protocol specification in Fig. 2, the input places P_1 and P_5 are located on site A, P_2 on site B, and P_3 on site C. In our derivation algorithm, one of the sites which have input places of T_u starts the execution of T_u. In this protocol specification, site C starts the execution by making the "start" transition "$T_u.start$" fire, which sends empty value tokens via communication places "$\alpha_{u.ca}$" and "$\alpha_{u.cb}$" to sites A and B, respectively. If site A has

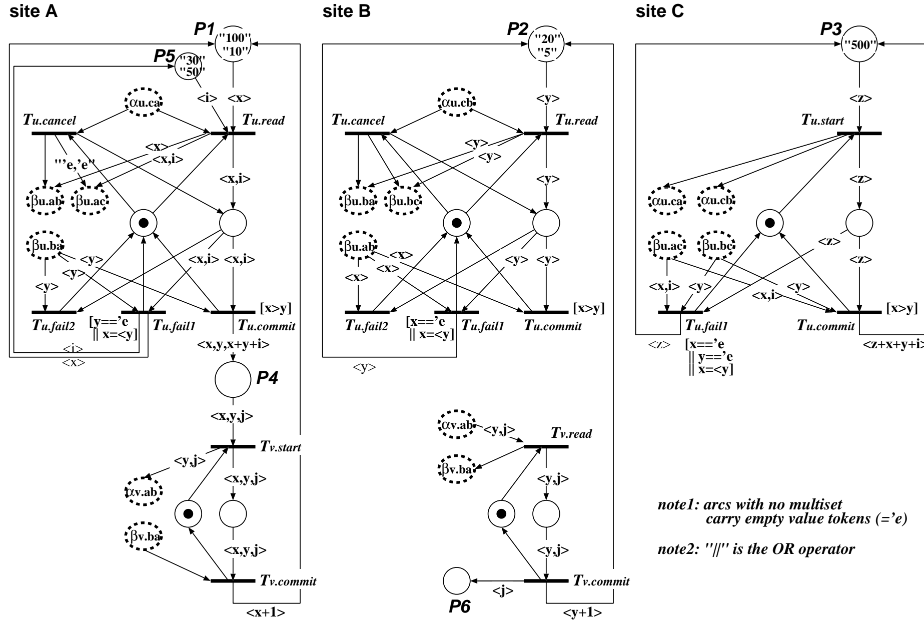

Fig. 2. Protocol specification corresponding to service specification in Fig. 1(c)

tokens (assignable sets) in the input places P_1 and P_5 and if it receives the empty value token from site C, it makes the "read" transition "$T_u.read$" fire, which sends the tokens in P_1 and P_5 to sites B and C via communication places "$\beta_{u.ab}$" and "$\beta_{u.ac}$", respectively. If either P_1 or P_5 does not have an assignable set, the "cancel" transition "$T_u.cancel$" will eventually fire on site A. The firing of the cancel transition means that the execution of T_u will be canceled due to the lack of assignable sets in input places P_1 and P_5 of T_u[3]. In order to let other sites know the fact, empty value tokens are sent to sites B and C by the firing of $T_u.cancel$. Similarly, site B reads a token from P_2 and sends it to both sites A and C. Consequently, every site can examine (i) whether all the input places of T_u have assignable sets or not, and (ii) whether the assignable sets satisfy the condition of T_u or not if (i) is true. If (i) and (ii) are true, the "commit" transition "$T_u.commit$" on each site fires and new tokens are generated on sites A and C to the output places P_4 and P_3, respectively (*i.e.* the execution of T_u has been committed). If either (i) or (ii) does not hold, the "fail" transition $T_u.fail_1$ or $T_u.fail_2$ on each site fires. $T_u.fail_1$ fires in case that an input place at an other site does not have an assignable set or (ii) does not hold, and $T_u.fail_2$ fires in case that an input place at the site itself does have an assignable set. The tokens

[3] In the Petri net formalism, $T_u.cancel$ may fire even when $T_u.read$ can fire. In a practical aspect, it can be easily avoided by prioritizing the firing of $T_u.read$ than that of $T_u.cancel$.

Table 1. Semantics of transitions in protocol specifications

Name	Semantics
$T.start$	initiates the execution of T by reading tokens from the input places of T and sends them to the other sites
$T.read$	(following $T.start$,) reads tokens from the input places of T and sends them to the other sites
$T.cancel$	(following $T.start$,) reads no token from the input places of T and sends empty value tokens to the other sites to let them know that an input place has no assignable set
$T.commit$	commits the execution of T
$T.fail_1$	cancels the execution of T due to the lack of tokens on the other sites or the condition of T
$T.fail_2$	cancels the execution of T due to the lack of tokens on the local site

read from the input places are returned to the input places, *i.e.*, the execution of T_u is aborted.

3 Derivation Algorithm

Our derivation algorithm decomposes a given service specification *Sspec* into a set of N specifications that represent the protocol behaviors of the N sites of the distributed system (called the protocol specification *Pspec*). The derived protocol depends on the given allocation of the places of the service specification to the N sites.

3.1 Overview of the Protocol for Executing a Given Transition

The basic protocol for executing a transition T of *Sspec* over multiple sites is as follows.

Depending on a given allocation of places, we identify the set of sites called *reading sites* which have at least one input place of T, and also the set of sites called *writing sites* which have at least one output place of T. Then we select one of the reading sites as the *primary site*.

At first, at the primary site (say site i), if there exists an assignable set (a set of tokens assignable to the multi-set on an arc) in each input place allocated to site i, site i takes those assignable sets from the places, and sends the token values to the reading and writing sites which need those values. Note that for the reading sites which do not need the token values, the primary site also sends empty value tokens. Consequently, at least one token is sent to each reading site. When a reading site (say site j) receives token(s) from the primary site, site j selects an assignable set from each input place of T allocated to site j (if such a set exists). Then site j sends the tokens to the other reading and writing sites. If such a set does not exist, site j sends empty value tokens to the other reading and writing sites. As a result, the reading and writing sites can examine

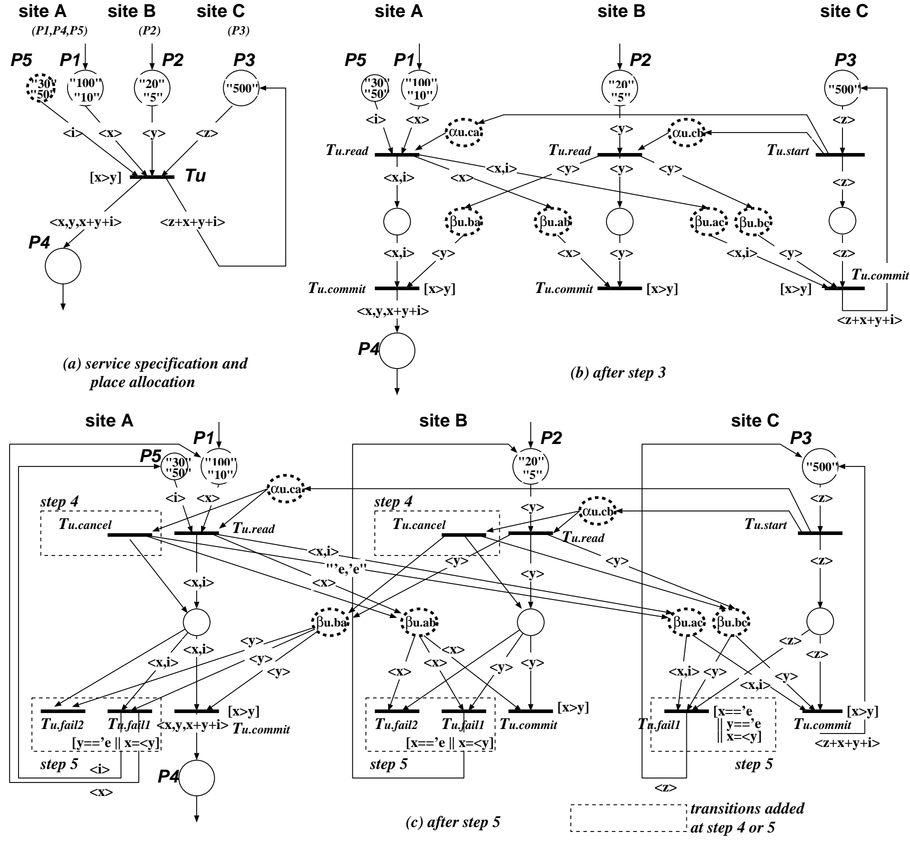

Fig. 3. Derivation algorithm snapshots

(i) whether all the input places of T have assignable sets or not, and (ii) whether the assignable sets satisfy the condition of T or not if (i) is true. If (i) and (ii) are true, reading sites discard the assignable sets and writing sites generate new tokens to the output places of T which are allocated to them. Otherwise the assignable sets which have been acquired by the reading sites are returned to the original input places of T. As stated above, in order to prevent deadlocks due to waiting for tokens where an assignable set does not exist in an input place of a transition, we consider a mechanism to cancel the execution of the transition when at least one of the input places has no assignable set.

3.2 Algorithm Description

According to the protocol explained in the previous section, we present our derivation algorithm in this section. The input to the algorithm is a service specification (denoted by $Sspec$) in the form of a Pr/T-net, the number N of

sites and an allocation of the places to the N sites. The output is a protocol specification (denote by $Pspec$) in the form of Pr/T-nets corresponding to the service specification and the allocation of the places to the N sites.

For better readability, we use the example specification in Fig. 1(c), especially transition T_u. Let us assume that $N = 3$ and P_1, P_4 and P_5 are allocated to site A, P_2 to site B and P_3 to site C as shown in Fig. 3(a). This allocation is the one that was used to derive the protocol specification in Fig. 2. According to this allocation of places, the reading sites of T_u are sites A, B and C, and the writing sites are sites A and C. We have chosen site C as the primary site of T_u.

[Derivation Algorithm]

1. Decompose T into a start transition $T.start$ at the primary site, read transitions $T.read$ and commit transitions $T.commit$ at the reading and writing sites. Note that read transitions at writing sites and commit transitions at reading sites are necessary for consistent execution and this is explained later. Let each commit transition have the condition of T. Then at each reading site, connect the input places of T allocated to the sites to $T.read$ (or $T.start$). Similarly, at each writing site, connect $T.commit$ at the site to the output places of T allocated to the site. Introduce a place between $T.read$ and $T.commit$.

   In the example, site C has $T.start$ and $T.commit$. Sites A and B have pairs of $T.read$ and $T.commit$.
2. From $T.start$ at the primary site (say site i) to $T.read$ at each site (say site j), introduce a communication place "α_{ij}" which carries from site i to site j (a part of) assignable sets read from the input places of T allocated to site i. These are used to generate new tokens or to examine the condition of T. If site j needs no value in those tokens, let the place carry an empty value token. Since the primary site needs to let every site j know that T has been selected to examine its executability, such a token is introduced.
3. Then from $T.read$ (or $T.start$) at each reading site (say site j) to $T.commit$ at each site (say site k), introduce a communication place "β_{jk}" which carries from site j to site k (a part of) assignable sets read from the input places of T allocated to site j. If site k needs no value in those tokens, let the place carry an empty value token. This is necessary to let the other sites know that site j has already read tokens. Also, from $T.read$ at each writing site j to $T.commit$ at the primary site i, introduce a communication place "β_{ji}" which carries an empty value token. This lets the primary site know that the site j has received the token from the primary site.

   Fig. 3(b) shows the specification after this step is applied. From $T_u.start$ on the primary site C to $T_u.read$ on sites A and B, communication places "$\alpha_{u \cdot ca}$" and "$\alpha_{u \cdot cb}$" are introduced, respectively. Moreover, from these $T_u.read$ to $T_u.commit$ on sites A, B and C, "$\beta_{u \cdot ab}$", "$\beta_{u \cdot ac}$", "$\beta_{u \cdot ba}$" and "$\beta_{u \cdot bc}$" are introduced.
4. For each reading site j, add a *cancel* transition $T.cancel$ which has α_{ij} as the input place and the output places of $T.read$ as the output places. Let $T.cancel$ generate an empty value token to each of its output places.

For example, in Fig. 3(c), the *cancel* transition on site A has "$\alpha_{u \cdot ca}$" as the input place. It also has "$\beta_{u \cdot ab}$", "$\beta_{u \cdot ac}$" and a normal place between T_u*.read* and T_u*.commit* as the output places.

5. For each reading or writing site, introduce a *fail* transition $T.fail_1$ which has the same input places as T*.commit*, the input places of T as the output places, and the condition "$\overline{C(T)} \parallel x_1 =='e \parallel x_2 =='e \parallel ...$" where $C(T)$ is the condition of T and x_i is a variable in the multi-sets on the incoming arcs. Therefore this transition fires *iff* the condition of T does not hold or at least one reading site has failed to acquire an assignable set. Moreover, for each reading site except the primary site, introduce a *fail* transition $T.fail_2$ which has the same input places as $T.fail_1$. Let $T.fail_2$ read an empty value token from the place between T*.read* and T*.commit*. Therefore this transition fires *iff* the site itself has failed to acquire an assignable set.

   Fig. 3(c) shows the specification after this step is applied. Since site C is the primary site, it does not have $T_u.fail_2$.
6. For each reading site or writing site, introduce a normal place with an empty value token. Let the place have T*.commit*, $T.fail_1$ and $T.fail_2$ as the input transitions and T*.read* and T*.cancel* as the output transitions. By this place, transition T is not executed by the other assignable sets before the current execution of T is completed.
7. Split each communication place into two places so that the specification of each site can be an independent Pr/T-net.

4 Timestamp-based Mutual Exclusion

In the Petri net formalism, resources of systems are usually modeled as tokens in places. Therefore concurrent transitions which require the same token (resources) share the places and thus form a choice structure.

In Pr/T-nets, a choice structure may be quite complex involving multiple synchronization transitions, multiple choice places and multiple tokens with values. However, it is not easy to realize such a complex structure in a distributed environment where these places are allocated to different sites (distributed choice), since we have to consider deadlock problems and efficiency. A simple example is as follows. Fig. 4(a) shows a service specification and a place allocation where two transitions require tokens in P_b and P_c. P_b and P_c are allocated to sites B and C, respectively. An example execution sequence of the protocol specification corresponding to this service specification and place allocation is shown in Fig. 4(b). In Fig. 4(b), the sites A and D (the primary sites of T_1 and T_2, respectively), sent tokens to sites B and C to let them acquire tokens in P_b and P_c. In this scenario, T_1 could acquire the token in P_b and T_2 could acquire the token in P_c. Therefore, if both T_1 and T_2 wait for the other token which has been acquired by T_2 and T_1, respectively, a deadlock results.

In our derivation algorithm of Section 3, such a deadlock never occurs since we included a cancel mechanism which can be executed when tokens do not exist in input places of transitions. For example, in the above case, the execution of T_2 is

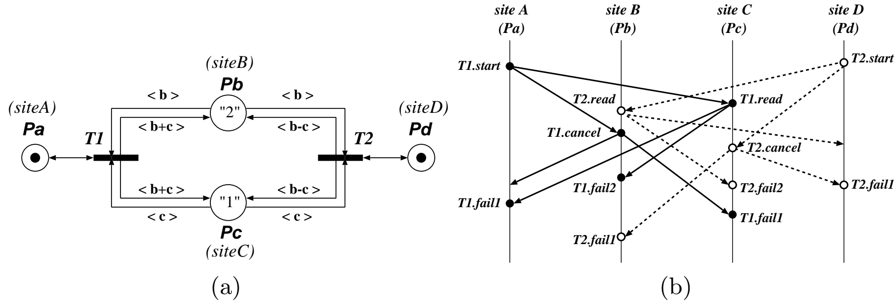

Fig. 4. (a) Service specification and place allocation with distributed choice places. (b) Timing chart (the execution of both T_1 and T_2 failed)

canceled by "T_2.*cancel*" transition on site B, and the execution of T_1 is canceled by the "T_1.*cancel*" transition on site C. However, this scenario may be repeated until either T_1 or T_2 has the chance to acquire tokens in both P_b and P_c. This will result, in general, in a number of trial and errors in acquiring resources. Moreover, there may be the case that one transition repeatedly acquires the tokens and the other is blocked.

In order to realize efficient control, in this section, we introduce a timestamp-based control mechanism. However, unlike usual cases such as concurrent transaction control in database systems[13], concurrent transitions require multiple resources distributed over multiple sites in our case, and the problem is much more complicated. Therefore, we have to design a new protocol exchanging timestamps and resources to be suitable for our derivation algorithm that uses high-level Petri net formalism.

4.1 Preliminaries

We formally define the structures to which our timestamp-based control should be applied.

For a place or transition s, let $\bullet s$ ($s\bullet$) denote the set of input (output) transitions or places of s. A set T of transitions is said to be a *conflict transition set iff* $|\bigcap_{t \in T} \bullet t| > 1$. The place set $\bigcap_{t \in T} \bullet t$ is called a *conflict place set*. This means that the transitions in a conflict transition set share more than one input place. Here, places in a conflict place set that never lose their tokens by firing of transitions in the conflict transition set are called *persistent places*.

We apply our timestamp-based mutual exclusion control to persistent places that belong to the same conflict place set and are allocated to different sites, and their output transitions (*i.e.* they belong to a conflict transition set). For example, places P_b and P_c in the service specification of Fig. 4(a) are in a conflict place set since they are shared by T_1 and T_2 which form a conflict transition set, and also P_b and P_c are allocated to different sites. The reason why we control access to places in a conflict place set is that such places may be accessed by

Table 2. Classification of a pair of a persistent place p and its output transition t

type	condition
`RW-persistent`	t reads tokens from p and writes new tokens to p. t uses the token values to generate tokens to (some other) output places
`RO-persistent`	t reads tokens from p and writes back the same tokens. t uses the token values to generate tokens to (some other) output places
`WO-persistent`	t reads tokens from p and writes new tokens to p. t does not use the token values to generate any token.

multiple transitions each of which represents the synchronization of these places, *i.e.*, the transition requires tokens in all the places. This results in conflict situation as exemplified above. Moreover, the reason why we focus only on persistent places is that in such a place, tokens will be returned and transitions can wait for tokens to be back even if some other transitions currently use them.

For the algorithm in the next section, we classify the pairs of a persistent place P and one of its output transitions T as shown in Table 2, (a) `RW-persistent`, (b) `RO-persistent` and (c) `WO-persistent`. "`RW`" indicates that T refers and modifies token values, "`RO`" indicates that T only refers token values and "`WO`" indicates that T never refers but modifies token values. Examples are shown in Fig. 5. In Fig. 5(a), T reads a token (it is assigned to variable x) and writes a new token $x + y$ to P. On the other hand, in Fig. 5(b), x is used to generate tokens, however the token in P is not modified (the token is returned with the same value). In Fig. 5(c), x is not used to generate new tokens, and token y is returned instead of x.

4.2 Algorithm for Adding Timestamp-based Mutual Exclusion Mechanism

We assume that each site has a clock that is synchronized with the clocks of the other sites[4]. The protocol presented in this section adds a pair of timestamps (read and write timestamps) to each token in persistent places in a conflict place sets, and determines the access order of transitions in a conflict transition set based on the time when the primary sites tried to execute the transitions. This enables the transitions in the conflict transition set to wait for the assignable sets (tokens) to be returned to the persistent places without causing deadlocks. The detailed description of our timestamp-based solution is presented below.

[Algorithm for Adding Timestamp-based Mutual Exclusion Mechanism]

[4] We use these clocks just to determine the total order of the execution of transitions in a conflict transition set. In this sense, these clocks are not needed to be synchronized precisely.

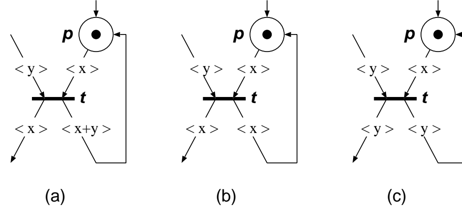

Fig. 5. Examples of a persistent place and a transition pairs. (a) `RW-persistent`, (b) `RO-persistent` and (c) `WO-persistent`

Let T be a conflict transition set and P be the conflict place set corresponding to T. Let us assume that at least two persistent places in P are allocated to different sites. This algorithm is applied to the protocol specification which is obtained by the algorithm of Section 3.

1. Let each token c in a persistent place have two variables $R\text{-}TS(c)$ (a read timestamp) and $W\text{-}TS(c)$ (a write timestamp). Let the values of those variables at the initial marking be zero.
2. Let the primary site (say site i) of $t_u \in T$ generate a timestamp ts_u on $t_u.start$ and include it to tokens sent to the other reading sites which have places in P.
3. If a reading site (say site j) has a persistent place p,
   - Build the conjunction of timestamp conditions (see Table 3) of tokens in an assignable set taken from p. Then add this condition to the current condition of $t_u.read$. Moreover, add its negation to the current condition of $t_u.cancel$. Then let $t_u.read$ generate *dummy tokens* from the tokens in the assignable set. A dummy token of the token c has an empty value and a pair of timestamps obtained by updating the timestamps of c according to the rules in Table 3.
   - Let $t_u.commit$ read the dummy tokens corresponding to the tokens which site j has read and kept, for the execution of t_u. Also let $t_u.commit$ change the empty value in the dummy token to the new value.

The idea is that we first classify the types of t_u based on how t_u modifies tokens in the persistent place p. Then we derive the timestamp condition like "if t_u only reads the token (read-only), the read operation should be later than the last write operation" as shown in Table 3. Note that the fourth row in the table is known as *Thomas's write rule*[13]. The rule allows a write-only operation request issued earlier than the last write operation to be "ignored" if it has been issued after the last read operation has done. If t_u's timestamp satisfies the condition in the table, t_u has the right to acquire an assignable set even though there is currently no assignable set in p (t_u can wait for tokens to be returned without deadlock).

It would be much better to see an example to understand how the mutual exclusion works. Fig. 6 shows the protocol specification which corresponds to

Table 3. Condition for timestamp of t_u to acquire token c and update rules of c's timestamps

type of (p, t_u)	timestamp condition for ts_u	update rules
RW-persistent	(R-TS(c) $< ts_u$) and (W-TS(c) $< ts_u$)	R-TS(c):=W-TS(c):= ts_u
RO-persistent	W-TS(c) $< ts_u$	R-TS(c):= ts_u
WO-persistent	(R-TS(c) $< ts_u$) and (W-TS(c) $< ts_u$)	W-TS(c):= ts_u
WO-persistent	R-TS(c) $< ts_u <$ W-TS(c)	–

the service specification in Fig. 4(a), and Fig. 7 shows its example timing charts. Fig. 7(a) shows the following scenario. In the service specification in Fig. 4(a), P_b and P_c are persistent places in a conflict place set and T_1 and T_2 are in the conflict transition set corresponding to the conflict place set. In the protocol specification, the primary sites of T_1 and T_2 are sites A and D, respectively, and P_b and P_c are allocated to site B and site C, respectively.

- The timestamp of T_2 ($=ts_2$) was newer than that of T_1 ($=ts_1$) (*i.e.* $ts_2 > ts_1$).
- The timestamp ts_1 arrived at site C after the arrival of ts_2. Since $ts_2 > ts_1$ and T_2 has already kept the token $< c >$ in P_c, the execution of T_1 was canceled at the site C, and empty value tokens indicating that "T_1 was canceled at site C" were sent to sites A and B.
- On the other hand, at site B, the arrival of ts_1 was earlier than ts_2. T_1 has kept the token $< b >$ and waited for the notification of from site C. However, the tokens from site C indicated that the execution of T_1 had been canceled at site C. Then site B immediately canceled the execution of T_1 and released the token $< b >$.
- While T_1 was keeping the token $< b >$, the timestamp ts_2 arrived at site B. Even though P_b had no token at that time, T_2 waits for $< b >$ to be released since P_b is a persistent place and ts_2 is newer than ts_1.
- After the execution of T_1 is canceled at site B, T_2 could keep the token $< b >$ and send the token values to sites A and C. At this point, the sites A, B and C had acquired tokens in the input places P_b, P_c and P_d of T_2 and thus had been ready to examine the condition of T_2.
- Let us assume that the token values satisfy the condition of T_2. Then the execution of T_2 is finished.

Fig. 7(b) shows another scenario where T_1 and T_2 were accepted in this order. However, because of space limitation, the explanation is omitted here.

5 Validation and Example

The validation of the derivation algorithm is an important issue to confirm the correctness of the proposed method. To validate the algorithm, we have to show that the derived protocol is equivalent to the given service.

Moreover, a practical example is mandatory to show the applicability of the method. As an example, we focus on a distributed database management

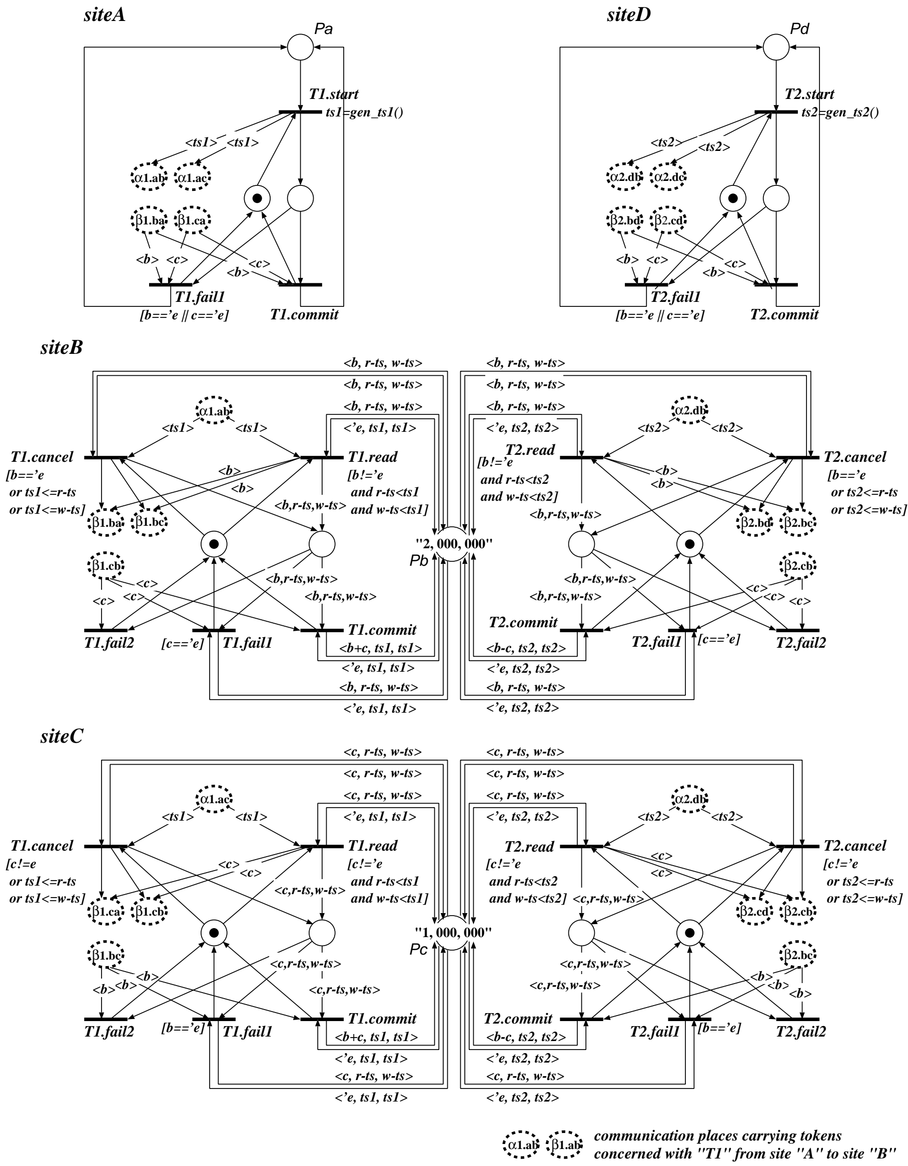

Fig. 6. Protocol specification corresponding to service specification and place allocation in Fig. 4(a)

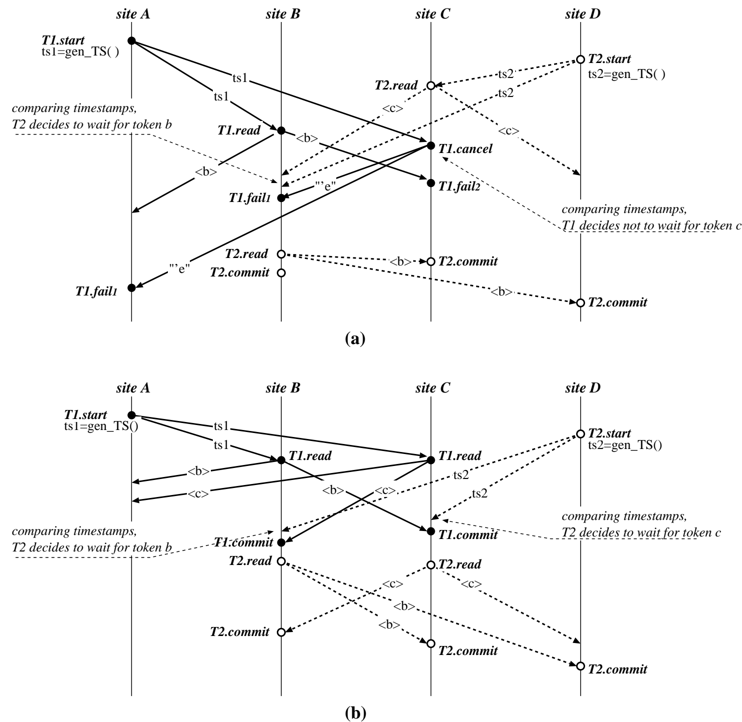

Fig. 7. Timing charts for protocol specification in Fig. 6

system (DBMS) described as Pr/T-net in Ref.[12], which is one of the typical systems to show the expressive and analytical power of Pr/T-nets. Although this example is rather traditional, we think that the concept can be applied to recent applications which use databases such as information management service systems on the web implemented by collaborative web servers.

Because of space limitation, these issues are given in a different document. Readers may refer to Ref. [14] for details.

6 Conclusion

We have proposed a protocol synthesis technique for systems modeled as Pr/T-nets (predicate/transition-nets), a first-order extension of Petri nets. Our technique is based on a top-down approach where a service requirement is defined in the form of a Pr/T-net like a centralized program, and then it is decomposed into communicating components located on different sites which, together, provide the required service. Our approach is original in the sense that it allows non-restricted class of Pr/T-nets for specifying the services. This is a very important feature to model recent distributed collaborative systems since they often include

multiple (and to be distinguished) processes such as mobile users. Moreover, we have presented a mutual exclusion algorithm for a distributed environment based on timestamps, which may improve the performance of the distributed systems. The applicability of our algorithm has been shown by an example.

Deriving protocol specification by hand is very complex and we truly need tool supports. Therefore we have a plan to develop a derivation tool which co-works with existing designing tools such as Design/CPN, and this is part of our future work.

References

1. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Springer-Verlag (1997)
2. Saleh, K.: Synthesis of Communication Protocols: an Annotated Bibliography. ACM SIGCOMM Computer Communication Review, Vol. 26, No. 5 (1996) 40–59
3. Erdogmus, H., Johnston, R.: On the Specification and Synthesis of Communicating Processes. IEEE Trans. on Software Engineering, Vol. SE-16, No. 12 (1990)
4. Gotzhein, R., Bochmann, G. v.: Deriving Protocol Specifications from Service Specifications Including Parameters. ACM Trans. on Computer Systems, Vol. 8, No. 4 (1990) 255–283
5. Kant, C., Higashino, T., Bochmann, G. v.: Deriving Protocol Specifications from Service Specifications Written in LOTOS. Distributed Computing, Vol. 10, No. 1 (1996) 29–47
6. Chu, P. -Y. M., Liu, M. T.: Protocol Synthesis in a State-transition Model. Proc. of COMPSAC '88 (1988) 505–512
7. Kahlouche, H., Girardot, J. J.: A Stepwise Requirement Based Approach for Synthesizing Protocol Specifications in an Interpreted Petri Net Model. Proc. of INFOCOM '96 (1996) 1165–1173
8. Al-Dallal, A., Saleh, K.: Protocol Synthesis Using the Petri Net Model. Prof. of 9th Int. Conf. on Parallel and Distributed Computing and Systems (PDCS'97) (1997)
9. Yamaguchi, H., El-Fakih, K., Bochmann, G.v., Higashino, T.: Protocol Synthesis and Re-synthesis with Optimal Allocation of Resources Based on Extended Petri Nets. Distributed Computing, Vol. 16, No. 1 (2003) 21–35
10. Khoumsi, A., Saleh, K.: Two Formal Methods for the Synthesis of Discrete Event Systems, Computer Networks and ISDN Systems, Vol. 29, No. 7 (1997) 759–780
11. Kapus-Koler, M.: Deriving Protocol Specifications from Service Specifications with Heterogeneous Timing Requirements. Proc. of 1991 Int. Conf. on Software Engineering for Real Time Systems (1991) 266–270
12. Voss, K.: Using Predicate/Transition-Nets to Model and Analyze Distributed Database Systems. IEEE Trans. on Software Engineering, Vol. 6, No. 6 (1980) 539–544
13. Korth, H. F., Silberschatz, A.: Database System Concepts. McGraw-Hill (1991)
14. Yamaguchi, H., Bochmann, G. v., Higashino, T.: Decomposing Service Definition in Predicate/Transition-Nets for Designing Distributed Systems. Online Document, http://www-tani.ist.osaka-u.ac.jp/techreport-e.html (2003)