An Efficient Method for Synthesizing Optimized Protocol Converters*

Z.P.Tao, G.v. Bochmann and R. Dssouli
Université de Montréal, Département d'Informatique et de
Recherche Operationnelle, C.P. 6128, Succ. ¨A¨
Montréal, P.Q., Canada H3C-3J7
Email: {tao, bochmann, dssouli}@iro.umontreal.ca

Abstract: *We propose an efficient algorithm for constructing optimized protocol converters to achieve interoperability between heterogeneous computer networks. This algorithm first derives constraints from the two protocols to be converted, and imposes the constraints to channel specifications, thus removing message sequences of the channel specifications that do not contribute to system progress. Then, an optimized converter is generated from a given service specification, the two protocol specifications and the modified channel specifications. The reduction relation [3] [9] is used to compare the service specification and the constructed internetworking system. Compared with related works, our method has three advantages: 1) it generates an optimized converter; 2) the service specification may be nondeterministic; 3) it may need less computation.*

1. Introduction

One of the difficulties that arise in interconnecting heterogeneous data networks is the problem of protocol mismatches [5] - incompatible protocols are used in heterogeneous networks. A typical example is the interconnection of wireless data networks with WAN or MAN. Communication gateways are widely used for solving this problem; they contain an important component - a protocol converter. The problem of protocol conversion can be explained informally as follows. Consider two different protocols A = (A1, A2) and B = (B1, B2) (Fig. 1). Suppose the two protocols provide similar services, but differ in

* This work is supported by the IDACOM-NSERC-CWARC Industrial Research Chair on Communication Protocols. This paper is an extended version of the paper presented at the conference IC3N'95, Las Vegas, USA, Sept. 1995.

certain details, and we want A1 to communicate with B2 through a protocol converter C. The converter C receives messages from one protocol, interprets them, and delivers appropriate messages to the other protocol in a well-defined order such that A1 communicates with C as if A1 communicated with A2, and B2 communicates with C as if B2 communicated with B1. The protocols and the converter together form an internetworking system and provide the required services specified by Sc as shown in Fig.2(a). An interconnection between A2 and B1 could be defined similarly.

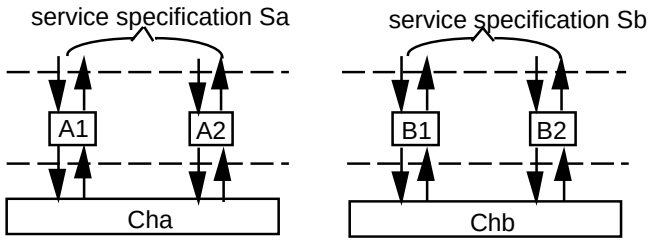


Fig.1 two protocols A and B

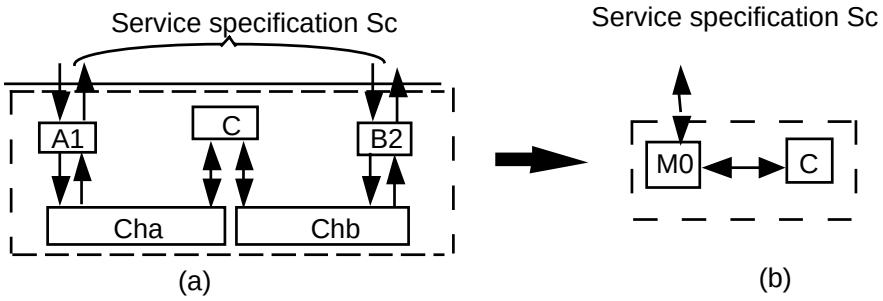


Fig.2

Protocol conversion is a complex problem since multiple protocols are considered. It is difficult to design a correct protocol converter by informal, heuristic methods. A formal approach is a reasonable choice in this area, which may minimize design errors and simplify design procedures. Several formal methods have been proposed for protocol conversion in the last several years. These methods can roughly be classified into the following two classes: the bottom-up methods and the top-down methods.

A bottom-up method begins with analyzing heuristically the low level functions of the protocols to be converted in order to find out PDU level constraints (for example message mapping relations between protocols), and these constraints are used to construct a converter from the Cartesian cross product of A2 and B1 [14] [17] [19] [20] [21]. The common limitations of the bottom-up methods are:

1) A message mapping set is required which can only be obtained heuristically.

2) It is difficult to validate the correctness of a given message set.

3) No service requirement of the interworking system is explicitly used, therefore the generated converter needs to be verified against a given service specification.

A top-down method explicitly uses a service specification of the interworking system as the semantic constraint. The main methods are outlined below. In [2], the concept of a service adapter is proposed for the concatenation of different communication services to be interconnected. A service adapter receives a service primitive, from one protocol, interprets it and sends it to the other protocol. The automatic construction of a PDU-level protocol converter from the given protocol specifications and a service adapter is described in [1]. In [15], a two-stage approach is developed to derive protocol converter. In the first stage, a service adapter from the service specifications of the two protocols is constructed by using the method proposed in [11]. In the second phase, a PDU level protocol converter is constructed by directly composing the service adapter and the underlying protocol specifications. Okumura discussed under what conditions the constructed internetworking system will inherit the properties from the original protocols. It is possible that the converter constructed in this way may contain some states and transitions that are never executed. An efficient algorithm was presented in [6] to remove the superfluous states and transitions. The basic idea is to remove from the underlying protocol entities composed with the service adapter those service primitives (and related states) that are unmatched with the service adapter, and those that can be reached *only* from the unmatched service primitives; then the algorithm computes and retains the strongly connected components starting with the initial state, and discards the rest of the machine. The disadvantage of the methods discussed above is that there may not exists a service adapter for two given protocols even if a PDU level converter does exist. Therefore, the application of the methods is limited.

Calvert and Lam proposed a top-down method [4] [7], which uses a safety property and a progress property to guarantee the correctness of the converter. The algorithm is divided into two phases. In the first phase, a set of states and transitions is constructed inductively by searching the giving protocol entities and the service specification under the safety constraint. The result is a specification with the maximum trace set satisfying the safety property. In the second phase, the states and transitions in the specification that violate the progress property are iteratively removed. If the final specification is not empty after the algorithm terminates, then the converter is obtained. The advantage of this method is that it does not have the limitation of the methods proposed in [6] and [15], i.e., the algorithm will generate a protocol converter if it exists. However, there are also some limitations:

1) The converter may contain superfluous states and transitions that do not contribute to the system progress, and may be harmful for system performance. An example given in [4] and [7] is shown in

Fig.3, where the states in the dotted box are superfluous. The converter can send back an unnecessary acknowledgment to a protocol even after receiving a data message correctly. If these states and transitions are not removed, the system performance will certainly be worse than optimal. Calvert suggested that these states and transitions should be better removed by hand (rather than by algorithm). However, it is not clear how to do so in general.

2) The service specification Sc must be deterministic.

3) The computation of the algorithm is complex.

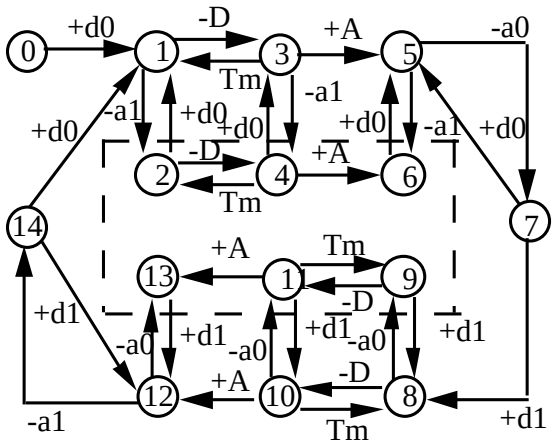


Fig.3 An unoptimized converter

In summary, the protocol conversion methods reported in literature are far from satisfactory due to the limitations discussed above. Nevertheless, we favor top-down methods because of the following reasons:

1) The service specification is used explicitly; it is not necessary to validate the internetworking system against its service specification after the converter is generated.

2) It does not require the establishment of a message mapping relation between protocols in advance, the most difficult part of the bottom-up methods.

In this paper, we propose an approach trying to overcome some of the limitations of top-down methods. We observed that the channel specifications may contain message sequences not contributing to system progress. The top-down methods proposed in [4] and [7] do not remove these unnecessary behaviours when a converter is derived from the channel specifications and protocol specifications. We can get design constraints from the protocol entities, and impose them to the channel specifications; then the obtained converter will have no unnecessary behaviour. Therefore, our approach first derives

constraints from the original protocols and imposes them on the channel specifications, thus removing the message sequences of the channel specifications that do not contribute to system progress. Then an optimized converter is generated from a given service specification, the two protocol specifications and the modified channel specifications. Since the unecessary states and transitions are removed from the first step, this approach may reduce computation. We use the reduction relation [3] [9] to compare the resulting interworking system and the service specification. This allows the treatment of nondeterministic service specifications. A "reduced" system must foresee the same interaction sequences, but may present less non-deterministic choice. Compared with related works, our method has three advantages:

1) It generates an optimized converter.

2) The service specification may be nondeterministic.

3) It may need less computation.

The paper is organized as follows. Section 2 will introduce preliminary definitions and formalize the protocol conversion problem. In Section 3, some theoretical aspects of our approach will be presented. In section 4, an algorithm for protocol conversion will be developed, and its correctness will be proved in Appendix. An example is given to show the application of the algorithm.

2. Definition of the Problem

2.1. General Definitions

We will use the model of Finite Labelled Transition Systems (FLTS) [13]. An FLTS is defined as follows.

Definition 1 (FLTS) [13]**:** A non deterministic FLTS M is a four-tuple M = (Q, Σ, δ, q0), where

Q is a finite set of states.

Σ is a set of observable events.

δ is a transition function, δ: (Σ≈{τ})∞Q∅2^Q with τ denoting an internal event, which defines a set S $\prod 2^Q$ of next states when an event e Σ≈ {τ} occurs in the current state q Q. When the FLTS is at state q, we say that the transition to q' with event e, written q– e∅ q', is enabled if q' δ(q, e), where q, q' Q, and eΣ≈{τ}. The transition q–e∅q' is said an incoming transition of q' and an outgoing transition of q.

q0Q is the initial state.

When the transition function is defined by δ: Σ∞Q∅Q, then the FLTS is said to be *deterministic*.

The behaviour of an FLTS is characterized by the set Σ of its events and the order in which they can be executed. The set Σ contains all externally observable events. Each event is considered atomic, i.e., no

other event can overlap with an atomic event during the time interval from its initiation to its termination. A system often contains two or more subsystems modelled by FLTSs. When two or more FLTSs are executed in parallel, they interact by requiring that an event in one FLTS be executed at the same time with another event in at least one other FLTS. If two or more FLTSs interact in this way, we say that the events required to be executed jointly are *directly coupled*, and together form an atomic event (also called rendezvous or interaction). To avoid confusion, we assume that all uncoupled events in the interacting FLTSs have a unique name. The interactions between the FLTSs can be specified by assigning the same name to events that are directly coupled. It is reasonable to make this assumption since we can rename uncoupled events to make them unique among the set of events executed by each FLTSs.

Table 1: Notations

Notation	**Meaning**
$q-e\varnothing$	$\exists q'$, such that $q - e \varnothing q'$.
$q - e \not\rightarrow$	$(q - e \varnothing)$, i.e., there is no state q' such that $q - e \varnothing q'$.
$q-\tau^k\varnothing q'$	An FLTS may engage in a sequence of k internal events, and after doing so, enters q'.
$q- \sigma\varnothing q_n$	For a sequence of events $\sigma=e_1e_2...e_n$ $(e_i\Sigma\approx\{\tau\})$, there exist states q0, q_1, ..., q_n such that $q_0- e_1\varnothing q_1$, $q_1- e_2\varnothing q_2$, ..., $q_{n-1}- e_n\varnothing q_n$. σ is called an execution sequence.
$q -\sigma\varnothing$	There exist a state qn such that $q - \sigma \varnothing q_n$
E(M)	The set of all execution sequences of M, e.g., $E(M) = \{\sigma\|q0-\sigma\varnothing\}$
$q = >_{\Sigma o} q'$	There is an execution sequence $\sigma(\Sigma-\Sigma o\approx\{\tau\})^*$ such that $q-\sigma\varnothing q'$, where $\Sigma o\prod\Sigma$.
$q = e >_{\Sigma o} q'$	There are two execution sequences $\sigma1$, $\sigma2(\Sigma-\Sigma o\approx\{\tau\})^*$ and two states q1 and q2 such that $q-\sigma1\varnothing q1$, $q1-e\varnothing q2$ and $q2-\sigma2\varnothing q'$, in short $q1-\sigma1e\sigma2\varnothing q'$.
$q = t >_{\Sigma o} q'$	For a sequence of events t = e1 ...en, where $ei\Sigma o$, $\exists\sigma0$, ..., σn $(\Sigma-\Sigma o\approx\{\tau\})^*$ such that $q - \sigma0$ e1 σ1 e2 ... en $\sigma n \varnothing q'$. t is called a trace.

$q = t \not\Rightarrow_{\Sigma o}$	™q', such that $q = t >_{\Sigma o} q'$.
$q = t >_{\Sigma o}$	$\exists q'$, such that $q = t >_{\Sigma o} q'$.
$Tr_{\Sigma o}(M)$	Is the set of traces of an FLTS M, i.e., $Tr_{\Sigma o}(M) = \{t \mid q0 = t >_{\Sigma o}\}$ for $\Sigma o \prod \Sigma$.
$p(S)$	For a given set S, $p(S)$ denotes a power set of S, i.e., set of subsets of S.
q *after* $_{\Sigma o}$ t	A state q in an FLTS M satisfying $q0 = t >_{\Sigma o} q$ for a trace t and initial state q0.
$Ref_{\Sigma o}(M, q)$	Is the refusal set of an FLTS M at state q for Σo, i.e., $Ref_{\Sigma o}(M, q) = \{e \mid q = e \not\Rightarrow_{\Sigma o}$ and $e \Sigma o\}$.

Note that in Table 1, given an FLTS M, $\Sigma o \prod \Sigma$ is the set of "observable events", which are those events that interact with the environment, possibly another FLTS. If $\Sigma o = \Sigma$, then the trace definition is the same as the one defined in [13].

For system analysis, we need to compose two FLTSs. Since directly coupled events may be invisible by the environment, we may model them by internal events for convenience. In this way we define a *coupled product* of two FLTSs as follows.

Definition 2 (*Coupled product*)**:** A *coupled product* M1 || M2 of two FLTSs M1 = (Q1, Σ1, δ1, p0) and M2 = (Q2, Σ2, δ2, q0) is an FLTS M = (Q, Σ, δp, (p0, q0)) such that:

Q is a subset of Q1∞ Q2; each element is of the form (p, q), where p Q1, q Q2;

$\Sigma = (\Sigma 1 \approx \Sigma 2) - (\Sigma 1 \leftrightarrow \Sigma 2)$;

(p0, q0) Q is the initial state;

δp is the transition function defined on Q such that for p, p' Q1, q, q' Q2:

1) $(p, q) - e_1 \varnothing (p', q)$ if $p - e_1 \varnothing p'$ and $e_1(\Sigma 1 - \Sigma 2) \approx \{\tau\}$;
2) $(p, q) - e_1 \varnothing (p, q')$ if $q - e_2 \varnothing q'$ and $e_2(\Sigma 2 - \Sigma 1) \approx \{\tau\}$;
3) $(p, q) - \tau \varnothing (p', q')$ if $p - \mu \varnothing p'$, $q - \mu \varnothing q'$ and $\mu \Sigma 1 \leftrightarrow \Sigma 2$
4) for other cases, no transition is defined.

In some cases, we need to compose two FLTSs that do not directly interact with each other. This can be done by computing the so-called *Cartesian cross product* , written M1∞M2, which is identical to the coupled product M1 || M2 in the case that $\Sigma 1 \leftrightarrow \Sigma 2 = \phi$.

Definition 3 (*# product*)**:** A *# product* M1 # M2 of two FLTSs M1 = (Q1, Σ1, δ1, p0) and M2 = (Q2, Σ2, δ2, q0) is an FLTS M = (Q, Σ, δp, (p0, q0)) where:

Q is a subset of Q1∞ Q2;

$\Sigma = \Sigma 1 \approx \Sigma 2$ is the set of events;

(p0, q0) is the initial state;

δp is the transition relation defined on Q such that for p, p' Q1 and q, q' Q2:

1) $(p, q) - e_1 \varnothing (p', q)$ if $p - e_1 \varnothing p'$ and $e_1 \ (\Sigma 1 - \Sigma 2) \approx \{\tau\}$;
2) $(p, q) - e_2 \varnothing (p, q')$ if $q - e_2 \varnothing q'$ and $e_2 \ (\Sigma 2 - \Sigma 1) \approx \{\tau\}$;
3) $(p, q) - \mu \varnothing (p', q')$ if $p - \mu \varnothing p'$ and $q - \mu \varnothing q'$ with $\mu \ \Sigma 1 \leftrightarrow \Sigma 2$.
4) for other cases, no transition is defined.

Compared with the coupled product, the only difference is that each directly coupled event $\mu \ \Sigma 1 \leftrightarrow \Sigma 2$ in # product is still observable, rather than represented as an internal event.

Definition 4 *(strong bisimulation relation)[12]:* A binary relation ξ on states is a strong bisimulation if for each $<p, q> \ \xi$ and each event $e \ \Sigma \approx \{\tau\}$, the following two conditions hold:

1) Whenever $p - e \varnothing p'$ then for some q', $q - e \varnothing q'$ and $(p', q') \ \xi$;
2) Whenever $q - e \varnothing q'$ then for some p', $p - e \varnothing p'$ and $(p', q') \ \xi$;

We write $p \cong q$ if $(p, q) \ \xi$.

Intuitively, a bisimulation may be thought of as a matching between states that has the property that if two states are matched then each α-derivative (the execution sequence starting from the state) of each state must be matched by some α-derivative of the other.

Definition 5 *(strong bisimulation equivalence of two FLTS)[12]:* Given two FLTSs M1 = (Q1, $\Sigma 1$, $\delta 1$, p0) and M2 = (Q2, $\Sigma 2$, $\delta 2$, q0), M1 and M2 are strong bisimulation equivalent if there is a strong bisimulation relation ξ which contains $<p0, q0>$, written $M1 \cong M2$.

Definition 6*(Reduction relation) :* Given two FLTSs M1 = (Q1, $\Sigma 1$, $\delta 1$, p0) and M2 = (Q2, $\Sigma 2$, $\delta 2$, q0), M1 and M2 satisfy reduction relation for $\Sigma o \prod \Sigma 1$, written $M1 \ _{\Sigma o} \ M2$, if the following conditions are satisfied:

1) $Tr_{\Sigma o}(M1) \prod Tr_{\Sigma o}(M2)$.
2) For any $t \prod Tr_{\Sigma o}(M1) \leftrightarrow Tr_{\Sigma o}(M2)$ and any q after t in M1, there is a state p after t in M2 such that $Ref_{\Sigma o}(M1, q) \prod Ref_{\Sigma o}(M2, p)$.

This definition is similar to the reduction relation defined in [3] and [9]. The difference is that we define the traces and rejected events by Σo, instead of $\Sigma 1$ and $\Sigma 2$. This definition can be explained by two concepts: the first is the *safety property* - the set of all traces of M1 is limited to the set of traces of M2; the second is the *progress property*: placed in any environment whose interface with M1 or M2 is defined by Σo, an event is rejected by M1 may also be rejected by M2. The progress property implies that M1 can not deadlock when M2 can not deadlock, i.e., M1 deadlocks less often than M2. $M1 \ _{\Sigma o} \ M2$ iff M1 and M2 satisfy the safety property and the progress property.

Definition 7 *(Submachine*): An FLTS M' = (Q', Σ', δ', q0') is a submachine of another FLTS M = (Q, Σ, δ, q0) if (a) q0' = q0; (b) Q' ∏ Q; (c) Σ' ∏ Σ; and (d) δ' ∏ δ.

2.2. Formal Definition of Protocol Conversion

The protocol conversion problem informally explained in the Introduction can be formally defined as follows: to find a definition of the behaviour of a converter C such that the following expression is satisfied:

$$A1||Cha||C \,||\, Chb||B2 \;_{\Sigma s}\; Sc$$

where Cha and Chb denote the channels between the protocol entities as shown in Fig.1, and Σs is the set of observable events of Sc. Since A1, Cha, Chb, and B2 are given, this expression can be represented by $M0||C \;_{\Sigma s}\; Sc$, where M0 = (A1||Cha)6(Chb||B2) as shown in Fig.2(b).

In addition, the converter should satisfy the following requirements:

1) The constructed converter C should have no superfluous transitions and states.
2) The converter C should work in such a way that A1 communicates with C as if A1 communicated with A2, and B2 communicates with C as if B2 communicated with B1. Therefore, the following condition should be satisfied at the service interface:

$$Sc_{\Sigma a1}\; A1||Cha||A2 \text{ and } Sc_{\Sigma b2}\; B1||Chb||B2$$

Definition 8 (*maximum solution*): Given M0 = (Q, Σ0, δ, p0) and Sc = (Q_S, Σ_S, δ_S, q0), a converter C is a maximum solution if for any other converter C' satisfying $M0||C' \;_{\Sigma s}\; Sc$, we have $Tr_{\Sigma c}(C') \,/\, Tr_{\Sigma c}(C)$, where Σc= Σ0 - Σs is the set of events of C and C'.

A maximum solution is not necessarily exactly what we want, since it may contain superfluous states and transitions. Protocol conversion often deals with well designed protocols which have already been used in practical networks. It is reasonable to assume that these protocols have no superfluous states and transitions, when considered alone.

To construct an optimized protocol converter, our basic argument is that if the converter C does not do more than A2 and B1 can do, i.e., $Tr_{\bullet a2}(C) \,/\, Tr_{\Sigma c}(A2)$ and $Tr_{\bullet b1}(C) \,/\, Tr_{\Sigma c}(B1)$, where •a2 is the set of events of A2 and •b1 is the set of events of B1, then C will have no transitions that do not contribute to the progress of the system. According to the discussion above, we formalize the concept of optimized converters as follows.

Definition 9 (*optimized converter):* a converter C is optimized if C has maximum sequences (in the sense of a maximum solution defined above) under the following condition: $Tr_{\bullet a2}(C) / Tr_{\Sigma c}(A2)$ and $Tr_{\bullet b1}(C) / Tr_{\Sigma c}(B1)$.

3. Some Theoretical Aspects

3.1. The Definition of a Corresponding Refusal Graph

To deal with nondeterministic service specifications, we will introduce the concept of refusal graph, shortly *Rgraph.*. The following definition is used when we define a Rgraph for a given FLTS.

Definition 10 (*After set*): Given an FLTS M = (Q, Σ, δ, q0) and •o/•, we define the *after set* of a state p Q for •o/• as $A_{\Sigma o}(p) = \{p' \mid p =>_{\bullet o} p'$. For any s $p(Q)\}$, we denote $s^{\tau} = \bigcup_{p \in S} A_{\Sigma o}(p)$.

The *After* set $A_{\Sigma o}(p)$ intuitively describes all the reachable states from a state p by executing zero, one or more events e•-•o ≈{τ}.

Definition 11 (*Refusal graph*): A *Rgraph.* is a 5-tuple $G_{\Sigma o} = (S, \Sigma o, \delta, R, s0)$, where

S is a finite set of states;

Σo is a set of events;

δ: S∞ΣoS is a transition function;

R: S$p(p(\Sigma o))$ is a mapping from S to a set of subsets of Σo. R(s) is called a set of refusal sets of state s.

s0 S is the initial state.

This definition is similar to the definition of an FLTS. However, there are two differences: first, from the definition of the transition relation δ: S∞ΣoS, a Rgraph is deterministic; second, there is a mapping relation R: S$p(p(\Sigma o))$ that assigns a set of subsets of Σo to each state s in S, written R(s). R(s) is called a set of refusal set for state s, which will be explained by the following definition.

Definition 12 (*Correspondence between an FLTS and a Rgraph*): Given an FLTS M = (Q, Σ, δ, q0) and a Rgraph $G_{\Sigma o}(M) = (S, \Sigma o, \delta', R, s0)$, where ΣoΣ. We say that $G_{\Sigma o}(M)$ is a corresponding Rgraph of M iff:

1) $S = \{s \mid s=\{qQ \mid q0 = t >_{\bullet o} q\}$ for all t $Tr_{\Sigma o}(M)\}$, which implies S $p(p(Q))$ for the observable events Σo.
2) $s0 = A_{\Sigma o}(q0)$;
3) ∀si, sjS, we have si–e∅sj iff $sj = \left(\bigcup_{p' \in si} \{q' \mid p' - e \; q'\} \right)^{\tau}$.

4) $\forall s S, R(s) = (\bigcup_{q \in s} Ref_{\Sigma o}(M, q))$;

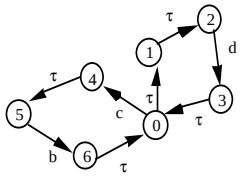
Obviously, given a different Σo, we have a different Rgraph. In this definition, a set of states in M is considered as one state in $G_{\Sigma o}(M)$. This is similar to the method given in [10] for transforming a nondeterministic finite state machine to a trace equivalent deterministic finite state machine, except for the refusal set. By ignoring the set of refusal sets for each state in $G_{\Sigma o}(M)$ we get a deterministic FLTS, denoted as $P_{\Sigma o}(M)$. This definition implies that Rgraph is unique up to an isomorphism because of the following two facts:

1) If there are two corresponding Rgraphs, $G_{\Sigma o}(M)$ and $G'_{\Sigma o}(M)$, for a given FLTS M, then $Tr_{\Sigma o}(G_{\Sigma o}(M)) = Tr_{\Sigma o}(G'_{\Sigma o}(M))$.
2) For any trace t, if $s0 = t >._o s$ in $G_{\Sigma o}(M)$, then there is a state s' in $G'_{\Sigma o}(M)$ such that $s0' = t >._o s'$ and R(s) = R(s').

Therefore, we have the following lemma.

Lemma 1 (Uniqueness of the Rgraph): For any FLTS, its corresponding Rgraph is unique up to an isomorphism.

Example 1: For the given FLTS M specified by Fig.4(a), where Σo = {c, d, b}, the obtained refusal graph is shown in Fig.4(b). In Fig.4(b), we have the shadowed boxes: $s0 = A_{\Sigma o}(0)$, $s1 = A_{\Sigma o}(3)$, $s2 = A_{\Sigma o}(4)$, $s3 = A_{\Sigma o}(6)$. The refusal sets are shown beside each state of the Rgraph.



(a)

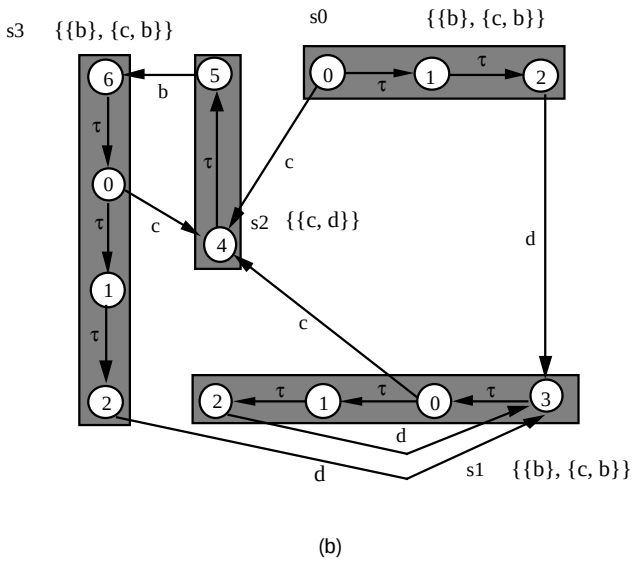


Fig.4 (a) The specification M, (b) The Rgraph of M.

3.2. Existence of Protocol Converter

Our goal is to construct a deterministic converter C such that $M0||C \ \Sigma_o$ Sc. This can be implemented in two steps: first, we may compute $M0\#G_{\Sigma S}(Sc)$ to obtain from M0 all execution sequences that satisfy the saftety property, i.e., $Tr_{\Sigma s}(M0\#G_{\Sigma S}(Sc)) \prod Tr_{\Sigma s}(Sc)$. Second, since some behaviours in $M0\#G_{\Sigma S}(Sc)$ may not satisfy the progress property, we need to prevent them from happening for avoiding deadlocks. To do so, we first define the concept of a machine with grouped states.

Definition 13 (Machine with grouped states (MGS)): Given an FLTS M = (Q, Σ, δ, q0), we define a corresponding machine with grouped states, written $MGS_{\Sigma o}(M) = (\mathbf{S}, \Sigma, \delta', s0_{q0})$, which is an FLTS such that:

1) **S** = ≈**si**, where **si** ↔ **sj** = ϕ for ij; each **si**∈**S** represents a group of states **si** = $\{si_q | qsi\}$, where si is as defined for the corresponding Rgraph $G_{\Sigma o}(M)$.
2) For any **si**, **sj**∈**S**, and any si_q∈**si**, sj_p∈**sj**, we have si_q –e∅sj_p in $MGS_{\Sigma o}(M)$ iff q –e∅q in M for e ∈Σ≈{τ}.

$MGS_{\Sigma o}(M)$ is different from $G_{\Sigma o}(M)$ in four aspects: first, the state space is different; second, the set of events is Σ, instead of Σo; third, the transition relation in $MGS_{\Sigma o}(M)$ is defined by using the states of **si**; fourth, there are no refusals associated with the states in $MGS_{\Sigma o}(M)$. An important relationship between $MGS_{\Sigma o}(M)$ and M is stated by the following lemma.

Lemma 2: For any FLTS M = (Q, Σ, δ, p) and any Σo∏Σ, $MGS_{\Sigma o}(M) \cong M$.

To implement the two-step construction, we face a fundamental problem: what behaviour in M0 may be inhibited? To answer this question, we have the following important theorem.

Theorem 1: Given M0 = (Q0, Σ0, δ0, p0), Sc = (Qs, Σs, δs, q0s) and a Σc=Σ0–Σs, let H be a submachine of $MGS_{\Sigma c}(M0\#G_{\Sigma S}(Sc))$. H ≅ $M0\#P_{\Sigma c}(H)$ iff H satisfies the following two conditions:

1) For each state (p, s) H, where, p is a state of M0 and s is a state of $G_{\Sigma S}(Sc)$, and ∀eΣs≈{τ}, p–e∅p' in M0 implies there must be a state (p', s') in H such that (p, s)–e∅(p', s').
2) For each state group **s** in H and eΣo, if ∃(p, s) **s** and state p in M0 such that (p–e∅)((p, s)–e$\not\rightarrow$) then (p', s')–e$\not\rightarrow$ for ∀(p', s')**s**.

The first condition implies that if an event in Σs ≈ {τ} occurs following a sample path in E(H), then the extended sample path must remains in E(H), provided that the extended path is in E(M0). The second conditions means that for a given **s** in H and (p, s) **s**, if an event eΣo is enabled in M0 at state p, but is disabled at state (p, s) in H, then all states in **s** should disable event e. The result of this theorem shows that the interaction between M0 and $P_{\Sigma c}(H)$ behaves exactly like what H does if H satisfies the two conditions. We call this property of H *invariance property* with respect to M0 and Sc.

Theorem 2: Given M0 = (Q0, Σ0, δ0, p0), Sc = (Qs, Σs, δs, q0s) and a projection Σc=Σ0–Σs. There is a deterministic protocol converter C = (Qc, Σc, δc, q0c) such that M0||C $_{\Sigma S}$ Sc iff there exists a submachine H of $MGS_{\Sigma c}(M0\#G_{\Sigma S}(Sc))$ having the invariance property with respect to M0 and Sc and satisfying H $_{\Sigma S}$ Sc.

Theorem 1 and Theorem 2 imply that we can construct a protocol converter by finding a submachine H of $MGS_{\Sigma c}(M0\#G_{\Sigma S}(Sc))$, which has the invariance property with respect to M0 and Sc and satisfies H $_{\Sigma S}$ Sc.

3.3. Optimization

As we have discussed in Section 2.2, the converter satisfying the condition in Theorem 2 may not be exactly what we want, since it may contain unnecessary states and transitions. To solve this problem, we have the following two basic observations:

1) Some superfluous transitions and states in the protocol converter are due to the property of the channel's behaviour: the channel is able to transmit any messages (including the unnecessary ones). So the unnecessary message sequences are also included in M0. However, the existing algorithms have not used any effective measure to remove them [2] [7] [11].

2) Some superfluous transitions and states may be due to superfluous transitions and states in the original protocols to be converted. Therefore, we make the assumption that there are no superfluous transitions and states in the given protocols.

Based on the observations above, the following theorem describes the basic idea of how to generate an *optimized* protocol converter.

Theorem 3: Given two protocols A and B, a global service specification Sc, the channel specifications Cha and Chb. Let Cha' = Cha#$P_{\bullet a2\text{-}\bullet s}$(A2), Chb' = Chb#$P_{\bullet b1\text{-}\bullet s}$(B1), M0 = (A1||Cha)∞(Chb||B2) and M0' = (A1||Cha')∞(Chb'||B2). If there exists a deterministic converter C such that M0||C Σ_S Sc, then

1) There is a maximum solution C' such that M0'||C' Σ_S Sc.
2) C' is an optimized converter for the given M0 and Sc.

This theorem implies that an optimized converter can be obtained by first obtaining some constraints from the given protocol entities, and removing those message sequences with the constraints from the given channel specifications that may result in superfluous states and transitions in the converter, then constructing the optimized converter with the modified channel specifications.

3.4. Diagnosis of the Reduction Relation

From Theorem 1 and Theorem 2, in order to construct a protocol converter we need to find a submachine H of $MGS_{\Sigma C}$(M0#$G_{\Sigma S}$(Sc)) such that H satisfies H Σ_S Sc. The following theorem is for this purpose.

Theorem 4: Given M0 = (Q, Σ0, δ, p0) and Sc = (Q_S, Σ_S, δ_S, q0). A submachine H of $MGS_{\Sigma C}$(M0#$G_{\Sigma S}$(Sc)) satisfies H Σ_S Sc iff for every state (p, s) of H, there is at least one refusal set Rf R(s) such that $Ref_{\Sigma S}$(H, (p, s)) ∏ Rf, where s is a state of $G_{\Sigma S}$(Sc).

4. Algorithm for Protocol Conversion

4.1. The Algorithm

Based on the theorem 2 through 4 explained above, it is easy to construct an algorithm for protocol conversion. The following algorithm is divided into five steps. In Step 1, the constraints for optimization are derived from the given protocol entities according to Theorem 3, and applied to the channel specifications. M0 is obtained by composing the protocol specifications and the modified channel specifications. In the second step, the execution sequences in M0 that violate the safety property is removed by computing M0' = M0#$G_{\Sigma S}$(Sc) and the states that do not satisfy Condition 1 of Theorem 1

are deleted by marking them Bad States (BD). In the third step, we construct a submachine F of $MGS_{\Sigma C}(M0')$ such that the states and transitions that violate the conditions of Theorem 1 are removed. In Step 4, we construct a submachine H of F such that the states and transitions that do not satisfy Theorem 2 (the progress property and the invariance property) are marked out, and the converter is obtained in Step 5 by computing $C = P_{\Sigma c}(H)$.

Algorithm-Conversion

/* Input: The Protocols A =(A1, A2) and B= (B1, B2), the channel cha of protocol A, the channel chb of protocol B, service specification Sc = (Qs, Σs, δs, q0s), and Σc=Σ0-Σs.

/* Output: an optimized protocol converter C.

Begin

Step 1 *Compute M0 from protocols and channels (Theorem 3):*

1) Derive the constraints: $A2' = P_{.a2 - .s}(A2)$ and $B1' = P_{.b1- .s}(B1)$.
2) Imposing the constraints to the channel specifications: Cha' = Cha#A2' and Chb' = Chb#B1'.
3) Construct M0 = (A1||Cha')∞(Chb'||B2) = (Q, Σ0, δ, p0)

Step 2 *Compute M0' = M0#$G_{\Sigma s}$(Sc) to get the behaviour from M0 satisfying the safety property:*

Compute M0' = M0#$G_{\Sigma S}$(Sc) and mark any state (p, s) of M0' BS ("Bad State") if there is a state p' in M0 such that $p - e \varnothing p'$ for eΣs, but there is not a state s' in $G_{\Sigma s}$(Sc) such that $s - e \varnothing s'$ (condition 1 of Theorem 1). The result is denoted as M0' = (Qp, Σp, δp, (p0, s0)), where s0 is the initial state of $G_{\Sigma s}$(Sc);

Step 3 *Construct a submachine F of $MGS_{\Sigma C}(M0')$ by removing all execution sequences of $MGS_{\Sigma O}(M0')$ that violate the conditions of Theorem 1.*

Create **s0** = $A_{\Sigma C}((p0, s0))$ and mark it TP ("To be Processed");

Create a transition labelled $e\Sigma s \approx \{\tau\}$ from (p, s)**s0** to (p', s')**s0** whenever $(p, s) - e \varnothing (p', s')$ exists.

Do the following while there is an **si** marked TP:

1) If there is a state (p, s)**si** marked BS then mark **si** BS; otherwise for every eΣc do the following:

a) Compute **si**(e)= $\bigcup_{(p, s) \in \mathbf{si}} \{A_{\Sigma C}((p', s'))|(p,s) - e \varnothing (p', s') \delta p\}$;

b) If there is a state (p, s) **si**(e) marked BS then forbid any transition labelled e at **si**; otherwise do the following:

i) if **si**(e) is not empty and there is no previously created **sj** containing exactly all the state pairs in **si**(e), do the following:

- Create such an **sj** containing all the state pairs in **si**(e).
- Create a transition labelled e∈Σs≈{τ} from (p, s) **sj** to (p', s') **sj** whenever (p, s) – e'∅ (p', s') exists.
- Mark **sj** TP;

ii) Create a transition labelled e from (p, s) **si** to (p', s') **sj** whenever (p, s) – e ∅ (p', s') exists.

2) Change the mark of **si** from TP to PD ("ProcesseD")

Step 4 *Construct an FLTS H by removing the states and transitions from F that violate* $F_{\Sigma s}Sc$ *(Theorem 2 and Theorem 4) and retaining the conditions of Theorem 1.*:

Repeat

a) For each **sj** marked BS in F and any state (p', s') **sj**, if there is an **si** in F and any state (p, s) **si** such that (p, s) – e ∅ (p', s') then forbid all transitions labelled e at **si**.

b) For each **si** and each state (p, s) **si**, if there is no Rf R(s) such that $Ref_{\Sigma s}(F, (p, s)) \prod Rf$ then mark **si** BS.

Until no more **si** has been marked BS during the last repetition.

Step 5 *Generate the converter C from H (Theorem 2):*

If s0 is marked BS then report "no solution C", otherwise compute $C = P_{\Sigma c}(H)$. (The states marked BS do not belong to H).

End

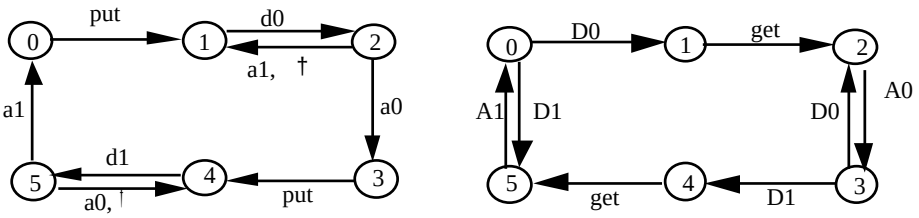
Since M0 and Sc are assumed to be finite, this algorithm will eventually terminate. It is obvious that the computational complexity of Step 3 is exponential in the worst case. However, according to our experience, the number of states of $MGS_{\Sigma c}(M0')$ is of the same order as M0' for many applications. The Step 4 of the algorithm can be implemented more efficiently by recursively checking only the states in which at least one transition is removed by the most recent manipulations of the algorithm.

Theorem 5: If there exists a deterministic converter C' such that A1||Cha||C'||Chb||B2 Σ_s Sc, then Algorithm-Conversion will generate an optimized protocol converter C such that A1||Cha||C||Chb||B2 Σ_s Sc.

4.2. An Example Application

In this section, we apply our algorithm to the conversion between the AB protocol and the NS protocol used in [4] and [7]. The AB protocol is shown in Fig.5. The "put" and "get" events constitute the interface with the user. The events labelled with τ are internal events that model *timeout* or message *loss*. Other events are coupled with the channels. The AB protocol attaches a one bit sequence number to each message transmitted. Sending data messages are denoted as "di" (where i = 1, 2), and receiving data messages are denoted as "Di". For each received data message "Di", the receiver returns an acknowledge message "Ai". If the sender received an acknowledgment "ai" whose sequence number does not match the sequence number of the last-sent data message, or a timeout event τ occurred, it will retransmit the data message. The NS protocol is shown in Fig.6. The "putn" and "getn" events constitute the interface with the user. This protocol does not use message sequence numbers. Sending data messages are denoted as "d", and receiving data messages are denoted as "D". For each received data message "D", the receiver returns an acknowledge message "A". The sender keeps sending the same message "d" after timeout, until it receives an acknowledgment "a". Both protocols guarantee that a message will be delivered at least once. While the NS protocol may deliver the same message multiple times, the AB protocol delivers a message exactly once. The two channel specifications are depicted in Fig.7. Both are not reliable. The message losses or corruption are represented by internal events. The desired service specification is shown in Fig.8, which tolerates duplicated messages.

Fig.9 shows the constraints, A1' and N0', obtained from the two protocol entities A1 and N0, respectively. The modified channel specifications, Chab' = Chab#A1' and Chns' = Chns#N0', are given in Fig.10. Fig.11 shows M0 constructed from A0, N1, Chab' and Chns'. Fig.12 is the refusal graph for the service specification. Fig.13(a) is the optimized converter C constructed by Algorithm-Conversion.

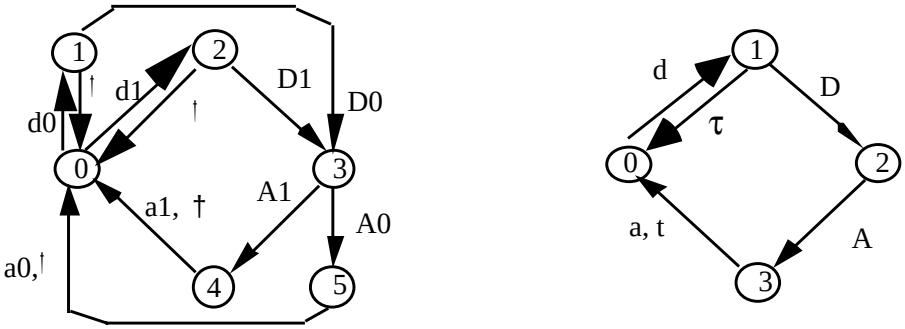


(a) The Sender A0 (b) The Receiver A1

Fig.5 The AB protocol specification

(a) The Sender (N0) (b) The Receiver (N1)

Fig.6 The NS protocol



(a) The AB channel Chab (b) The NS channel Chns.

Fig.7 The channel specification.

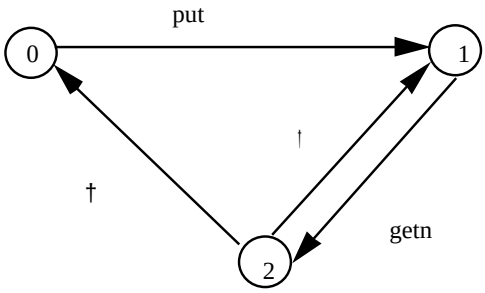
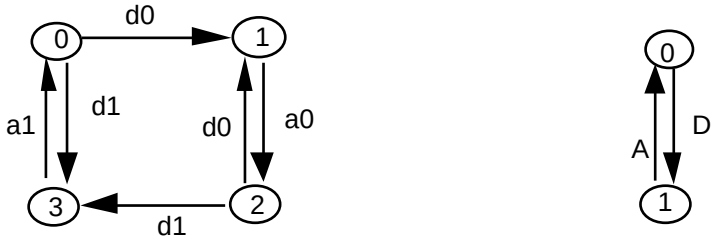


Fig.8 The service specifications.



(a) A0’ (b) N1’

Fig.9 The generated constraints in the first step.

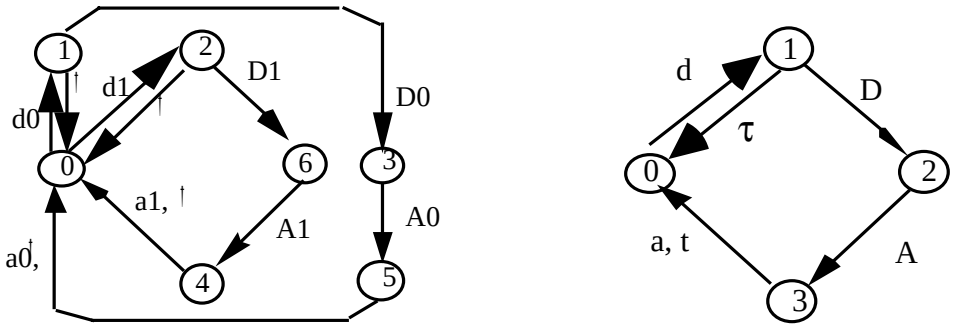


Fig.10 The modified channel specification with constraints: Chab' and Chns'.

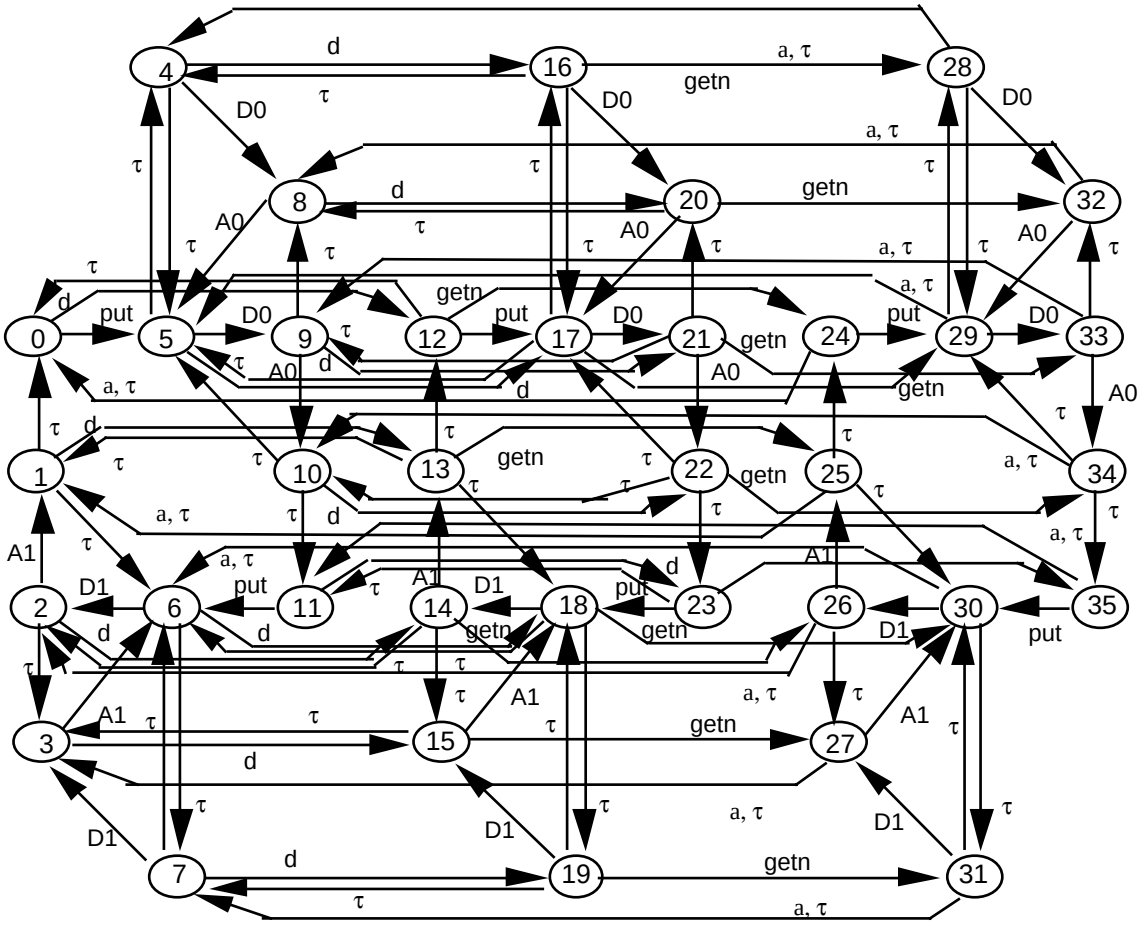


Fig.11 M0 = (A0||Chab')∞(N1||Chns')

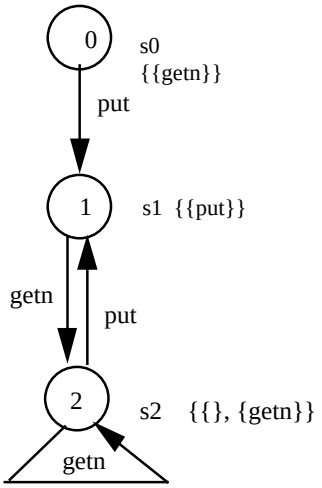


Fig. 12 The specification of $G_{\Sigma s}(Sc)$.

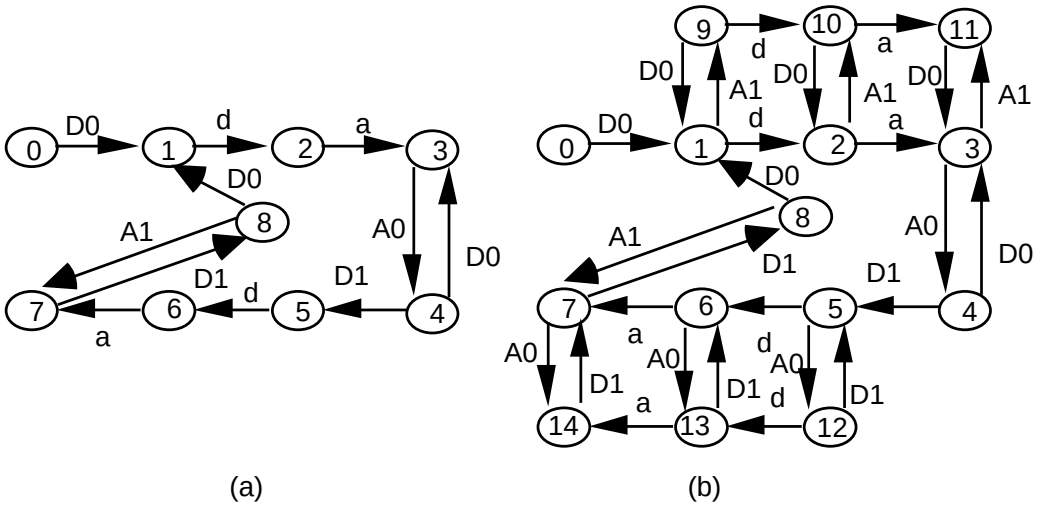


Fig.13 (a) The optimized converter by Algorithm-Conversion ; b) the unoptimized converter

Discussion: This example is the same used in [4] and [7]. Since the algorithm proposed in [4] does not deal with nondeterministic service specifications, the service specification in [4] is specified by not only using observable events, but also explicitly using *timeout* and message *loss* events. In this paper, we simply model the message *loss* and *timeout* by internal event τ, which simplifies the specification and the algorithm. Modelling service specification by explicitly using message losses and timeout events is not what we want. As discussed in [4] and [7], in the worst case the number of states to be checked by the algorithm proposed in [4] is $2^{|Qm0|} \infty |Qsc|$, where $|Q_{m0}|$ and $|Q_{sc}|$ denote the number of states of M0 and Sc, respectively. From the construction of $MGS_{\Sigma c}(M0\#G_{\Sigma S}(Sc))$, the number of states of $MGS_{\Sigma c}(M0\#G_{\Sigma S}(Sc))$ is mainly determined by its transitions in the worst case. Our algorithm can reduce

the computation significantly compared with the algorithms proposed in [4] [7] [16], since the major computation needed is in Step 3 (in fact it is exponential in the worst case), and the unnecessary execution sequences appeared in the unoptimized converter are removed by Step 1 (therefore the number of states and transitions of $MGS_{\Sigma C}(M0\#G_{\Sigma S}(Sc))$ can be reduced significantly). If we do not apply Step 1 to the example given above, the number of states needs to be checked will be several times larger than that needed by Algorithm-Conversion in Step 3 and Step 4. The mathematical analysis in this aspect is difficult in general. Fig.13(b) shows the unoptimized converter directly generated by algorithms proposed in [4] [7] [16], in which six states and sixteen transitions are superfluous.

5. Conclusions

In this paper we have defined the concept of optimized protocol converter, and proposed a simple top-down algorithm to construct optimized converters from a given service specification and two protocol specifications. The basic idea is to derive constraints from the protocol specifications and impose the constraints on the channel specifications. Compared with related works, our method has the following three advantages: (1) it generates an optimized converter; (2) The service specification may be nondeterministic; (3) It may need less computation.

This algorithm may also be adapted to construct controllers for discrete event systems [18], as described in [22].

References

[1] G. v. Bochmann, "Deriving Protocol Converters for Communication Gateways", IEEE Trans. Comm., Vol. 38, No. 9, Sept. 1990.

[2] G. v. Bochmann, et al, "Design Principles for Communication Gateways", IEEE Journal on Selected Areas in Communications, Vol. 8, No. 1, Jan. 1990.

[3] E. Brinksma, G. Scollo, C.Steenbergen, LOTOS Specification, their implementations, and their Tests, in Proceedings of IFIP Workshop PSTV, 1987.

[4] K. L. Calvert, S.S. Lam, Deriving a Protocol Converter: A Top-Down Method, Proceedings of ACM SIGCOMM '89.

[5] P. E. Green, JR, Protocol Conversion, IEEE Transaction on Communications, Vol. 34, No. 3, March 1986.

[6] D. M. Kristol, et al, "Efficient Gateway Synthesis from Formal Specifications", Proceedings of ACM SIGCOMM'91. See also IEEE/ACM Trans. on Networking, Vol.1, No.2, April, 1993.

[7] S.S. Lam, K. L. Calvert, Formal Methods for Protocol Conversion, IEEE Journal on Selected Areas in Communications, Vol.8, No. 1, January 1990..

[8] S.S. Lam, Protocol Conversion, IEEE Trans. Software Eng., Vol. 14, Mar.1988.

[9] G. Leduc, A Framework Based on Implementation Relations for Implementing LOTOS Specifications, Computer Networks and ISDN Systems 25 (1992).

[10] Harry R. Lewis, Elements of the Theory of Computation, Printice-Hall, 1981, pp59-pp62.

[11] P. Merlin and G. v. Bochmann, On the Construction of Submodule Specifications and Communication Protocols, ACM TOPLAS 5(1), 1983.

[12] Robin Milner, Communication and Concurrency, Prentice Hall, 1989.

[13] Rocco De Nicola, Extensional Equivalences for Transition Systems, Acta Informatica 24, 1987.

[14] K. Okumura, A Formal Protocol Conversion Method, Proceedings of ACM SIGCOMM '86.

[15] K. Okumura, Generation of Proper Adapters and Converters from a Formal Service Specification, Proceedings of IEEE INFOCOM '90.

[16] Parrow, J. Submodule Construction as Equation Solving in CCS, Theoretical Computer Science, 68, 1989.

[17] M. Rajagopal, et al, Synthesizing a Protocol Converter from Executable Protocol Traces, IEEE Transactions on Computers, Vol. 40, No. 4, April 1991.

[18] P.J.G. Ramadge and W.M. Wonham "The Control of Discrete Event Systems" Proceedings of the IEEE, Vol.77, No.1, Jan. 1989.

[19] J.C. Shu, and M.T. Liu, A Synchronization Model for Protocol Conversion, In Proc. IEEE INFOCOMM '89, Ottawa, Canada, 1989.

[20] J.C. Shu, and M.T. Liu, An Approach to Indirect Protocol Conversion, Computer Networks and ISDN Systems 21, 1991.

[21] Y.W. Yao, W.S. Chen, and M.T. Liu, A Modular Approach to Constructing Protocol Converters, Proc. IEEE INFOCOM'90, San Francisco, CA, 1990.

[22] Z.P.Tao, G.v.Bochmann, and R.Dssouli, A Model and an Algorithm of Subsystem Construction, Proc. ISCA PDC95, Sept. 1995, Orlando, Florida, U.S.A.

Appendix

Proof of Lemma 2:

Construct a relation $\zeta = \{<p, si_p> | p$ is a state of M, si is a state group of $MGS_{\Sigma O}(M), si_p si\}$, then for any $<p, si_p> \zeta$, $p -e\varnothing p'$ in M implies $si_p -e\varnothing sj_{p'}$ in $MGS_{\Sigma O}(M)$; and $si_p -e\varnothing sj_{p'}$ implies $p -e\varnothing p'$ and $sj_{p'}$, s_j according to Definition 13. Hence $<p', sj_{p'}> \zeta$. Therefore, $M \cong MGS_{\Sigma O}(M)$.

The proof of Theorem 1:

Let us construct a relation $\zeta = \{<(p, s), (p, s_h)> | (p, s)$ is a state of H, (p, s_h) is a state of $M0\#P_{\Sigma O}(H)$ and (p, s) s_h in $P_{\Sigma O}(H)\}$. Obviously $<(p0, s0), (p0, s_{h0})> \zeta$. If $H \cong M0\#P_{\Sigma O}(H)$, then for any $<(p, s), (p, s_h)> \zeta$, $(p, s) -e\varnothing (p', s')$ implies $(p, s_h) -e\varnothing (p', s_h')$; and $(p, s_h) -e\varnothing (p', s_h')$ implies $(p, s) -e\varnothing (p', s')$ and (p', s') s_h' in $P_{\Sigma O}(H)$. Hence $<(p', s'), (p', s_h')> \zeta$, and ζ is a strong bisimulation relation.

(>) Assume that $H \cong M0\#P_{\Sigma O}(H)$. If there is a state (p, s) in H, where, p is a state of M0 and s is a state of $G_{\Sigma S}(Sc)$, and an event $e\Sigma s \approx \{\tau\}$, such that $p -e\varnothing p'$ in M0 but $(p, s) -e\not\rightarrow$ in H, then at state (p, s_h) of $M0\#P_{\Sigma O}(H)$, where s_h is a state of $P_{\Sigma O}(H)$ and (p, s) s_h, $(p, s_h) -e\varnothing$ according to # product. Therefore, $<(p, s), (p, s_h)> \zeta$. This contradicts with $H \cong M0\#P_{\Sigma O}(H)$.

If there is State-set s in H and $e\Sigma o$, and $\exists (p, s)$ s_h such that $(p -e\varnothing)((p, s) -e\not\rightarrow)$ and $(p', s') -e\varnothing$ for some $(p', s')s_h$, then $(p, s_h) -e\varnothing$ must be true in $M0\#P_{\Sigma O}(H)$ according to # product. Therefore, $<(p, s), (p, s_h)> \zeta$. This also contradicts with $H \cong M0\#P_{\Sigma O}(H)$.

(<) Construct a relation $\zeta = \{<(p, s), (p, s_h)> | (p, s)$ is a state of H, (p, s_h) is a state of $M0\#P_{\Sigma O}(H)$ and (p, s) s_h in $P_{\Sigma O}(H)\}$. If the two conditions of Theorem 1 are satisfied, then for any $<(p, s), (p, s_h)> \zeta$, $(p, s_h) -e\varnothing (p', s_h')$ in $M0\#P_{\Sigma O}(H)$ iff $(p, s) -e\varnothing (p', s')$ in H and (p', s') s_h' in $P_{\Sigma O}(H)$. So, $<(p', s'), (p', s_h')> \zeta$. Since $<(p0, s0), (p0, s_{h0})> \zeta$ by the construction of ζ, we have $H \cong M0\#P_{\Sigma O}(H)$.

The proof of Theorem 2:

(>) If such a H exists, according to Theorem 1, $M0\#P_{\Sigma O}(H) \cong H$, therefore $M0||P_{\Sigma O}(H) \cong H|_{\Sigma o}$, where $H|_{\Sigma o}$ denotes the FLTS derived from H by replacing every event $e\Sigma o$ with an internal event τ. From the condition $H \ _{\Sigma S}$ Sc, we have $M0||C \ _{\Sigma S}$ Sc, where $C = P_{\Sigma O}(H)$.

(<) If there is a solution C such that $M0||C \ _{\Sigma S}$ Sc, let $H' = M0\#C$, we have $H' \ _{\Sigma O}$ Sc. Obviously, H' is a submachine of $M0\#G_{\Sigma S}(Sc)$ if we rename the states. Construct a relation $\zeta = \{<(p, s), (p, s_h)> | (p, s)$

is a state of H', (p, s_h) is a state of M0#PΣ_O(H') and (p, s) s_h in PΣ_O(H')}. Obviously <(p0, s0), (p0, s_{h0})> ζ. Then, for any <(p, s), (p, s_h)> ζ, (p, s_h) $-e\varnothing$ (p', s_h') in M0#PΣ_O(H) iff (p, s)$-e\varnothing$(p', s') in H and (p', s') s_h' in PΣ_O(H). Therefore, H' ≅ M0#PΣ_O(H'). Let H = MGSΣ_O(H'), from Lemma 2, we proved the theorem.

The proof of Theorem 3:

1) Assuming there is a converter C such that M0||C Σ_S Sc, then there is a submachine H of MGSΣ_O(M0#GΣ_S(Sc)) satisfying the conditions of Theorem 2. Let M0' = (A1||Cha')∞(Chb'||B2), and remove the transitions and states from MGSΣ_O(M0'#GΣ_S(Sc)) that do not satisfy the conditions of Theorem 2, we get a submachine H' of MGSΣ_O(M0'#GΣ_S(Sc)). H' can be considered to be a submachine of H after removing the transitions and states that violate the conditions Tr$_{\bullet a2}$(H)/Tr$_{\bullet a2\text{-}\bullet s}$(A2) and Tr$_{\bullet b1}$(H)/Tr$_{\bullet b1\text{-}\bullet s}$(B1). From the requirements Sc$_{\Sigma a1}$ A1||Cha||A2 and Sc$_{\Sigma b2}$ B1||Chb||B2, if H is not empty, then H' is not empty. Therefore, there is a maximum solution C' such that M0'||C' Σ_S Sc.

2) From M' = (A1||Cha')∞(Chb'||B2), we have Tr$_{\bullet a2}$(C')/Tr$_{\Sigma c}$(A2) and Tr$_{\bullet b1}$(C'/ Tr$_{\Sigma c}$(B1). Hence, C' is optimized according Definition 9.

The proof of Theorem 4:

(>) If H Σ_S Sc, for any state (p, s) in H, where s is a state of GΣ_S(Sc), there must be a trace t TrΣ_S(H) ↔TrΣ_S(Sc) such that q0c = t > q' in Sc such that RefΣ_S(H, (p, s)) ∏ RefΣ_S(Sc, q'), where q' s_h in GΣ_S(Sc). Therefore, there is at least one refusal set Rf R(s) such that RefΣ_S(H, (p, s)) ∏ Rf.

(<) From the definition of # product, TrΣ_S(H) ∏ TrΣ_S(Sc), i.e., Condition 1 of Definition 6 is true. For any trace t TrΣ_S(H) ↔TrΣ_S(Sc), and any state (p, s) after t in H, if there is at least one refusal set Rf R(s) such that RefΣ_S(H, (p, s)) ∏ Rf, then there must be a state q' after t in Sc such that q's and RefΣ_S(Sc, q') = Rf. Therefore, Condition 2 of Definition 6 is true. Hence H Σ_S Sc.

The proof of Theorem 5:

If there exists a deterministic converter C' such that A1||Cha||C'||Chb||B2 Σ_S Sc, then A1||Cha'||C'||Chb'||B2 Σ_S Sc is also true according to Theorem 3. Let M0' = (A1||Cha')∞(Chb'||B2), according to Theorem 2, no execution sequences of E(A1||Cha'||C'||Chb'||B2) will be removed in Step 2, Step 3 and Step 4. Therefore, s0 will not be marked BS. The algorithm will generate a converter C such that A1||Cha||C||Chb||B2 Σ_S Sc. Since Step 1 of the algorithm will generate M0', the conditions Tr$_{\bullet a2}$(C)/Tr$_{\Sigma c}$(A2) and Tr$_{\bullet b1}$(C)/ Tr$_{\Sigma c}$(B1) are satisfied. The theorem is proved.