IEICE TRANS. INF. & SYST., VOL.E100–D, NO.6 JUNE 2017

PAPER *Special Section on Formal Approach*

Synthesizing Pareto Efficient Intelligible State Machines from Communication Diagram*

Toshiyuki MIYAMOTO[†a)], ***Senior Member***

SUMMARY For a service-oriented architecture based system, the problem of synthesizing a concrete model, i.e., behavioral model, for each service configuring the system from an abstract specification, which is referred to as choreography, is known as the choreography realization problem. In this paper, we assume that choreography is given by an acyclic relation. We have already shown that the condition for the behavioral model is given by lower and upper bounds of acyclic relations. Thus, the degree of freedom for behavioral models increases; developing algorithms of synthesizing an intelligible model for users becomes possible. In this paper, we introduce several metrics for intelligibility of state machines, and study the algorithm of synthesizing Pareto efficient state machines.

key words: *unified modeling language, choreography realization problem, Petri nets, automatic synthesis, service-oriented architecture*

1. Introduction

The internationalization of business activities and information and communication technology have intensified competition among companies. Companies are under pressure to quickly respond to business needs, and the time frame for making changes to existing business and launching new businesses has been shortened. Therefore, the need to quickly change or build information systems has been increasing. Under such circumstances, service-oriented architecture (SOA) [1] has been attracting attention as the architecture of information systems. In SOA, an information system is built by composing independent software units called peers.

In this paper, we consider the problem of synthesizing a concrete model from an abstract specification. We assume that a concrete model describes the behavior of peers and an abstract specification describes how peers interact with each other. In SOA, the problem of synthesizing a concrete model from an abstract specification is known as the choreography realization problem (CRP) [2], [3]. The abstract specification, called *choreography*, is defined as a set of interactions among peers, which are given by a dependency relation among messages; the concrete model is called *service implementation*, which defines the behavior of the peer. This paper uses the communication diagram and the state machine of Unified Modeling Language (UML) 2.x [4] to describe the choreography and service implementation, respectively. In this paper, it is assumed that the dependency relation is acyclic. Thus, only choreography with no iteration can be accepted.

Bultan and Fu formally studied the CRP [2]. They used collaboration diagrams of UML 1.x and showed that the conditions for the given choreography are realizable. In addition, they showed a method for synthesizing a set of finite state machines with projection mapping. However, the synthesized state machines are not *intelligible* because the number of states increases exponentially as the number of messages increases.

Intelligibility, however, is highly subjective and it is difficult to discuss this concept quantitatively. Cruz-Lemus et al. experimentally evaluated the relationship between some metrics of state machines and the time taken to understand them [5]. According to the results, state machines are more easily understood as values of the following metrics become small: the number of simple states (NSS), the number of transitions (NT), the number of guards (NG), and the number of do-activities (NA). We use these metrics for intelligibility evaluation. Because we have plural metrics, there may exist a set of state machines, called Pareto efficient state machines, each of which is superior to others in terms of at least one of the metrics.

Miyamoto et al. proposed the Construct State-machine Cutting Bridges (CSCB) method, a method for synthesizing hierarchical state machines from a communication diagram [6]. In the method, dependency relations among sent and received message events are represented by Petri nets [7]; state machines are then synthesized. Recently, a new notion called re-constructible decomposition of acyclic relations was introduced; a necessary and sufficient condition for a decomposed relation to be re-constructible was shown [8]. In this paper, we extend the CSCB method for synthesizing Pareto efficient state machines using the decomposition scheme.

2. Preliminaries

2.1 Relations

Let Σ be a finite set and $\mathcal{R}$ be a relation on Σ. The transitive closure and reduction of $\mathcal{R}$ is denoted by $\mathcal{R}^+$ and $\mathcal{R}^-$, respectively. A relation $\mathcal{R}$ is called *cyclic* if e_1 and $e_2 \in \Sigma$ exist such that $(e_1, e_2) \in \mathcal{R}$ and $(e_2, e_1) \in \mathcal{R}^+$; otherwise it is

Manuscript received August 16, 2016.
Manuscript revised December 26, 2016.
Manuscript publicized March 7, 2017.

[†]The author is with the Division of Electrical, Electronic and Information Engineering, Graduate School of Engineering, Osaka University, Suita-shi, 565–0871 Japan.

[*]This work was supported by JSPS KAKENHI Grant Number JP26330083.

a) E-mail: miyamoto@eei.eng.osaka-u.ac.jp

DOI: 10.1587/transinf.2016FOP0002

called *acyclic*. The set of all topological sorts of an acyclic directed graph $(\Sigma, \mathcal{R})$ is denoted by $\mathfrak{L}(\mathcal{R})$. A topological sort is called a *word* and the set is called a *language*.

Let C be a set and $\{\Sigma_c\}$ be a partition of Σ wrt $c \in C$. Let $\mathcal{R}_c$ be a relation on Σ_c and their set be $\{\mathcal{R}_c\} = \{\mathcal{R}_c \subseteq \Sigma_c^2 \mid \Sigma_c \in \{\Sigma_c\}\}$. A relation $\mathcal{R}_{\mathrm{com}} \subseteq \mathcal{R} \setminus (\bigcup_c \Sigma_c^2)$ is called a *communal relation* of $\mathcal{R}$.

Definition 1 (Re-constructible Decomposition): Given a set $\{\mathcal{R}_c\}$ of relations and a communal relation $\mathcal{R}_{\mathrm{com}}$, the relations $\{\mathcal{R}_c\}$ are *re-constructible* to $\mathcal{R}$ if $\mathfrak{L}(\mathcal{R}_{\mathrm{com}} \cup \bigcup_c \mathcal{R}_c) = \mathfrak{L}(\mathcal{R})$.

Relations $\mathcal{R}_c^{\max}$, $\mathcal{R}_c^{\min}$, and $\mathcal{R}^{\min}$ are defined as follows:

$$\mathcal{R}_c^{\min} = \Sigma_c^2 \cap \mathcal{R}^-, \quad (1)$$

$$\mathcal{R}_c^{\max} = \Sigma_c^2 \cap \mathcal{R}^+, \text{ and} \quad (2)$$

$$\mathcal{R}^{\min} = \mathcal{R}_{\mathrm{com}} \cup (\textstyle\bigcup_c \mathcal{R}_c^{\min}), \quad (3)$$

where $\mathcal{R}_c^{\max}$, $\mathcal{R}_c^{\min}$, and $\mathcal{R}^{\min}$ are acyclic because they are subrelations of $\mathcal{R}$.

We put the following assumption on relation $\mathcal{R}$ and its communal relation $\mathcal{R}_{\mathrm{com}}$.

Assumption 1: $\mathfrak{L}(\mathcal{R}) = \mathfrak{L}(\mathcal{R}^{\min})$.

This assumption relates to realizability of choreography; please refer to [8] for details.

On realizing choreography, we want to find a re-constructible relations $\{\mathcal{R}_c\}$. The following theorem [8] gives upper and lower bounds for $\{\mathcal{R}_c\}$.

Theorem 1: Under Assumption 1, $\{\mathcal{R}_c\}$ is re-constructible iff $\forall c : \mathcal{R}_c^{\min} \subseteq \mathcal{R}_c \subseteq \mathcal{R}_c^{\max}$.

2.2 cbUML

A subset of UML, which is called cbUML, was introduced by Miyamoto et al. in [6].

Definition 2 (cbUML): A cbUML model is a tuple $(C, \mathcal{M}, \mathcal{A}, \mathcal{CD}, \mathcal{SM})$, where C is the set of classes, $\mathcal{M}$ is the set of messages, $\mathcal{A}$ is the set of attributes, $\mathcal{CD}$ is the set of communication diagrams, and $\mathcal{SM}$ is the set of state machines.

One class exists for each peer, and a state machine defines its behavior. A communication diagram describes a scenario, which is an interaction of peers.

2.2.1 Messages

The set $\mathcal{M}$ of messages is partitioned by the type of messages: $\mathcal{M} = \mathcal{M}_{sop} \cup \mathcal{M}_{aop} \cup \mathcal{M}_{rep}$, where $\mathcal{M}_{sop}$ is the set of *synchronous messages* generated by synchronous calls, $\mathcal{M}_{aop}$ is the set of *asynchronous messages* generated by asynchronous calls, and $\mathcal{M}_{rep}$ is the set of *reply messages* to synchronous messages. Let $\mathcal{M}_s = \mathcal{M}_{sop}$ and $\mathcal{M}_a = \mathcal{M}_{aop} \cup \mathcal{M}_{rep}$. Correspondence between the synchronous call and its reply is given by the function $ref : \mathcal{M} \to \mathcal{M} \cup \{\mathrm{nil}\}$, such that $\forall m \in \mathcal{M}_{sop} : ref(m) \in \mathcal{M}_{rep}$, $\forall m \in \mathcal{M}_{rep} : ref(m) \in \mathcal{M}_{sop}$, $\forall m \in \mathcal{M}_{aop} : ref(m) = \mathrm{nil}$, and $\forall m \in \mathcal{M}_{sop} \cup \mathcal{M}_{rep} : ref(ref(m)) = m$.

In UML, each message has two events: a *send event* and a *receive event*. For a synchronous message, the receive event occurs immediately after the send event. However, for a discussion that occurs subsequently, we need two events that occur sequentially. Therefore, we defined that each synchronous message has two events: a *preparation event* for message sending and a *send-receive event* where the preparation event is a caller's event and the send-receive event is a callee's event. The preparation event and the send-receive event of a synchronous message $m \in \mathcal{M}_s$ are denoted by $\$m$ and $!m$, respectively. For an asynchronous or a reply message $m \in \mathcal{M}_a$, the send and receive events are denoted by $!m$ and $?m$, respectively. Hereafter, an *active event* is the send-receive event of a synchronous message or the send event of an asynchronous or a reply message. The set Σ of message events and set Δ of active events are defined as follows:

$$\Sigma = \{\$m, !m \mid m \in \mathcal{M}_s\} \cup \{!m, ?m \mid m \in \mathcal{M}_a\}, \text{and} \quad (4)$$

$$\Delta = \{!m \mid m \in \mathcal{M}\}. \quad (5)$$

The acyclic relation $\Rightarrow_{\mathcal{M}}$ on the order of the caller's and callee's events for each message is defined as follows:

$$\Rightarrow_{\mathcal{M}} = \{(\$m, !m) \mid m \in \mathcal{M}_s\} \cup \{(!m, ?m) \mid m \in \mathcal{M}_a\}. \quad (6)$$

2.2.2 Communication Diagrams

Definition 3 (Communication Diagram): A communication diagram $cd \in \mathcal{CD}$ is a tuple $cd = (C^{cd}, \mathcal{M}^{cd}, Conn^{cd}, line^{cd}, D^{cd})$, where $C^{cd} \subseteq C$ is the set of classes, which are called *lifelines* and correspond to peers; $\mathcal{M}^{cd} \subseteq \mathcal{M}$ is the set of messages; $Conn^{cd} \subseteq C^{cd} \times C^{cd}$ is the set of connectors, which is given as a symmetric relation on C^{cd}; $line^{cd} : \mathcal{M}^{cd} \to Conn^{cd}$ assigns a connector for each message; and $D^{cd} \subseteq \Delta \times \Delta$ indicates a dependency relation among active events, where D^{cd} must be acyclic.

Superscripts may be omitted if the context is clear.

A *conversation* is a sequence of messages exchanged among peers [2]. The set of conversations defined by a communication diagram cd is denoted by $\mathfrak{C}(cd) \subseteq \mathcal{M}^*$, where $\mathcal{M}^*$ is the set of all sequences of distinct messages.

Definition 4: A conversation $\sigma = m_1 m_2 \cdots m_n$ is in $\mathfrak{C}(cd)$ if and only if $\sigma \in \mathcal{M}^*$ and the corresponding sequence $\gamma = !m_1 !m_2 \cdots !m_n$ of active events satisfy $\forall i, j \in [1..n] : (!m_i, !m_j) \in D \Rightarrow i < j$.

If there exists a communication diagram $cd \in \mathcal{CD}$ such that $\sigma \in \mathfrak{C}(cd)$, then $\sigma \in \mathfrak{C}(\mathcal{CD})$.

2.2.3 State Machines

Definition 5 (State Machine): A state machine is a tuple $sm = (V, R, r^{\mathrm{t}}, \Theta, \Phi, E, C, B)$, where V is the set of vertices, R is the set of regions, $r^{\mathrm{t}} \in R$ is the top region, Θ is an ownership relation between vertices and regions, Φ is the set of transitions, E is the set of events, C is the set of constraints, and B is the set of behaviors.

In UML state machines, although there are various kinds of states and pseudo-states, only *simple states*, *composite states*, *final states*, and *initial pseudo-states* are used in this paper because is it enough for the discussion. Therefore, the set V of vertices is partitioned into the following types of subsets:$V = SS \cup CS \cup FS \cup IS$, where SS is the set of simple states, CS is the set of composite states, FS is the set of final states, and IS is the set of initial pseudo-states.

A region, except for the top region, is owned by a composite state and a composite state is owned by a region. The ownership relation Θ is defined as a function from $(V \cup R) \setminus \{r^{\mathrm{t}}\}$ to $(CS \cup R)$, and $\Theta(x_1) = x_2$ means that x_1 is owned by x_2. For $x \in V \cup R$, let $des(x) = \{x' \mid \exists i > 0 : \Theta^i(x') = x\}$ be the set of descendants of x, where $\Theta^1(\cdot) = \Theta(\cdot)$ and $\Theta^i(\cdot) = \Theta(\Theta^{i-1}(\cdot))$ $(i > 1)$. The top region r^{t} exists in the root of each state machine; this region is not owned by any composite state, and every state and region in any composite state are descendants of the top region.

Definition 6 (Orthogonal State): Two vertices v_1 and $v_2 \in V$ are called *orthogonal* and are denoted by $v_1 \perp v_2$ if there exist different regions r_1 and $r_2 \in R$ such that $r_1 \neq r_2$, $\Theta(r_1) = \Theta(r_2)$, $v_1 \in des(r_1)$, and $v_2 \in des(r_2)$.

Definition 7 (Consistent State): A set $\hat{V} \subset V$ of vertices is called *consistent* if and only if for each $v_1, v_2 \in \hat{V}$; if $v_1 \neq v_2$ then $v_1 \perp v_2$, $v_1 \in des(v_2)$, or $v_2 \in des(v_1)$.

The set E of events is given as $E = \Sigma \cup \{\tau\}$, where Σ is the set of message events in the state machine and τ is the *completion event* that occurs when a transition with no trigger event fires.

A transition $tr \in \Phi$ is a tuple $tr = (src, tri, grd, eff, tgt)$, where $src \in V$ is the originating vertex of the transition, trigger $tri \in E$ is the event that makes the transition fire, guard $grd \in C$ is a condition to fire, effect $eff \in B$ is an optional behavior to be performed when the transition fires, and $tgt \in V$ is the target vertex. The set $\{src, tgt\}$ must not be consistent. A caller's event becomes an effect and a callee's event becomes a trigger; therefore, $\Sigma \subseteq B$. The set B of behaviors may contain an effect that manipulates the attributes of the corresponding class. A guard condition must be a Boolean expression and the attributes of the corresponding class may be used.

A word $w \in \Sigma^*$ is accepted by the set $\mathcal{SM}$ of state machines if every state machine is in the final state in the top region after occurring all events in w, where an intuitive description on the operational semantics is given in Appendix A. A conversation is obtained from an accepted word by removing all non-active events and replacing every active event by its message. The set of all conversations for $\mathcal{SM}$ is denoted by $\mathfrak{C}(\mathcal{SM})$.

2.2.4 Intelligibility Metrics

Intelligibility is highly subjective and it is difficult to discuss this concept quantitatively. Cruz-Lemus et al. experimentally evaluated the relationship between some metrics of state machines and the time taken to understand them [5]. According to the results, state machines are more easily understood as values of following metrics become small: the number of simple states (NSS), number of transitions (NT), number of guards (NG), and number of do-activities (NA). This paper uses the first three metrics (NSS, NT, and NG) for intelligibility because we do not use do-activities.

The study by Cruz-Lemus et al. did not evaluate the effect of the number of depth of hierarchical state machines. However, it is easily expected that as the number of depth increases, the state machine becomes harder to understand. Thus, we add the metric: number of depth (ND).

In our study, a state machine transitions by communicating other state machines. When it communicates with plural state machines concurrently, regions in a composite state may be generated. In such a case, the number of communication partners in a region should be smaller, thus, we add another metric: sum of number of partners for all regions (NP).

Definition 8: State machine sm_1 is more intelligible than state machine sm_2, denoted by $sm_1 \prec sm_2$, iff the value of sm_1 is smaller than or equal to the value of sm_2 for all metrics and smaller at least one metric.

The intelligibility relation is a partial order and thus, minimal elements exist. The set of minimal elements is called Pareto frontier.

3. Choreography Realization

3.1 Choreography Realization Problem

A single communication diagram describes a scenario, which is an interaction of peers in the system. All the behaviors of the system are indicated by a set of communication diagrams; this is referred to as choreography.

Problem 1 (CRP): For a given set $\mathcal{CD}$ of communication diagrams, is it possible to synthesize the set $\mathcal{SM}$ of state machines that satisfy $\mathfrak{C}(\mathcal{CD}) = \mathfrak{C}(\mathcal{SM})$? If possible, obtain the set of state machines.

If not possible, it is preferred that state machines that mimic the choreography as closely as possible are synthesized. A set of state machines that satisfy $\mathfrak{C}(\mathcal{CD}) \supseteq \mathfrak{C}(\mathcal{SM})$ is called a *weak realization* of the given choreography. However, the set of empty state machines such that $\mathfrak{C}(\mathcal{SM}) = \emptyset$ is a weak realization for any choreography; such a realization is called *trivial*. Hereinafter, choreography is called *unrealizable* if non-trivial realization does not exist.

In this paper, we assume that the set of communication diagrams is singleton, $\mathcal{CD} = \{cd\}$, and we study an algorithm to synthesize Pareto efficient state machines from cd.

Let $\Rightarrow$ be the acyclic relation on the set of events defined by cd, $\mathcal{R}_c$ be the acyclic relation for peer c, and $\mathcal{SM}$ be the set of state machines synthesized from $\{\mathcal{R}_c\}$. Under the assumption that the state machine that behaves equivalently to $\mathcal{R}_c$ can be synthesized, we showed the following theorem [8].

Theorem 2: If $\{\mathcal{R}_c\}$ is re-constructible to $\Rightarrow$, then $\mathcal{SM}$ is a strong realization of cd.

Corollary 3: If $\Rightarrow_c^{\min} \subseteq \mathcal{R}_c$ for all $c \in C$, then $\mathcal{SM}$ is a weak realization of cd.

3.2 Extended CSCB Method

We extend the CSCB method [6] to find a set of Pareto efficient state machines in terms of intelligibility. Some relations are introduced in 3.2.1 and the main algorithm is shown in 3.2.2.

3.2.1 Relations

Because the acyclic relation D is a relation on active events, we have to extend it to the relation on active and non-active events. The acyclic relation $\Rightarrow \subseteq \Sigma^2$ on the set of events is obtained by augmenting D, as follows:

$$\begin{aligned}\Rightarrow\; = \;& D \cup \{(?m_1, !m_2) \mid m_1 \in \mathcal{M}_a, m_2 \in \mathcal{M}_a, \Omega(?m_1, !m_2)\}\\ &\cup \{(?m_1, \$m_2) \mid m_1 \in \mathcal{M}_a, m_2 \in \mathcal{M}_s, \Omega(?m_1, \$m_2)\}\\ &\cup \{(!m_1, \$m_2) \mid m_1 \in \mathcal{M}_s, m_2 \in \mathcal{M}_s, \Omega(!m_1, \$m_2)\}\\ &\cup \Rightarrow_{\mathcal{M}} \cup \{(!m, e) \mid m \in \mathcal{M}_s, \Omega(\$m, e)\},\end{aligned} \tag{7}$$

where $\Omega(e_1, e_2)$ is true when both events e_1 and e_2 occur in the same peer and $(!e_1, !e_2) \in D$, where $!e_1$ and $!e_2$ are the corresponding active events for events e_1 and e_2, respectively.

The communal relation for decomposition is given as follows:

$$\Rightarrow_{\mathrm{com}} = \;\Rightarrow_{\mathcal{M}} \cup \{(!m, e) \mid m \in \mathcal{M}_s, \Omega(\$m, e)\}, \tag{8}$$

where $\Rightarrow_{\mathcal{M}}$ is a natural ordering where the callee's event of a message follows the caller's event of the same message; $\{(!m, e) \mid m \in \mathcal{M}_s, \Omega(\$m, e)\}$ implies that an event e that follows a preparation event $\$m$ of a synchronous message and occurs in the same peer follows the send-receive event $!m$ of the message. As stated before, a caller of a synchronous message waits for the occurrence of callee's receive event. Therefore, $!m$ precedes e. In the case of state machines of cbUML, any event following a preparation event follows the send-receive event, as described in the execution semantics of state machines. Therefore, the order given by $\Rightarrow_{\mathrm{com}}$ is kept when multiple state machines are executed in parallel.

Let us define $Y_c^{\max}$ and $Y_c^{\min}$ as follows:

$$\begin{aligned}Y_c^{\max} &= (\Rightarrow_c^{\max} \cup \{(?ref(m), e) \mid\\ &\qquad m \in \mathcal{M}_s, e \neq ?ref(m), (\$m, e) \in \Rightarrow_c^{\max}\})^{-} \qquad (9)\\ Y_c^{\min} &= (\Rightarrow_c^{\min} \cup \{(?ref(m), e) \mid\\ &\qquad m \in \mathcal{M}_s, e \neq ?ref(m), (\$m, e) \in \Rightarrow_c^{\max}\})^{-} \qquad (10)\end{aligned}$$

The first set is the projected relation of the transitive closure (reduction) of $\Rightarrow$ on the set of events of peer c. The second set adds the additional constraints so that only the receive event $?ref(m)$ of the reply message of a synchronous message m is the direct successor of the preparation event $\$m$. From Theorem 2, we can find $\Rightarrow_c$ between $Y_c^{\max}$ and $Y_c^{\min}$ to synthesize state machines those are strong realization.

Algorithm 1: Extended CSCB method

```
1  Construct an acyclic relation $\Rightarrow$ on the set of events.
2  foreach peer $c$ do
3      Derive $Y_c^{\max}$ and $Y_c^{\min}$ from $\Rightarrow$.
4      foreach acyclic relation $\Rightarrow_c$ in $\mathfrak{A}_c$ do
5          Construct MMG $N$ from $\Rightarrow_c$.
6          Construct set $\mathcal{N}$ of CMMGs from $N$.
7          foreach $N' \in \mathcal{N}$ do
8              Construct a state machine from $N'$.
9              Evaluate the state machine, and update Pareto
               frontier.
10 Output Pareto efficient state machines.
```

Algorithm 2: Constructing an MMG [6]

```
   Input: $\Rightarrow_c$
   Output: MMG $N = (P, T, F, G, A)$
1  begin
2      $P \leftarrow \{p_{(e_1,e_2)} \mid (e_1, e_2) \in \Rightarrow_c\} \cup \{p_s, p_e\}$ ;
3      $T \leftarrow \{t_e \mid e \in \Sigma_c\} \cup \{t_{\mathrm{init}}, t_{\mathrm{end}}\}$ ;
4      $F \leftarrow \{(t_{e_1}, p_{(e_1,e_2)}), (p_{(e_1,e_2)}, t_{e_2}) \mid (e_1, e_2) \in \Rightarrow_c\}$
5          $\cup\{(p_s, t_{\mathrm{init}}), (t_{\mathrm{end}}, p_e)\}$ ;
6      $\forall t \in T : G(t) \leftarrow \emptyset$ ;
7      $\forall e \in \Sigma_c : A(t_e) \leftarrow e; A(t_{\mathrm{init}}) = A(t_{\mathrm{end}}) = \epsilon$ ;
```

3.2.2 Algorithm

The pseudo-code of the extended CSCB method is shown in Algorithm 1. We use Petri nets in the algorithm; please see Appendix B for Message Petri Net (MMG) and so on. The algorithm synthesizes Pareto efficient state machines. Our objective is synthesizing Pareto efficient state machines and is not choosing the best one. The best one should be chosen among them by a designer.

At line 1, the acyclic relation $\Rightarrow$ is constructed by (7). At line 3, two acyclic relations $Y_c^{\max}$ and $Y_c^{\min}$ are derived from $\Rightarrow$ by (9) and (10), respectively.

At line 4, we enumerate all relations between $Y_c^{\min}$ and $Y_c^{\max}$. Let denote the power set of a set A by $\mathcal{P}(A)$. The candidate set $\mathfrak{A}_c$ of $\Rightarrow_c$ is given as follows:

$$\mathfrak{A}_c = \{\Rightarrow_c^{\min} \cup r \mid r \in \mathcal{P}(Y_c^{\max} \setminus Y_c^{\min})\} \tag{11}$$

At line 5, for each acyclic relation $\Rightarrow_c \in \mathfrak{A}_c$, an MMG is constructed by using Algorithm 2.

Algorithm 3: Constructing CMMGs

```
   Input: MMG $N$
   Output: Set $\mathcal{N}$ of CMMGs
 1 begin
 2     Calculate set $U$ of bridges in $N$.
 3     $\mathcal{N} \leftarrow \emptyset$;
 4     cutBridges($N$, $U$, $\mathcal{N}$);
 5 cutBridges($N$, $U$, $\mathcal{N}$)
 6     if $U = \emptyset$ then
 7         Separate fork and join transitions.
 8         Find one-to-one correspondence.
 9         Insert dummies after receiving reply.
10         $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$;
11     foreach bridge $u \in U$ do
12         Cut bridge $u$ in $N$ (let $N'$ be the new one).
13         Calculate set $U'$ of bridges in $N'$.
14         cutBridges($N'$, $U'$, $\mathcal{N}$);
```

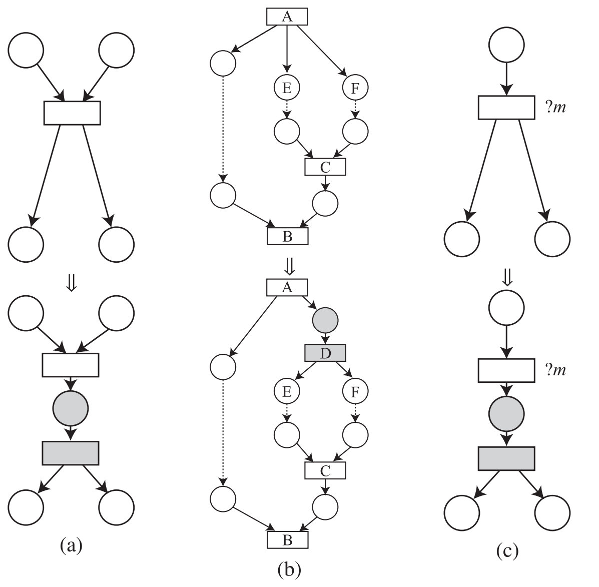

Fig. 1 (a) Separating a fork and join transition, (b) finding one-to-one correspondence, and (c) inserting dummy transition after receiving reply.

Algorithm 4: Cutting a bridge

```
  Input: MMG $(P, T, F, G, A)$
  Input: bridge $b = (n_1, n_2, \ldots, n_{r-1}, n_r)$
1 $F \leftarrow F \setminus \{(n_1, n_2), (n_{r-1}, n_r)\}$;
2 $G(n_3) \leftarrow G(n_3) \cup \{n_1\}$;
3 if $r = 3$ then
4     $P \leftarrow P \setminus \{n_2\}$;
5 else
6     $G(n_r) \leftarrow G(n_r) \cup \{n_{r-2}\}$;
7     $F \leftarrow F \cup \{(t_{\mathrm{init}}, n_2), (n_{r-1}, t_{\mathrm{end}})\}$;
```

At line 6, a set of CMMGs is constructed by using Algorithm 3. In the algorithm, U represents the set of bridges in the MMG. The procedure `cutBridges` is called recursively. The `cutBridges` takes three parameters: an MMG N, the set U of bridges in N, and the set $\mathcal{N}$ of CMMGs.

Algorithm 5: Converting CMMG to a state machine [6]

```
   Input: CMMG $(P, T, F, G, A)$
   Output: State machine $(V, R, r^t, \Theta, \Phi, E, C, B)$, Attribute $\mathcal{A}$
 1 begin
 2     $T \leftarrow T \backslash \{t \mid Empty(t), |\bullet t| = |t \bullet| = 1\}$;
 3     $\mathcal{A} \leftarrow \{fired_t \mid t \in \bigcup_{t' \in T} G(t')\}$;
 4     $E \leftarrow \{Event(t) \mid t \in T\}$;
 5     $C \leftarrow \{Constraint(t) \mid t \in T\}$;
 6     $B \leftarrow \{Behavior(t) \mid t \in T\}$;
 7     $V \leftarrow \emptyset$; $R \leftarrow \emptyset$; $r^t \leftarrow$ new Region();
 8     RNG($p_s, r^t, p_e$);
 9 RNG($p_s, r, p_e$)
10     $t \leftarrow p_s\bullet$; $t_e \leftarrow \bullet p_e$;
11     while $Empty(t_e) \wedge |\bullet t_e| = 1$ do
12         $t_e = \bullet\bullet t_e$;
13     while $Empty(t) \wedge |t \bullet| = 1$ do
14         $t = t \bullet\bullet$;
15     $ip \leftarrow$ new InitialPseudoState(); $\Theta(ip) \leftarrow r$;
16     if $Event(t) = Constraint(t) = \varepsilon$ then
17         $s \leftarrow ip$;
18     else
19         $s \leftarrow$ new SimpleState(); $\Theta(s) \leftarrow r$;
20         new Transition $(ip, \tau, \varepsilon, \varepsilon, s)$;
21     while $t \neq null$ do
22         $ev \leftarrow Event(t)$; $const \leftarrow Constraint(t)$;
           $beh \leftarrow Behavior(t)$;
23         if $t = t_e$ then
24             $s' \leftarrow$ new FinalState();
25             new Transition $(s, ev, const, beh, s')$;
26             return ;
27         if $Empty(t) \wedge |t \bullet| = 1$ then
28             $t \leftarrow t \bullet\bullet$; continue;
29         if $|t \bullet| \geq 2$ then
30             $s' \leftarrow$ new CompositeState(); $\Theta(s') \leftarrow r$;
31             forall the $p' \in t\bullet$ do
32                 $r' \leftarrow$ new Region(); $\Theta(r') \leftarrow s$;
33                 $p'' \leftarrow \bullet FJ(t) \cap Succ(p')$;
34                 RNG($p', r', p''$);
35             $t \leftarrow FJ(t)$ ;
36         else
37             if $A(t) \in \{\$m \mid m \in \mathcal{M}_s\}$ then $t \leftarrow t \bullet\bullet$;
38             $s' \leftarrow$ new SimpleState(); $\Theta(s') \leftarrow r$;
39             $t \leftarrow t \bullet\bullet$;
40         new Transition $(s, ev, const, beh, s')$;
41         $s \leftarrow s'$;
```

If U is empty, a CMMG can be obtained by applying the following three transformation rules (Fig. 1):

1. separating fork and join transitions,
2. finding one-to-one correspondence, and
3. inserting dummy transition after receiving reply.

If U is not empty, a bridge u is cut by using Algorithm 4. Let N' be the new MMG obtained by cutting bridge u. We calculate the set U' of bridges in N' and call `cutBridges` recursively. Note that the set of bridges is finite and the number of bridges monotonically decreases. Therefore, the depth and width of the recursive call is finite.

For each CMMG N', Algorithm 5 is executed to construct a state machine. In the algorithm, $fired_t$ is a Boolean variable and it is added as an attribute of the corresponding class. If $A(t)$ is a callee's event, then $Event(t)$ is the event; otherwise, $Event(t) = \tau$. $Constraint(t)$ returns the guard conditions as follows:

$$Constraint(t) = \begin{cases} \bigwedge_{t \in G(t)} fired_t & \text{if } G(t) \neq \emptyset, \text{ and} \\ \varepsilon & \text{otherwise,} \end{cases}$$

where ε stands for empty expression. $Behavior(t)$ yields the following effect:

$$Behavior(t) = \begin{cases} c.m() & \text{if } A(t) = \$m, \\ \text{send } m \text{ to } c & \text{if } A(t) = !m, m \in \mathcal{M}_{aop}, \\ \text{reply to } m & \text{if } A(t) = !m, m \in \mathcal{M}_{rep}, \\ fired_t = true & \text{if } fired_t \in \mathcal{A}, \text{ and} \\ \varepsilon & \text{otherwise,} \end{cases}$$

where the first three expressions show a synchronous call, an asynchronous call, and a reply to a synchronous call in cbUML and c is the callee service. Note that the first three conditions and the fourth condition are not alternatives. The predicate $Empty(t)$ is true if and only if $Event(t) = \tau$, $Constraint(t) = \varepsilon$, and $Behavior(t) = \varepsilon$. The "new" expression shows that a new vertex or region is generated, and it is added to V or R.

Procedure RNG(p_s, r, p_e) constructs a state machine in region r for a subnet between places p_s and p_e. Lines from 11 to 14 eliminate the transitions that do not have any function. Because a transition from an initial pseudo-state cannot have any triggers and guards [4], a simple state is inserted (lines 19 and 20). Lines 21 to 41 generate states and transitions. When the end of the region is reached, a final state is generated (lines 23 to 26). Lines 27 and 28 also eliminate the transitions that do not have any function. If a transition has more than one output place, procedure RNG(·) is called recursively (lines 29 to 35). $Succ(p')$ in line 33 returns the set of successors of place p'. As shown in Fig. A· 2, calling a synchronous operation and receiving its reply message is represented by a single transition in cbUML; therefore, the succeeding transition that receives the reply message is skipped at line 37.

State machines are evaluated in terms of intelligibility relation $\prec$, and Pareto frontier is updated.

3.3 CSCB Tools

We have implemented the extended CSCB method as a plug-in of Rational Software Architect (RSA), a UML modeling tool, and presented in [9]. We use RSA as the editor for communication diagrams and the repository to store synthesized state machines. One can download the plug-in from the author's page[†].

4. Example

An example choreography is shown in Fig. 2, which is taken

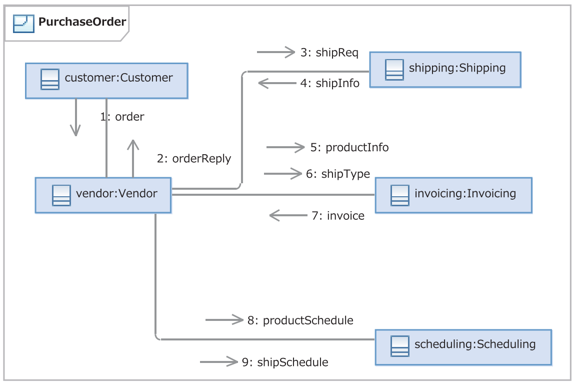

Fig. 2 Purchase Order System

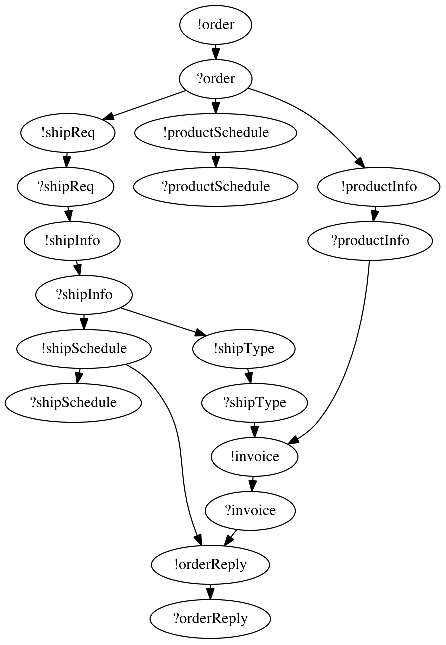

Fig. 3 $\Rightarrow$ of the purchase order system

from BPEL-WS 2.0 specification [10][††]. The system is composed of five services: `Customer`, `Vendor`, `Shipping`, `Invoicing`, and `Scheduling`. When `Vendor` receives `order` from `Customer`, it sends requests to `Shipping`, `Invoicing`, and `Scheduling`.

The request to `Shipping` is `shipReq`, and the reply from `Shipping` is `shipInfo`. The requests to `Invoicing` are `productInfo` and `shipType`; the reply from `Invoicing` is `invoice`. `Vendor` sends `productInfo` after receiving `order`; it sends `shipType` after receiving `shipInfo`. `Invoicing` does internal process to make an invoice after receiving `productInfo` and `shipType`; then it sends `invoice` as a reply. The request to `Scheduling` are `productSchedule` and `shipSchedule`. `Vendor`

[†]http://is.eei.eng.osaka-u.ac.jp/miyamoto/index.php?CSCB(Eng)

[††]In the specification, many elements, such as variables, link types, and data handling, are used. However, considering them is out of the scope of the CRP.

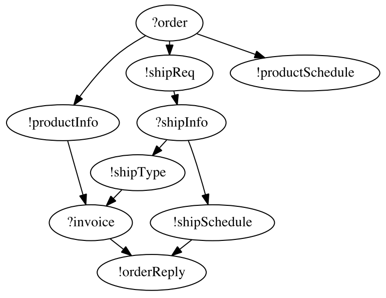

Fig. 4 $\Rightarrow^{\max}_{\texttt{Vendor}}$ of the purchase order system

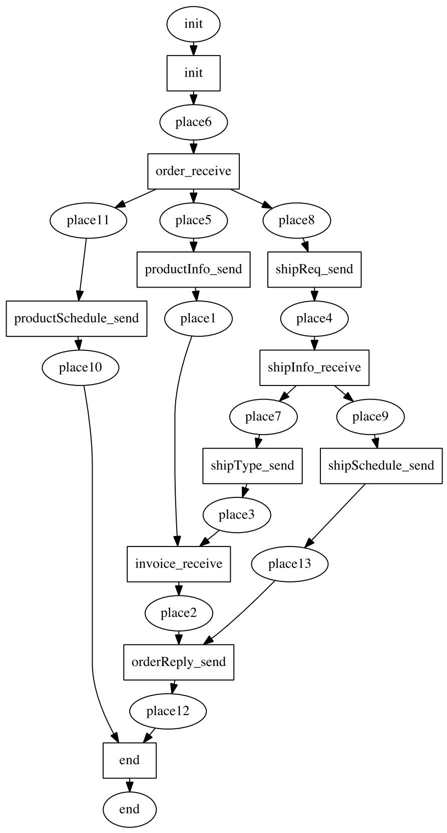

Fig. 5 The MMG of `Vendor` constructed form $Y^{\max}_{\texttt{Vendor}}$

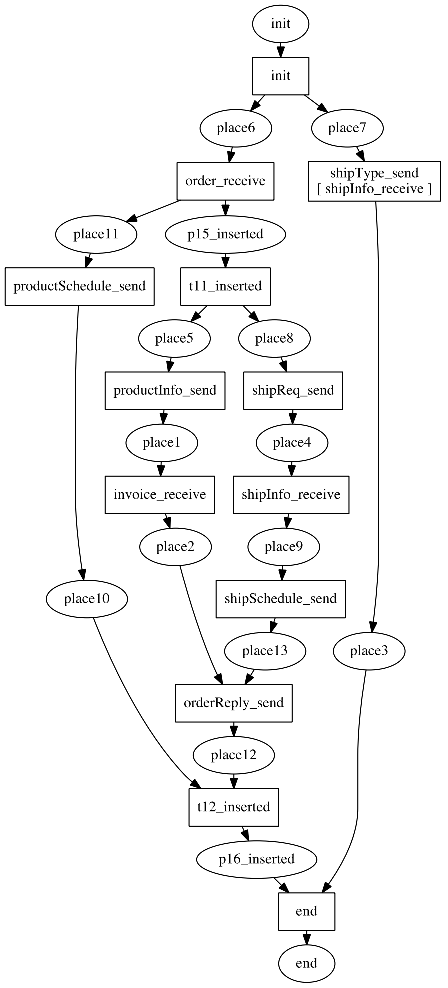

Fig. 6 The CMMG of `Vendor` transformed from the MMG in Fig. 5

sends `productSchedule` after receiving `order`; it sends `shipSchedule` after receiving `shipInfo`. `Vendor` sends `orderReply` to `Customer` after receiving `shipInfo` and `invoice` and sending `shipSchedule`. The acyclic relation $\Rightarrow$ on events is shown in Fig. 3.

Let us synthesize state machines for `Vendor`.

First, we synthesize the state machine by the CSCB method in [6]. The CSCB method uses $Y^{\max}_{\texttt{Vendor}}$, shown in Fig. 4, as the relation $\Rightarrow_{\texttt{Vendor}}$. Algorithm 2 constructs the MMG in Fig. 5. The MMG has a T-T bridge from `shipInfo_receive` to `invoice_receive`. Algorithm 4 cuts the bridge by moving the segment from `place7` to `place3` between two transitions, `init` and `end`. Figure 6 represents the CMMG. Algorithm 5 constructs the state machine in Fig. 7.

In the state machine, the composite state V1 has two regions. But, the transition from state V3 can occur when the Boolean variable `shipInfo_receive` is true. This variable becomes true in the state transition from V13 to V14. The main flow of `Vendor` is represented in the another region in V1. Three regions in V7 and V11 correspond to the three concurrent processes.

Next, we synthesize the state machine by the extended CSCB method. The extended CSCB method uses acyclic relations $Y^{\max}_{\texttt{Vendor}}$ and $Y^{\min}_{\texttt{Vendor}}$, which are shown in Figs. 4 and 8, respectively. We have eight relations in $\mathfrak{A}_{\texttt{Vendor}}$ because

$$\begin{aligned}Y^{\max}_{\texttt{Vendor}} \setminus Y^{\min}_{\texttt{Vendor}} = \{&(?\text{productInfo}, ?\text{invoice}),\\ &(!\text{shipReq}, ?\text{shipInfo}),\\ &(!\text{shipType}, ?\text{invoice})\}.\end{aligned}$$

From these relations, twelve state machines are constructed. Table 1 shows the value of intelligibility metrics. The values in column No. is the number of a relation and a bridge. The state machine in Fig. 7 is 8-1. The value of state machine 7-1 is smaller than or equal to the value of other state machines for all metrics and smaller than at least one metric. The state machine 7-1 is the most intelligible one

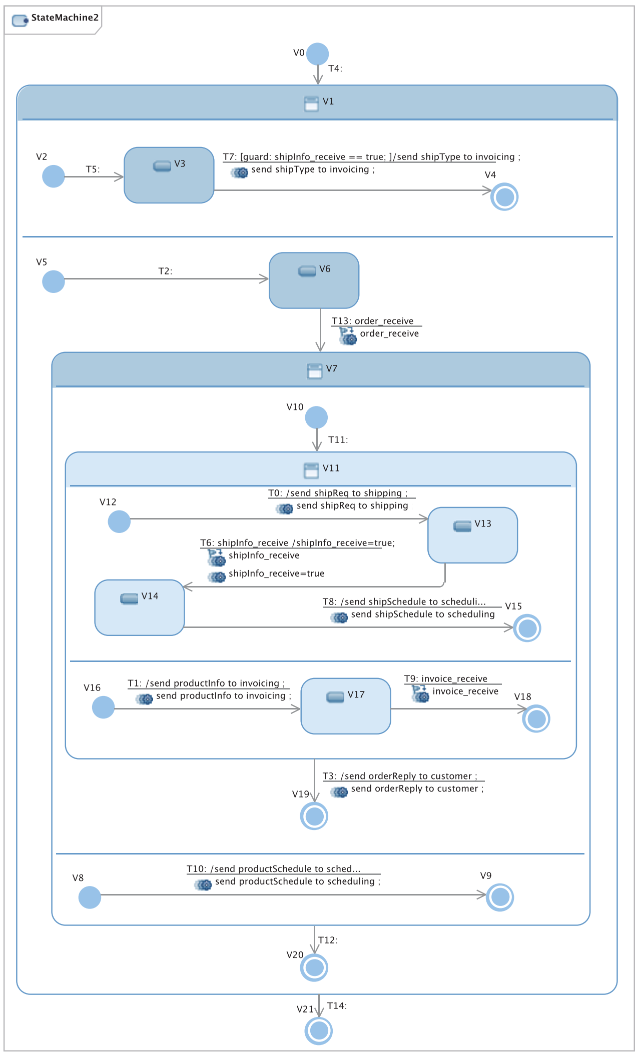
StateMachine2
V0
T4:
V1
V2
T5:
V3
T7: [guard: shipInfo_receive == true;]/send shipType to invoicing ;
send shipType to invoicing ;
V4
V5
T2:
V6
T13: order_receive
order_receive
V7
V10
T11:
V11
T0: /send shipReq to shipping ;
send shipReq to shipping
V12
V13
T6: shipInfo_receive /shipInfo_receive=true;
shipInfo_receive
shipInfo_receive=true
V14
T8: /send shipSchedule to scheduli...
send shipSchedule to scheduling
V15
V16
T1: /send productInfo to invoicing ;
send productInfo to invoicing ;
V17
T9: invoice_receive
invoice_receive
V18
T3: /send orderReply to customer ;
send orderReply to customer ;
V19
T10: /send productSchedule to sched...
send productSchedule to scheduling ;
V8
V9
T12:
V20
V21
T14:

Fig. 7 The state machine of `Vendor` constructed from the CMMG in Fig. 6

Table 1 NSS, NT, NG, ND, and NP of state machines of `Vendor`

No.	NSS	NT	NG	ND	NP
1-1	6	16	1	3	9
2-1	5	16	1	3	8
2-2	6	15	1	3	8
3-1	5	16	1	3	8
3-2	6	15	1	3	8
4-1	3	14	0	3	8
5-1	6	14	1	3	7
6-1	7	16	2	3	8
6-2	7	16	2	3	8
6-3	4	15	1	3	8
7-1	3	11	0	3	7
8-1	5	15	1	4	7

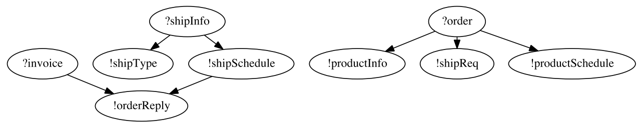

Fig. 8 $\Rightarrow^{\min}_{\texttt{Vendor}}$ of the purchase order system

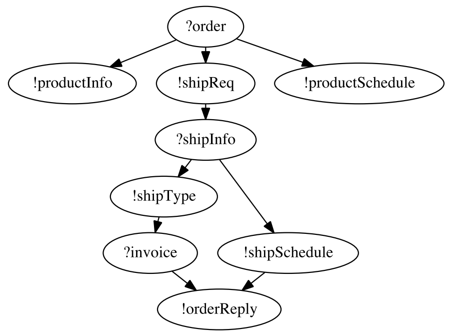

Fig. 9 $\Rightarrow_{\texttt{Vendor}}$ from that the best state machine of `Vendor` is synthesized

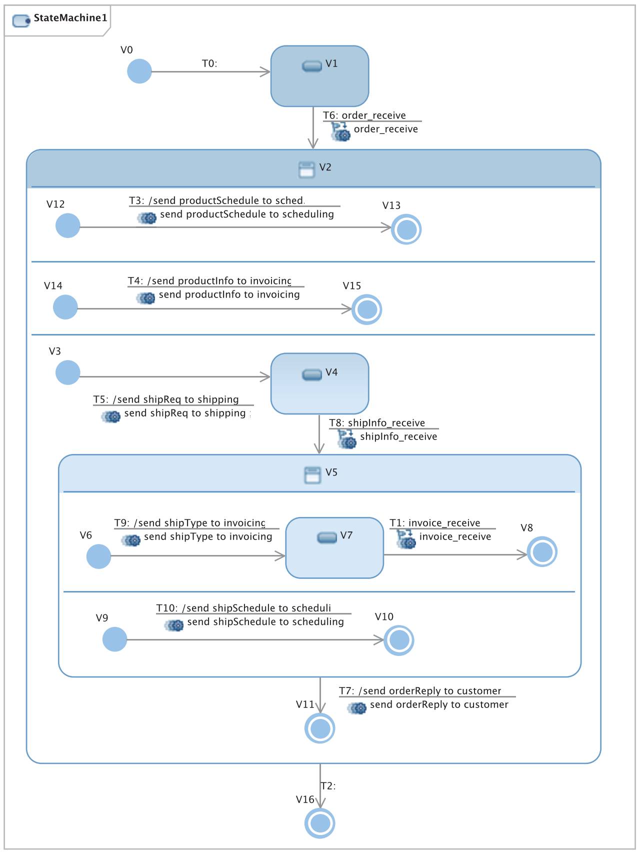
StateMachine1
V0
T0:
V1
T6: order_receive
order_receive
V2
V12
T3: /send productSchedule to sched...
send productSchedule to scheduling
V13
V14
T4: /send productInfo to invoicing
send productInfo to invoicing
V15
V3
V4
T5: /send shipReq to shipping
send shipReq to shipping
T8: shipInfo_receive
shipInfo_receive
V5
V6
T9: /send shipType to invoicing
send shipType to invoicing
V7
T1: invoice_receive
invoice_receive
V8
V9
T10: /send shipSchedule to scheduli...
send shipSchedule to scheduling
V10
T7: /send orderReply to customer
send orderReply to customer
V11
T2:
V16

Fig. 10 State machine of `Vendor` by the extended CSCB method

among the twelve state machines. In Figs. 9 and 10, the relation and the best state machine are shown. The state machine behaves as follows. When `Vendor` receives `order`, it sends `productSchedule` to `Scheduling`, `productInfo` to `Invoicing`, and `shipReq` to `Shipping`. After that, it receives `shipInfo`, then it sends `shipSchedule` to `Shipping` and `shipType` to `Invoicing`. Finally, when it receives `invoice`, it sends `orderReply` to `Customer`.

Comparing two state machines in Figs. 7 and 10, the latter is simpler and, thus, more intelligible than the former. In the former, the transition from state V3 has a guard and

Table 2 Intelligibility comparison

method	NSS	NT	NG	ND	NP
projection	30	61	0	1	4
CSCB	5	15	1	4	7
extended CSCB	3	11	0	3	7

the state transition can occur after occurring the state transition from V13 to V14. Thus, the former has implicit control flow. On the contrary, the latter does not have any guards and the three region in the composite state V2 represents the three concurrent processes to three services. We think the latter is easier to understand its behavior.

Let us compare the proposed method with other methods: projection [2] and CSCB [6] methods. Table 2 shows the comparison result.

The projection method is equivalent to constructing the state space, therefore, NSS and NT become large, NG is zero, and ND is one. Compared to the projection method, CSCB and extended CSCB methods succeeded in reducing NSS and NT. However, state machines by the projection and the extended CSCB methods are not comparable in terms of intelligibility because the projection method is better on ND and NP and the extended CSCB method is better on NSS and NT. Therefore, these state machines are members of the Pareto frontier.

The CSCB method uses $Y_c^{\max}$ as $\Rightarrow_c$. Because $Y_c^{\max} \in \mathfrak{A}_c$, the state machine synthesized by the CSCB method is synthesized also by the extended CSCB method. Thus, the extended CSCB method always synthesizes more intelligible state machines than the CSCB method.

We may synthesize other state machines those are members of the Pareto frontier. For example, we can synthesize a state machine with ND=2 by flattening the composite state V5 of the state machine in Fig. 10. The extended CSCB method does not synthesize every state machine in the Pareto frontier. However, extending the extended CSCB method to synthesize flattened state machines is not difficult.

5. Conclusion

In this paper, we discussed an approach to CRP, considering the intelligibility of state machines. A set of metrics for synthesized state machines is introduced and an algorithm of synthesizing Pareto efficient state machines for the CRP is proposed. Currently, our algorithm uses the brute force method. When the number of messages get larger, we may need any efficient search strategy.

A method for synthesizing state machines from multiple communication diagrams is also left for future research.

References

[1] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall Professional Technical Reference, 2005.

[2] T. Bultan and X. Fu, "Specification of realizable service conversations using collaboration diagrams," Service Oriented Computing and Applications, vol.2, no.1, pp.27–39, April 2008.

[3] J. Su, T. Bultan, X. Fu, and X. Zhao, "Towards a theory of web service choreographies," Proc. 4th Intl. Conf. on Web Services and Formal Methods, pp.1–16, Sept. 2007.

[4] Object Management Group, "OMG Unified Modeling Language (OMG UML), superstructure," Aug. 2011. (accessed Oct. 31, 2013).

[5] J.A. Cruz-Lemus, M. Genero, and M. Piattini, "Metrics for UML statechart diagrams," in Metrics for Software Conceptual Models, ed. M. Genero, M. Piattini, and C. Calero, pp.237–272, Imperial College Press, 2005.

[6] T. Miyamoto, Y. Hasegawa, and H. Oimura, "An approach for synthesizing intelligible state machine models from choreography using petri nets," IEICE Trans. Inf. & Syst., vol.E97.D, no.5, pp.1171–1180, May 2014.

[7] T. Murata, "Petri nets: Properties, analysis and applications," Proc. IEEE, vol.77, no.4, pp.541–580, April 1989.

[8] T. Miyamoto, "Choreography realization by re-constructible decomposition of acyclic relations," IEICE Trans. Inf. & Syst., vol.E99.D, no.6, pp.1420–1427, 2016.

[9] T. Miyamoto, "CSCB tools: A tool to synthesize pareto optimal state machine models from choreography using Petri nets," International Workshop on Petri Nets and Software Engineering, pp.335–340, 2016.

[10] OASIS, "Web services business process execution language version 2.0," 2006.
http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html

[11] J. Esparza and M. Silva, "Circuits, handles, bridges and nets," in Advances in Petri Nets 1990, ed. G. Rozenberg, Lecture Notes in Computer Science, vol.483, pp.210–242, Springer, 1991.

Appendix A: Operational Semantics of State Machines

Due to space limitations, the details of the operational semantics of state machines are omitted. A state machine has a message pool, and its state is defined by a consistent set of active states, a set of suspended regions, a set of messages in the message pool, and values of the attributes. A transition may fire when the originating vertex is active, the message of the trigger event is in the message pool or is the completion event, and the guard is true. When the transition fires, the originating vertex and its descendants are inactivated, the message is removed from the message pool, the effect is executed, and the target vertex and initial pseudostates in the first descendant regions are activated. The steps for synchronous calls and asynchronous calls are explained with examples.

Figure A· 1 shows the execution steps of an asynchronous call. In the figure, a state, a transition, and a region is represented by a round-cornered rectangle, an arrow, and a rectangle with dashed lines. However, in Fig. A· 1, regions are omitted for simplification. The gray states are active. When state machine sm1 transitions from state s11 to state s12 due to the completion event, an asynchronous call is executed. At this time, the send event $!m$ occurs and message m is added to the message pool of sm2. The state machine sm2 transitions from state s21 to state s22, consuming message m due to the receive event $?m$.

Figure A· 2 shows the execution steps of a synchronous call. A synchronous call is executed in sm1. At this time, the preparation event $\$m$ occurs in sm1, and the region that contains the transition is suspended, where the suspended region is represented by the gray region. Moreover, mes-

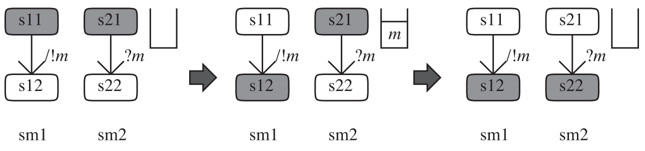

Fig. A· 1 Steps for an asynchronous call

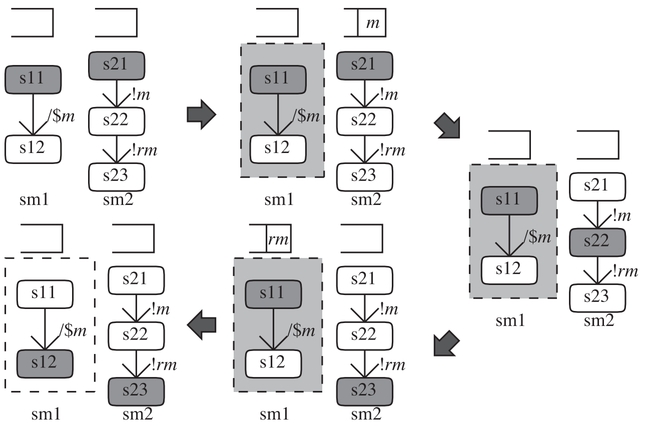

Fig. A· 2 Steps for a synchronous call

sage m is added to the message pool of sm2. State machine sm2 transitions from state s21 to s22, consuming message m by the occurrence of the send-receive event $!m$. Next, sm2 sends a reply message rm to sm1 upon transitioning from s22 to s23. At this time, the send event $!rm$ occurs, and message rm is added to the message pool of sm1. Now, sm1 releases the suspended region and transitions from state s11 to state s12, consuming reply message rm by the occurrence of the receive event $?rm$. Note that the receive event $?rm$ does not appear in the state machine because they were using the region-suspend mechanism.

Appendix B: Message Petri Nets

A Petri net [7] is a tuple $N = (P, T, F)$, where P is the set of places, T is the set of transitions, and $F \subseteq P \times T \cup T \times P$ is the flow relation. For $x \in P \cup T$, the set $\{y \in P \cup T \mid (y, x) \in F\}$ is called the preset of x and denoted by $\bullet x$. Similarly, the set $\{y \mid (x, y) \in F\}$ is called the postset of x and it is denoted by $x\bullet$. For set X, $\bullet X = \cup_{x \in X} \bullet x$ and $X\bullet = \cup_{x \in X} x\bullet$.

A place $p \in P$ is called a *source* place and a *sink* place when $\bullet p = \emptyset$ and $p\bullet = \emptyset$, respectively. Similarly, a transition $t \in T$ is called a source transition and a sink transition when $\bullet t = \emptyset$ and $t\bullet = \emptyset$, respectively. A transition t is called a *fork* transition and a *join* transition when $|t \bullet| > 1$ and $|\bullet t| > 1$, respectively. The sets of join transitions and fork transitions are denoted by T_{join} and T_{fork}, respectively. Under the standard definition of Petri nets, if $\forall p \in P : |\bullet p| = 1$ and $|p \bullet| = 1$, then the Petri net is called a *marked graph*. In this paper, we relax the condition as follows: $\forall p \in P : |\bullet p| \leq 1$ and $|p \bullet| \leq 1$.

Definition 9 (MMG): A message marked graph (MMG) is a tuple $N = (P, T, F, G, A)$, where the underlying Petri net (P, T, F) satisfies the following conditions:

1. N is acyclic,
2. Only one source place p_s and one sink place p_e exist,
3. No source transitions and sink transitions exist, and
4. $|p_s \bullet| = 1$, $|\bullet p_e| = 1$, and $\forall p \in P \backslash \{p_s, p_e\} : [|\bullet p| = 1, |p \bullet| = 1]$.

$G : T \to 2^T$ is a firing constraint and the partial function $A : T \to \Sigma \cup \{\epsilon\}$ assigns an event for each transition[†].

Definition 10 (Bridge [11]): Let $N = (P, T, F)$, and $N_1 = (P_1, T_1, F_1)$ and $N_2 = (P_2, T_2, F_2)$ be subnets of N. An elementary path[††] $\pi = (n_1, \ldots, n_r), r \geq 2$ is a *bridge* from N_1 to N_2 if and only if $\pi \cap (P_1 \cup T_1) = \{n_1\}$ and $\pi \cap (P_2 \cup T_2) = \{n_r\}$. When n_1 and n_r are transitions, the bridge is called a T-T bridge. P-T, T-P, and P-P bridges are defined in a similar way.

Note that any MMG has only T-T bridges, no P-T, T-P, or P-P bridge is in the MMG, because the cardinality of preset or postset of any place is at most one.

Definition 11 (Parallel Path): For two elementary paths $\pi_1 = (n_1, \ldots, n_r)$ and $\pi_2 = (n_1, \ldots, n_r)$, if $\pi_1 \cap \pi_2 = \{n_1, n_r\}$, then π_1 and π_2 are called *parallel paths*.

For a transition $t \in T$ in a T-T bridge-free MMG, $FJ(t) \subseteq T$ (resp. $JF(t) \subseteq T$) is the set of terminal (resp. starting) transitions of parallel paths starting from (resp. terminating at) t.

Definition 12 (Convertible MMG): An MMG is called a *convertible MMG (CMMG)* if the following conditions hold:

1. $|T_{fork}| = |T_{join}|$,
2. $T_{fork} \cap T_{join} = \emptyset$,
3. For any parallel paths η_1 and η_2, there exist no bridge from one to the other,
4. If $A(t) = \$m$, then $t \bullet \bullet = \{t'\}$, $A(t') = ?ref(m)$, and
5. If $A(t) = ?m$ and $m \in \mathcal{M}_{rep}$, then $|t \bullet| = 1$.

[†]$A(t) = \epsilon$ means that no event is assigned to t.

[††]A path is called elementary if no vertices appear twice or more in it.

Toshiyuki Miyamoto received his B.E. and M.E. degrees in electronic engineering from Osaka University, Japan in 1992 and 1994, respectively. Moreover, he received Dr. of Eng. degree in electrical engineering from Osaka University, Japan in 1997. From 2000 to 2001, he was a visiting researcher in Department of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA. Currently, he is an Associate Professor with the Division of Electrical, Electronic and Information Engineering, Osaka University. His areas of research interests include theory and applications of concurrent systems and multi-agent systems. He is a member of IEEE, SICE, and ISCIE.