IEICE TRANS. INF. & SYST., VOL.E99–D, NO.6 JUNE 2016

PAPER Special Section on Formal Approach

Choreography Realization by Re-Constructible Decomposition of Acyclic Relations*

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

SUMMARY For a service-oriented architecture-based system, the problem of synthesizing a concrete model (i.e., a behavioral model) for each peer configuring the system from an abstract specification—which is referred to as choreography—is known as the choreography realization problem. In this paper, we consider the condition for the behavioral model when choreography is given by an acyclic relation. A new notion called re-constructible decomposition of acyclic relations is introduced, and a necessary and sufficient condition for a decomposed relation to be re-constructible is shown. The condition provides lower and upper bounds of the acyclic relation for the behavioral model. Thus, the degree of freedom for behavioral models increases; developing algorithms for synthesizing an intelligible model for users becomes possible. It is also expected that the condition is applied to the case where choreography is given by a set of acyclic relations.

key words: *SOA, model-based development, communication diagram, state machine, choreography realization problem*

1. Introduction

The internationalization of business activities and information technology in companies has intensified competition among them. 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. It is not easy for designers to design a concrete model directly from requirements because huge gaps exist between requirements and concrete models. However, defining an abstract specification is relatively simple. Therefore, if we can automatically synthesize a concrete model from a well-written abstract specification, the designer's workload would decrease significantly and product quality would improve.

In the software engineering literature, several studies have synthesized a concrete model from an abstract specification. Harel et al. proposed a methodology for synthesizing statechart models from scenario-based requirements [2]. Whittle et al. proposed a methodology for synthesizing hierarchical state machine models from expressive scenario descriptions [3]. Liang et al. defined a set of comparison criteria and surveyed 21 different synthesis approaches [4].

In SOA, the problem of synthesizing a concrete model from an abstract specification is known as the choreography realization problem (CRP) [5], [6]. 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 [7] 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. However, this restriction should be removed, and to do so, the notion of concatenation of acyclic relations in [8] could be used.

Bultan and Fu formally studied the CRP [6]. 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. Furthermore, they adopt the semantics that message send and receive events for a synchronous call occur sequentially. Under these semantics, the UML specification that "the execution of the call operation action waits until the execution of the invoked behavior completes and a reply transmission is returned to the caller" [7] cannot be represented.

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 [9]. 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

Manuscript received June 29, 2015.
Manuscript revised October 19, 2015.
Manuscript publicized May 2, 2016.

[†]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 26330083.

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

DOI: 10.1587/transinf.2015FOP0001

the number of do-activities (NA).

Miyamoto et al. proposed a method for synthesizing hierarchical state machines from the choreography given in communication diagrams called the Construct State-machine Cutting Bridges (CSCB) method [10]. In the method, dependency relations among sent and received message events are represented by Petri nets [11]; state machines are then synthesized. Because state machines synthesized by the CSCB method are hierarchical, they are more intelligible than state machines synthesized with projection mapping.

The CSCB method assumes that choreography is defined by only one communication diagram. However, this is restrictive because a system must work well in a variety of cases. For example, in an e-commerce system, one communication diagram defines the process when the ordered item is in stock, and another one defines the process when it is out of stock. In the latter case, the shop must order the item to a wholesaler. Thus, the communication diagrams must be different. Thus, we have to remove the restriction. During the analysis process in [10], we found that the CSCB method and the projection mapping method synthesize unnecessary complex state machines in some examples. In both methods, dependency relations among events for each peer are derived, and then state machines are synthesized. A simple question then arises: "What is the necessary and/or sufficient condition for the dependency relation?" If we can use simpler dependency relations, synthesized state machines become simpler. Moreover, the condition is useful to develop the synthesizing algorithm of state machines when choreography is given by a set of communication diagrams.

We can find several studies on reconstruction, decomposition, and/or a combination of acyclic relations [12]–[15]. However, none of these studies can be used for our problem. Thus, we introduce a new notion on the decomposition of acyclic relations and study the CRP in this paper.

In this paper, we consider the condition for the behavioral model when choreography is given by an acyclic relation. In Sect. 2, a new notion called the re-constructible decomposition of acyclic relations is introduced, and a necessary and sufficient condition for a decomposed relation to be re-constructible is shown. In Sect. 3, we define terms for the CRP. In Sect. 4, we describe the CRP and study realizability conditions for choreography using the re-constructibility.

2. Re-Constructible Decomposition

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 called *acyclic*. Hereinafter, we assume that every relation is 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 $\mathcal{R}_c^{\max}$, $\mathcal{R}_c^{\min}$, $\mathcal{R}^{\max}$, and $\mathcal{R}^{\min}$ are defined as follows:

$$\mathcal{R}_c^{\min} = \Sigma_c^2 \cap \mathcal{R}^-, \tag{1}$$
$$\mathcal{R}_c^{\max} = \Sigma_c^2 \cap \mathcal{R}^+, \tag{2}$$
$$\mathcal{R}^{\min} = \mathcal{R}_{\mathrm{com}} \cup (\textstyle\bigcup_c \mathcal{R}_c^{\min}), \text{ and} \tag{3}$$
$$\mathcal{R}^{\max} = \mathcal{R}_{\mathrm{com}} \cup (\textstyle\bigcup_c \mathcal{R}_c^{\max}), \tag{4}$$

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

The following lemma holds wrt $\mathcal{R}^{\min}$ and $\mathcal{R}^{\max}$.

Lemma 1: $\mathcal{R}^{\min+} = \mathcal{R}^{\max+}$.

Proof: From the definition, it is clear that $\mathcal{R}^{\min+} \subseteq \mathcal{R}^{\max+}$.

For any $(e_1, e_2) \in \mathcal{R}^{\max+}$, consider the longest path from e_1 to e_2 on the graph $(\Sigma, \mathcal{R}^{\max+})$. The path must exist on $(\Sigma, \mathcal{R}^{\min+})$. Therefore, $\mathcal{R}^{\max+} \subseteq \mathcal{R}^{\min+}$. □

The following lemmas hold on acyclic relations.

Lemma 2: $\mathfrak{L}(\mathcal{R}_2) \subseteq \mathfrak{L}(\mathcal{R}_1)$ iff $\mathcal{R}_1 \subseteq \mathcal{R}_2^+$.

Proof: If $\mathcal{R}_1 \subseteq \mathcal{R}_2^+$, then any word in $\mathfrak{L}(\mathcal{R}_2)$ does not violate relation $\mathcal{R}_1$. Thus, $\mathfrak{L}(\mathcal{R}_2) \subseteq \mathfrak{L}(\mathcal{R}_1)$.

If $\mathcal{R}_1 \not\subseteq \mathcal{R}_2^+$, there must exist a pair $(e_1, e_2) \in \mathcal{R}_1 \setminus \mathcal{R}_2^+$. $(e_1, e_2) \notin \mathcal{R}_2^+$ implies that $(e_1, e_2) \notin \mathcal{R}_2$. Thus, a word in $\mathfrak{L}(\mathcal{R}_2)$ in which e_2 precedes e_1 exists. That means $\mathfrak{L}(\mathcal{R}_2) \not\subseteq \mathfrak{L}(\mathcal{R}_1)$. □

Note that if $\mathcal{R}_1 \subseteq \mathcal{R}_2$, then $\mathcal{R}_1 \subseteq \mathcal{R}_2^+$. Thus, Lemma 2 means that if $\mathcal{R}_1 \subseteq \mathcal{R}_2$, then $\mathfrak{L}(\mathcal{R}_2) \subseteq \mathfrak{L}(\mathcal{R}_1)$. However, $\mathfrak{L}(\mathcal{R}_2) \subseteq \mathfrak{L}(\mathcal{R}_1)$ does not imply $\mathcal{R}_1 \subseteq \mathcal{R}_2$.

Lemma 3: $\mathfrak{L}(\mathcal{R}_1) = \mathfrak{L}(\mathcal{R}_2)$ iff $\mathcal{R}_1^+ = \mathcal{R}_2^+$.

Proof: $\mathcal{R}_1^+ = \mathcal{R}_2^+ \Leftrightarrow \mathcal{R}_1^+ \subseteq \mathcal{R}_2^+ \wedge \mathcal{R}_2^+ \subseteq \mathcal{R}_1^+ \Leftrightarrow \mathcal{R}_1 \subseteq \mathcal{R}_2^+ \wedge \mathcal{R}_2 \subseteq \mathcal{R}_1^+ \Leftrightarrow \mathfrak{L}(\mathcal{R}_2) \subseteq \mathfrak{L}(\mathcal{R}_1) \wedge \mathfrak{L}(\mathcal{R}_1) \subseteq \mathfrak{L}(\mathcal{R}_2) \Leftrightarrow \mathfrak{L}(\mathcal{R}_2) = \mathfrak{L}(\mathcal{R}_1)$ □

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

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

From Lemma 3, the re-constructibility and the assumption can be checked by the equivalence of transitive closures of relations. In general, enumerating all topological sorts requires exponential complexity [16], but calculating transitive closure requires cubic polynomial complexity [17]. Thus, using transitive closure is a less costly way to check.

From the definition, it is clear that $\mathcal{R}^{\min} \subseteq \mathcal{R}$. Thus, the following lemma holds.

Lemma 4: Dissatisfaction of Assumption 1 implies that $\mathfrak{L}(\mathcal{R}) \subset \mathfrak{L}(\mathcal{R}^{\min})$.

From Lemmas 1 and 3 and Assumption 1, Eq. (5)

holds.

$$\mathfrak{L}(\mathcal{R}^{\max}) = \mathfrak{L}(\mathcal{R}^{\min}) = \mathfrak{L}(\mathcal{R}) \quad (5)$$

The following theorem holds finally.

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

Proof:

$\forall c : \mathcal{R}_c^{\min} \subseteq \mathcal{R}_c \subseteq \mathcal{R}_c^{\max} \Leftrightarrow \mathcal{R}^{\min} \subseteq (\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c) \subseteq \mathcal{R}^{\max}$
$\Rightarrow \mathfrak{L}(\mathcal{R}^{\max}) \subseteq \mathfrak{L}(\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c) \subseteq \mathfrak{L}(\mathcal{R}^{\min})$ ($\because$ Lemma 2)
$\Leftrightarrow \mathfrak{L}(\mathcal{R}) = \mathfrak{L}(\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c)$ ($\because$ (5))

Then, suppose that $\mathfrak{L}(\mathcal{R}^{\max}) \subseteq \mathfrak{L}(\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c) \subseteq \mathfrak{L}(\mathcal{R}^{\min})$. From Lemma 2, $\mathcal{R}^{\min} \subseteq (\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c)^+$ and $(\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c) \subseteq \mathcal{R}^{\max +}$. Thus, $(\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c^{\min}) \subseteq (\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c)^+$ and $(\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c) \subseteq (\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c^{\max})^+$. If $\mathcal{R}_c^{\min} \not\subseteq \mathcal{R}_c$, then there must exist a pair $(e_1, e_2) \in \mathcal{R}_c^{\min} \setminus \mathcal{R}_c$. However, $(\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c^{\min}) \subseteq (\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c)^+$ means the existence of e_3 such that $(e_1, e_3), (e_3, e_2) \in (\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c)^+$. This contradicts the definition of $\mathcal{R}_c^{\min}$. Thus, $\mathcal{R}_c^{\min} \subseteq \mathcal{R}_c$. Similarly, $\mathcal{R}_c \subseteq \mathcal{R}_c^{\max}$. Finally, $\mathcal{R}^{\min} \subseteq (\mathcal{R}_{\text{com}} \cup \bigcup_c \mathcal{R}_c) \subseteq \mathcal{R}^{\max}$ holds. □

3. Preliminaries

3.1 cbUML

Let us introduce a subset of UML called cbUML. The complete set of cbUML is described in [18]. This section shows a simplified version of cbUML, which is sufficient for the discussion of this paper.

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.

3.1.1 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 $ref : \mathcal{M} \to \mathcal{M} \cup \{\text{nil}\}$, such that $\forall m \in \mathcal{M}_{sop} : ref(m) \in \mathcal{M}_{rep}$, $\forall m \in \mathcal{M}_{rep} : ref(m) \in \mathcal{M}_{sop}$, $\forall m \in \mathcal{M}_{aop} : ref(m) = \text{nil}$, and $\forall m \in \mathcal{M}_{sop} \cup \mathcal{M}_{rep} : ref(ref(m)) = m$.

The peers behave differently during interactions depending on the type of message, as follows. In the case of a synchronous call, the caller's execution is suspended until the caller receives a reply from the callee. However, in the case of an asynchronous call, the caller can continue to operate, regardless of the behavior of the callee.

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 define 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:

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

$$\Delta = \{!m \mid m \in \mathcal{M}\}. \quad (7)$$

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

$$\Rightarrow_{\mathcal{M}} = \{(\$m, !m) \mid m \in \mathcal{M}_s\} \cup \{(!m, ?m) \mid m \in \mathcal{M}_a\}. \quad (8)$$

3.1.2 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 [6]. 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 $\sigma \in \mathfrak{C}(\mathcal{CD})$.

3.1.3 State Machines

Definition 5 (State Machine): A state machine is a tuple

$sm = (V, R, r^t, \Theta, \Phi, E, C, B)$, where V is the set of vertices, R is the set of regions, $r^t \in R$ 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. 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 $(V \cup R) \setminus \{r^t\}$ 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. According to the UML specification [7], triggers, guards, and effects are denoted as "$tri[grd]/eff$" in diagrams.

Due to space limitations, the details of the operational semantics of state machines are omitted. They have been developed based on [19], [20] and reported in [18]. 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 pseudo-states in the first descendant regions are activated. The steps for synchronous calls and asynchronous calls are explained with examples.

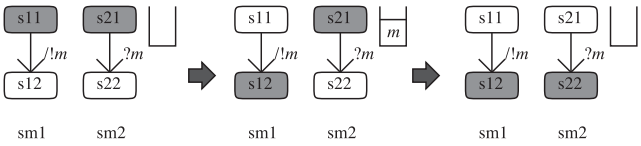

Fig. 1 Steps for an asynchronous call.

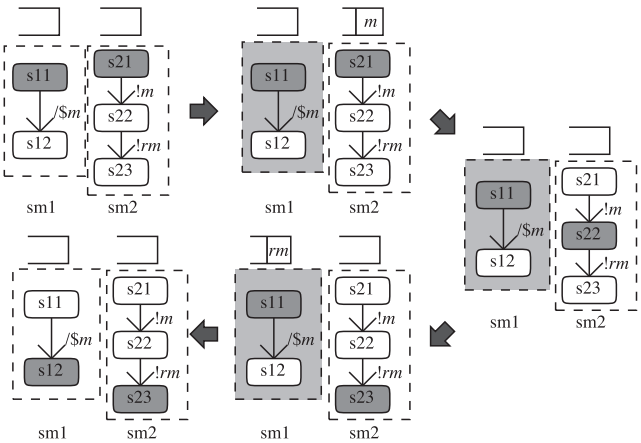

Fig. 2 Steps for a synchronous call.

Figure 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. 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 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, message 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 we are using the region suspend mechanism.

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. 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 $\mathfrak{C}(\mathcal{SM})$.

4. Choreography Realization and Re-Constructibility

4.1 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.

4.2 CSCB Method

We proposed the CSCB method that synthesizes state machines from a communication diagram in [10]. Due to space limitations, the details of the algorithm are omitted here. State machines are synthesized as follows:

1. Construct an acyclic relation $\Rightarrow$ on the set of events.

For each peer c, perform the following steps.

2. Derive an acyclic relation $\Rightarrow_c$ from $\Rightarrow$.
3. Construct a state machine from $\Rightarrow_c$.

Recall that we assume Assumption 1 for $\Rightarrow$.

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:

$$\begin{aligned} \Rightarrow = \; & D \cup \{(?m_1, !m_2) \mid m_1 \in \mathcal{M}_a, m_2 \in \mathcal{M}_a, \Omega(?m_1, !m_2)\} \\ & \cup \{(?m_1, \$m_2) \mid m_1 \in \mathcal{M}_a, m_2 \in \mathcal{M}_s, \Omega(?m_1, \$m_2)\} \\ & \cup \{(!m_1, \$m_2) \mid m_1 \in \mathcal{M}_s, m_2 \in \mathcal{M}_s, \Omega(!m_1, \$m_2)\} \\ & \cup \Rightarrow_{\mathcal{M}} \cup \{(!m, e) \mid m \in \mathcal{M}_s, \Omega(\$m, e)\}, \end{aligned} \tag{9}$$

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:

$$\Rightarrow_{\mathrm{com}} = \; \Rightarrow_{\mathcal{M}} \cup \{(!m, e) \mid m \in \mathcal{M}_s, \Omega(\$m, e)\}, \tag{10}$$

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.

The relation Y_c for a peer c is given as follows:

$$\begin{aligned} Y_c = \Rightarrow_c^{\max} \cup \{ & (?ref(m), e) \mid \\ & m \in \mathcal{M}_s, e \neq ?ref(m), (\$m, e) \in \Rightarrow_c^{\max}\}. \end{aligned} \tag{11}$$

The first set is the projected relation of the transitive closure 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 $\$m$. Next, the acyclic relation $\Rightarrow_c$ for a peer c is obtained by transitively reducing Y_c, as follows:

$$\Rightarrow_c = Y_c^-. \tag{12}$$

In the above procedure, the transitive closure of $\Rightarrow$ is projected to each peer, and a similar procedure is used in other existing studies, such as [6]. However, the derived relation sometimes becomes too restrictive. Theorem 1 shows that the transitive reduction of $\Rightarrow$ is sufficient to derive $\Rightarrow_c$. This paper proposes replacing Y_c, as follows:

$$\begin{aligned} Y_c = \Rightarrow_c^{\min} \cup \{ & (?ref(m), e) \mid \\ & m \in \mathcal{M}_s, e \neq ?ref(m), (\$m, e) \in \Rightarrow_c^{\min}\}. \end{aligned} \tag{13}$$

4.3 Realizability

Let Pro be a mapping that translates a word of acyclic relation $\Rightarrow$ on Σ to a conversation. A conversation $\sigma \in \mathrm{Pro}(\mathfrak{L}(\Rightarrow))$ is obtained by removing non-active events and replacing each active event with the corresponding message from a word $w \in \mathfrak{L}(\Rightarrow)$. The following equation then holds because $\Rightarrow$ defined by Eq. (9) is obtained by just inserting a non-active event on D:

$$\mathfrak{C}(cd) = \mathrm{Pro}(\mathfrak{L}(D)) = \mathrm{Pro}(\mathfrak{L}(\Rightarrow)). \tag{14}$$

Let $\mathcal{R}_c$ be the acyclic relation for peer c. Under the assumption that the state machine that behaves equivalently

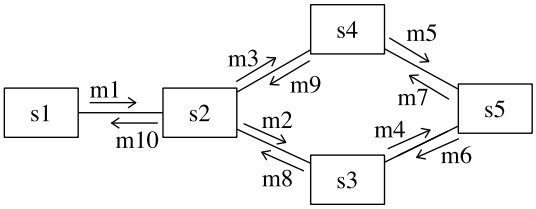

Fig. 3 A choreography with five peers

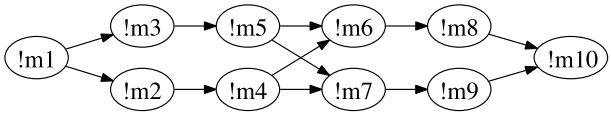

Fig. 4 D of choreography in Fig. 3

to $\mathcal{R}_c$ can be synthesized, the acyclic relation for the system is $\Rightarrow_{\text{com}} \cup(\bigcup_c \mathcal{R}_c)$. Thus, the following equation holds:

$$\mathfrak{C}(\mathcal{SM}) = \text{Pro}(\mathfrak{L}(\Rightarrow_{\text{com}} \cup(\bigcup_c \mathcal{R}_c))). \quad (15)$$

Then we can obtain the following theorem directly from Definition 1 and Eqs. (14) and (15).

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 $c \in \mathcal{C}$, then $\mathcal{SM}$ is a weak realization of cd.

Proof: $\forall c \in \mathcal{C} : \Rightarrow_c^{\min} \subseteq \mathcal{R}_c$ implies that $\Rightarrow_{\text{com}} \cup(\bigcup_c \Rightarrow_c^{\min}) \subseteq \Rightarrow_{\text{com}} \cup(\bigcup_c \mathcal{R}_c)$. From Lemma 2, $\mathfrak{L}(\Rightarrow_{\text{com}} \cup(\bigcup_c \mathcal{R}_c)) \subseteq \mathfrak{L}(\Rightarrow_{\text{com}} \cup(\bigcup_c \Rightarrow_c^{\min})) = \mathfrak{L}(\Rightarrow)$. □

The following sufficient condition for multiple communication diagram cases can be easily obtained from Corollary 3.

Corollary 4: If $\Rightarrow_c^{\min} \subseteq \mathcal{R}_c$ for all $c \in \mathcal{C}$ and $cd \in \mathcal{CD}$, then $\mathcal{SM}$ is a weak realization of $\mathcal{CD}$.

The above condition is too restrictive because the state machines accept only the common behavior of the set of communication diagrams. Although we need to relax the condition, it is beyond the scope of this paper.

On the acyclic relation $\Rightarrow_c$ that is derived by the procedure in Sect. 4.2, the following corollaries hold.

Corollary 5: $\mathcal{SM}$ synthesized from $\Rightarrow_c$ is a weak realization.

Proof: It is obvious that $\Rightarrow_c^{\min} \subseteq \Rightarrow_c$ for all $c \in \mathcal{C}$. From Corollary 3, $\mathcal{SM}$ is a weak realization. □

Corollary 6: When $\mathcal{CD}$ does not use any synchronous message, $\mathcal{SM}$ synthesized from $\Rightarrow_c$ is a strong realization.

Proof: When no synchronous message is used, $\Rightarrow_c^{\min} = \Rightarrow_c$. Therefore, $\Rightarrow_c$ is re-constructible. □

Figure 3 shows choreography for a system composed of five peers. Figure 4 shows the dependency relation on messages; all messages are asynchronous. Peer s2 sends messages m2 and m3 to peers s3 and s4, respectively, after the receive message m1 from peer s1. Peer s5 sends messages m6 and m7 after receiving messages m4 and m5. Peer s2 sends message m10 to peer s1 after receiving messages m8 and m9.

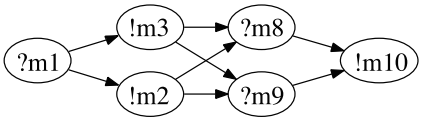

Fig. 5 $\Rightarrow_{s2}$ by the method in [10]

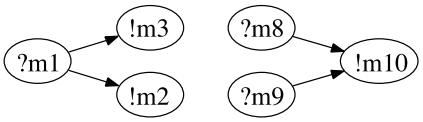

Fig. 6 $\Rightarrow_{s2}$ by the method in this paper

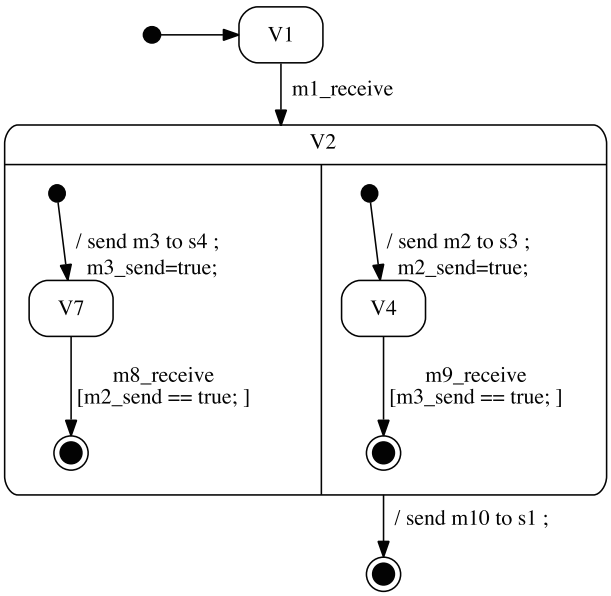

Fig. 7 State machine synthesized from $\Rightarrow_{sm2}$ in Fig. 5

The choreograph satisfies Assumption 1, and no synchronous message is used; therefore, the choreography is strongly realizable. Figure 5 shows $\Rightarrow_{s2}$ when Eq. (11) is used. When Eq. (13) is used, $\Rightarrow_{s2}$ becomes as shown in Fig. 6. This means that peer s2 need not be responsible for the dependency among messages m2, m3, m8, and m9.

From Theorem 1, we can choose any acyclic relation between the relations in Figs. 5 and 6 without loosing strong realizability. That makes it possible to design a synthesizing algorithm that considers the intelligibility of models for users. Let us see an example. Figure 7 depicts the state machine synthesized from the acyclic relation in Fig. 5 by using the algorithm in [10]. If one can find the acyclic relation in Fig. 8, the state machine in Fig. 9 is synthesized by the same algorithm. Because receive event of m8 can occur without sending m2 in the relation in Fig. 8, we can delete the effects and guards from the state machine in Fig. 7. As

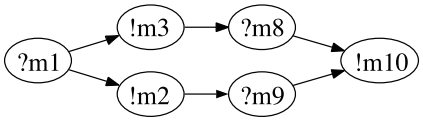

Fig. 8 A $\Rightarrow_{s2}$ between the relations in Figs. 5 and 6

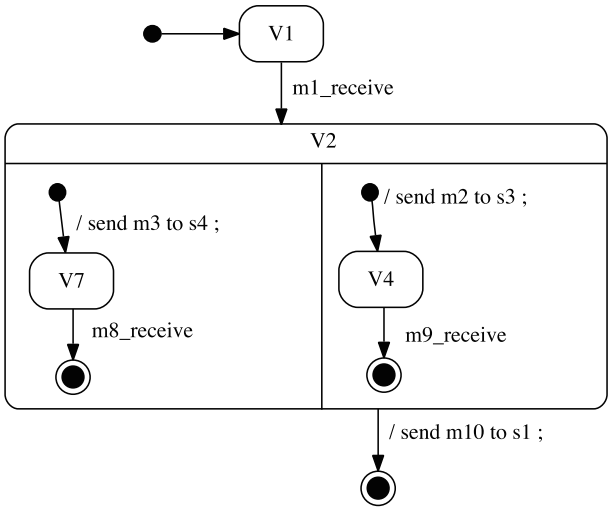

Fig. 9 State machine synthesized from $\Rightarrow_{sm2}$ in Fig. 8

Table 1 Number of simple states (NSS), transitions (NT), guards (NG), and do-activities (NA) in state machines in Figs. 7 and 9

state machine	NSS	NT	NG	NA
Fig. 7	3	7	2	0
Fig. 9	3	7	0	0

Table 1 shows, the state machine in Fig. 9 is more intelligibility than one in Fig. 7.

Moreover, we assumed that choreography is given by a communication diagram in this paper. In practice, however, a set of communication diagrams will be given for specifying various scenarios, and it is required that those synthesized state machines cope with all scenarios. Suppose that we have two communication diagram and the acyclic relations for a peer are different, it is not easy synthesizing a state machine that satisfies both acyclic relations. The condition proposed in this paper provides lower and upper bounds of the acyclic relation for each behavioral model. Thus, there are possibilities of finding the acyclic relation that satisfies both the specifications. That is useful for synthesizing state machines from a set of communication diagrams.

4.4 Assumption 1 in the CRP

The following lemma holds for Assumption 1 in the CRP.

Lemma 5: If $\Rightarrow$ does not satisfy Assumption 1, the choreography is not strongly realizable.

Proof: If Assumption 1 does not hold, $\mathfrak{L}(\Rightarrow) \subset \mathfrak{L}(\Rightarrow^{\min})$ from Lemma 4. Because $\mathfrak{L}(\Rightarrow^{\min}) = \mathfrak{L}(\Rightarrow^{\max})$, $\mathfrak{L}(\Rightarrow) \subset \mathfrak{L}(\Rightarrow^{\max})$. That implies $\Rightarrow_{\mathrm{com}} \subset \Rightarrow \setminus (\bigcup_c \Rightarrow_c^{\max})$. Let $(e_1, e_2) \notin \Rightarrow_{\mathrm{com}}$ and $(e_1, e_2) \in \Rightarrow \setminus (\bigcup_c \Rightarrow_c^{\max})$, then events e_1 and e_2 occur in different peers and the peer in which event e_2 occurs cannot sense the occurrence of event e_1. □

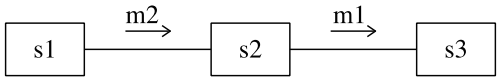

Fig. 10 Un-realizable choreography

Fig. 11 $\Rightarrow$ of choreography in Fig. 10

Fig. 12 $\Rightarrow^{\min}$ of choreography in Fig. 10

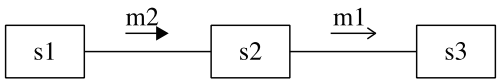

Fig. 13 Realizable choreography

Fig. 14 $\Rightarrow$ and $\Rightarrow^{\min}$ of choreography in Fig. 13

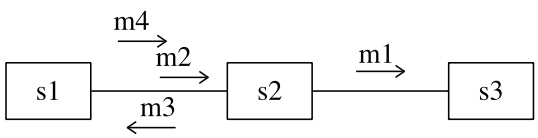

Fig. 15 Weakly realizable choreography

Figure 10 shows a typical un-realizable choreography, where messages m1 and m2 are asynchronous, and D = {(!m1, !m2)}. That means that sending message m2 must follow m1. However, because peer s1 cannot know when peer s2 sends m1, this choreography is not realizable. Figures 11 and 12 show $\Rightarrow$ and $\Rightarrow^{\min}$, respectively. Transitive closures of these are not the same; this choreography does not satisfy Assumption 1. In fact, for a word γ = !m2!m1?m1?m2 and $\gamma \in \mathfrak{L}(\Rightarrow^{\min})$ but $\gamma \notin \mathfrak{L}(\Rightarrow)$.

In the communication diagram in Fig. 13, the message m2 is changed to synchronous from Fig. 10, where the reply message is omitted to simplify. The synchronous message m2 cannot be sent until peer s2 sends m1; thus, this choreography is realizable. Figure 14 shows $\Rightarrow$ of this choreography. This is also $\Rightarrow^{\min}$. Thus, this choreography satisfies the assumption.

Figure 15 shows another choreography, where D = {(!m1, !m2), (!m1, !m3), (!m3, !m4)}. Similar to the case in Fig. 10, this choreography does not satisfy Assumption 1. However, if peer s1 sends message m2 after receiving mes-

sage m3, then the state machine is a non-trivial weak realization. Let us look at this in more detail. We can find three conversations from this choreography: m1m2m3m4, m1m3m2m4, and m1m3m4m2. Putting m2 after m3 means that we forget the conversation m1m2m3m4. The last example shows that dissatisfaction with Assumption 1 does not equal un-realizability of the given choreography. We may need another criterion to check un-realizability, but that is left for future research.

5. Conclusion

This paper approached the CRP from the viewpoint of re-constructible decomposition of acyclic relations and derived lower and upper bounds of the acyclic relation for each peer. The bounds are useful to develop the synthesizing algorithm of state machines that are intelligible to users or when choreography is given by a set of communication diagrams. We intend to further investigate these subjects in the future.

References

[1] T. Erl, Service-Oriented Architecture: Concepts, Technology, and Design, Prentice Hall Professional Technical Reference, 2005.
[2] D. Harel, H. Kugler, and A. Pnueli, "Synthesis revisited: generating statechart models from scenario-based requirements," in Formal Methods in Software and Systems Modeling, ed. H.J. Kreowski, U. Montanari, F. Orejas, G. Rozenberg, and G. Taentzer, pp.309–324, Springer, 2005.
[3] J. Whittle and P.K. Jayaraman, "Synthesizing hierarchical state machines from expressive scenario descriptions," ACM Trans. on Software Eng. and Methodology, vol.19, no.3, pp.1–45, Jan. 2010.
[4] H. Liang, J. Dingel, and Z. Diskin, "A comparative survey of scenario-based to state-based model synthesis approaches," Proc. Intl. Workshop on Scenarios and State Machines: Models, Algorithms, and Tools, pp.5–12, May 2006.
[5] 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.
[6] 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.
[7] Object Management Group, "OMG Unified Modeling Language (OMG UML), superstructure," Aug. 2011. (accessed Oct. 31, 2013).
[8] B. Caillaud, P. Caspi, A. Girault, and C. Jard, "Distributing automata for asynchronous networks of processors," European Journal of Automation, vol.31, no.3, pp.503–524, 1997.
[9] 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.
[10] 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.
[11] T. Murata, "Petri nets: Properties, analysis and applications," Proc. IEEE, vol.77, no.4, pp.541–580, April 1989.
[12] D. Kratsch and J.-X. Rampon, "Towards the reconstruction of posets," Order, vol.11, no.4, pp.317–341, 1994.
[13] M. Rademaker, B. De Baets, and H. De Meyer, "Informative combination of multiple partial order relations," in Multicriteria Ordering and Ranking: Partial Orders, Ambiguities and Applied Issues, Systems Research Institute, Polish Academy of Sciences, 2008.
[14] M. Rademaker and B. De Baets, "Consistent union and prioritized consistent union: new operations for preference aggregation," Annals of Operations Research, vol.195, no.1, pp.237–259, Feb. 2011.
[15] C. Pang, J. Wang, Y. Cheng, H. Zhang, and T. Li, "Topological sorts on DAGs," Information Processing Letters, vol.115, no.2, pp.298–301, Feb. 2015.
[16] G. Brightwell and P. Winkler, "Counting linear extensions," Order, vol.8, no.3, pp.225–242, 1991.
[17] D. Jungnickel, Graphs, Networks and Algorithms, 3rd ed., Springer, 2007.
[18] Y. Hasegawa, H. Niimura, and T. Miyamoto, "A UML subset for design and verification of systems based on SOA," IEICE Technical Report, MSS2011-48, Nov. 2011 (in Japanese).
[19] W. Damm, B. Josko, A. Pnueli, and A. Votintseva, "A discrete-time UML semantics for concurrency and communication in safety-critical applications," Science of Computer Programming, vol.55, no.1–3, pp.81–115, March 2005.
[20] J. Dubrovin and T. Junttila, "Symbolic model checking of hierarchical UML state machines," 2008 8th International Conference on Application of Concurrency to System Design, pp.108–117, 2008.

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.