IEICE TRANSACTIONS

on Fundamentals of Electronics, Communications and Computer Sciences

DOI:10.1587/transfun.2024MAP0004

Publicized:2024/08/20

This advance publication article will be replaced by the finalized version after proofreading.

A PUBLICATION OF THE ENGINEERING SCIENCES SOCIETY

The Institute of Electronics, Information and Communication Engineers

Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3 chome, Minato-ku, TOKYO, 105-0011 JAPAN

PAPER

Generating Event Structure from Set of Acyclic Relations in Choreography Realization

Toshiyuki MIYAMOTO[†], ***Senior Member and*** **Hiroki AKAMATSU**[††], ***Nonmember***

SUMMARY Service-Oriented Architecture (SOA) is an information system architecture. In SOA, the problem of synthesizing the concrete model from an abstract specification is known as the Choreography Realization Problem (CRP). So far, we proposed to use event structures as a modeling formalism; we studied conflict reduction of event structures to check the realizability of a choreography. In this paper, we study the method to generate event structure from choreography.

key words: *SOA, choreography realization, event structure*

1. Introduction

Service-Oriented Architecture (SOA) is an information system architecture [1]. In SOA, an information system is constructed by combining independent software units called services. Orchestration and choreography are known as a method of implementing SOA-based systems, and Stutz et al. [2] compare them in the automation context.

In SOA, the problem of synthesizing the concrete model from an abstract specification is known as the Choreography Realization Problem (CRP) [3]. We have studied the CRP when choreography is given by an acyclic relation of events in [4] and when choreography is given by two acyclic relations in [5]. However, we found that acyclic relations were not an adequate formalism to model choreography; we proposed to use event structures [6, 7] as a modeling formalism in [8]. In [9], we studied conflict reduction of event structures to check the realizability of a choreography. In this paper, we study the method to generate event structure from choreography.

2. Preliminaries

2.1 Choreography Realization Problem

Figure 1 shows the choreography realization procedure in our research. We defines an abstract specification using communication diagrams in unified modeling language (UML) [10]. The communication diagram is one of behavior diagrams of UML. "Communication Diagrams focus on the interaction between Lifelines where the architecture of the internal structure and how this corresponds with the message passing is central. The sequencing of Messages is given through a sequence numbering scheme [10]." In this paper, the Lifelines

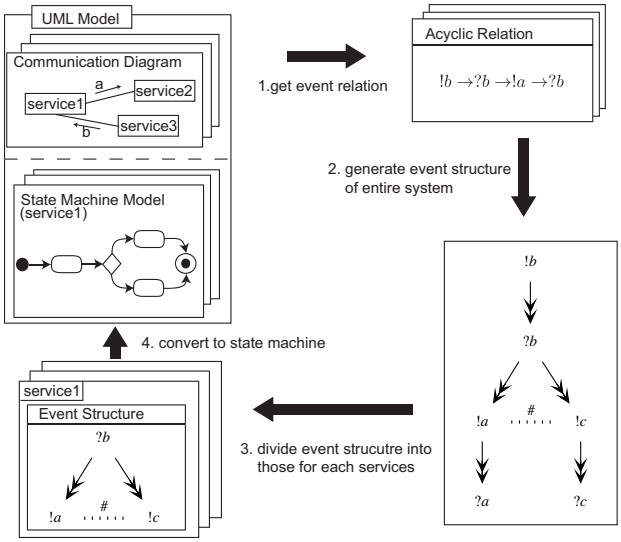


Fig. 1: choreography realization procedure

are called services; the sequencing of Messages is defined as an acyclic relation in some way. Because one communication diagram is used to define one scenario, a set of communication diagrams is used to define a choreography. A concrete model is expressed using the state-machine diagram of UML. Because one state-machine diagram is used to express the behavior of one service, a set of state-machine diagrams is used to define the behavior of entire system.

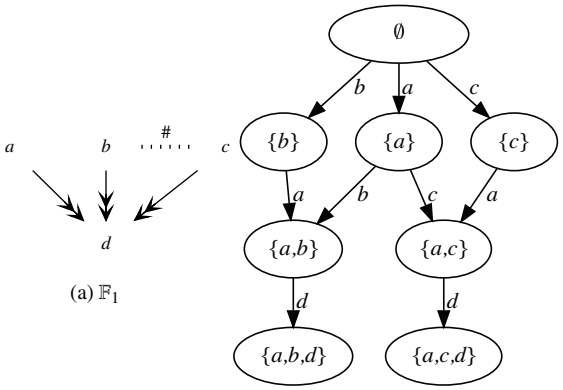
At step 1 of our procedure, ordering relation of events that occur in services is got for each communication diagrams using the method proposed in [11]. At step2, one event structure of entire systems is generated by combining event relations. The procedure to generate an event structure is shown in Sect. 3. An event structure for each service is retrieved by dividing the event structure at step 3; a state-machine diagram is generated by converting an event structure to a state-machine diagram at step 4. This paper focuses the generating method of the event structure of entire system at step 2.

2.2 Event Structure

Event structures are mathematical modeling notation that is able to express behavior of concurrent systems. The fundamental class of event structures is the prime event structure [6]; this paper uses the flow event structure [7].

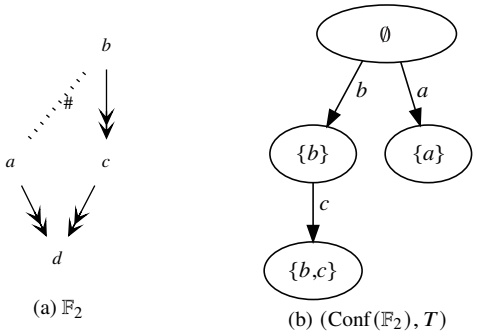
[†]The author is with the Faculty of Information Science and Technology, Osaka Institute of Technology, irakata, JAPAN.

[††]The author is with the School of Engineering, Osaka University, Suita, JAPAN.



(a) $\mathbb{F}_1$ (b) $(\mathrm{Conf}(\mathbb{F}_1), T)$

Fig. 2: FES $\mathbb{F}_1$ (a) and its configuration space (b)



(a) $\mathbb{F}_2$ (b) $(\mathrm{Conf}(\mathbb{F}_2), T)$

Fig. 3: An example of semantic conflict. Events a and c are in semantic conflict because no configuration that contains both a and c exists in $\mathrm{Conf}(\mathbb{F}_2)$.

Definition 1 (flow event structure) A *flow event structure* (FES) is a tuple $\mathbb{F} = (E, \prec, \#, \lambda)$, where E is a set of events, $\prec$ is a *flow* relation, $\#$ is a *conflict* relation such that

1. the flow relation $\prec \subseteq E \times E$ is irreflexive;
2. the conflict relation $\# \subseteq E \times E$ is symmetric,

and $\lambda : E \to \Sigma$ is a labeling function from events to the alphabet Σ.

The labeling function is used for two or more events having the same label. Initially, the label is the event itself. In Sect. 3.2, we will introduce the combination procedure of event structures. Then, two or more events have the same meaning; in this case, the events have the same label.

The $\prec$-predecessors of an event $e \in E$ are defined as $\bullet e = \{e' \mid e' \prec e\}$; the definition of $\prec$-predecessors is extended to the set X of events as $\bullet X = \bigcup_{x \in X} \bullet x$.

Fig. 2 (a) depicts an FES. A flow relation is represented with a double-headed arrow, and a conflict relation is represented with a dotted line labeled by a sharp ($\#$). When is is complicated, the $\#$ label will be omitted.

The flow relation is not required to be transitive. The $\prec$-predecessors $\bullet e$ of an event e can be seen as a set of possible immediate causes for e in an FES. In an FES, conflicts can exist in $\bullet e$; the event e needs to be preceded by a maximal and conflict-free subset of $\bullet e$. The set of maximal and conflict-free $\prec$-predecessors of an event e is denoted by $\Phi(e)$. In the FES in Fig. 2 (a), events b and c are in conflict; $\prec$-predecessors of event d are $\bullet d = \{a, b, c\}$ and the set of maximal and conflict-free $\prec$-predecessors of d is $\Phi(d) = \{\{a, b\}, \{a, c\}\}$. Thus, the occurrence of d must be preceded by either $\{a, b\}$ or $\{a, c\}$.

In an FES, the flow relation is required to be irreflexive only; thus, it could be cyclic. An FES is called *acyclic* when the flow relation is acyclic. In this paper, we assume any FES to be acyclic.

The notion of configuration in an FES is defined in [7, 12, 13] as follows:

Definition 2 (configuration) The configuration of an FES $\mathbb{F}$ is a finite set of events $C \subseteq E$ such that

1. (conflict-freeness) $\neg(e \# e')$ for all $e, e' \in C$;
2. ($\prec$-closedness) $\prec^*_{|C}$ is a partial order;
3. (maximality) for all $e \in C$ and $e' \notin C$ s.t. $e' \prec e$, there exists an $e'' \in C$ such that $e' \# e'' \prec e$,

where $\prec^*$ is the reflexive and transitive closure of $\prec$; $\prec_{|C}$ is the restriction of $\prec$ to the relation on C.

The set of all "labeled"-configurations of an FES $\mathbb{F}$ is denoted by $\mathrm{Conf}(\mathbb{F})$. A configuration space is the graph whose vertex set is $\mathrm{Conf}(\mathbb{F})$ and an edge represents a transition from one configuration to another configuration.

A configuration is a conflict-free and $\prec$-closed subset of events. The third condition means that given an event $e \in C$, for any $\prec$-predecessor $e' \prec e$, either $e' \in C$ or it is excluded by the existence of $e'' \in C$ such that $e' \# e'' \prec e$. Thus, for any $e \in C$, the configuration C must include a maximal and conflict-free subset of the $\prec$-predecessors of e. The configuration space of the FES $\mathbb{F}_1$ is depicted in Fig. 2 (b).

In the FES in Fig. 3 (a), events a and c are not syntactically in conflict; however, there is no configuration that contains both a and c. Thus, a and c are semantically in conflict.

Definition 3 (semantic conflict) The events e and e' in an FES $\mathbb{F}$ are in *semantic conflict*, which is denoted by $e \#_S e'$, when for all configurations $C \in \mathrm{Conf}(\mathbb{F})$, it does not hold that $\{e, e'\} \subseteq C$.

Clearly, $\# \subseteq \#_S$. Moreover, $e \#_S e$ could be possible; in this case, e never occurs and is called *dead*.

In our procedure shown in Fig. 1, an event structure is divided into event structures for each service. One may raise the idea that non-existence of conflicts among services is a necessary condition for realizability of given choreography. However, this is not true from the existence semantic conflicts. In [9], we introduced the notion of conflict reduction for the checking of realizability.

The following definitions are introduced in [7] to define sets of events that can be safely merged.

Definition 4 For given set X of events, the set of maximal

and conflict-free subsets of X is denoted by $mc(X)$.

Definition 5 (direct conflict). Let $\mathbb{F} = (E, \#, \prec, \lambda)$ be an FES and let $e, e' \in E$. We say that e is a direct conflict for $e\#e'$, denoted as $e\#_\delta e'$, if $\exists Y \in mc(\bullet e)$ such that $Y \cup \{e'\}$ is conflict-free.

3. Generate event structure of entire system

3.1 Overview

As described in 2.1, in the proposed procedure, the event structure of entire system is generated from multiple acyclic relations. The proposed procedure is composed of two steps: 1) combining multiple event structures into an event structure and 2) folding the event structure. Because an acyclic relation is a special case of event structure, the combination procedure is discussed as the problem of combining multiple event structures.

3.2 Combining event structures

Let $F = \{\mathbb{F}_1, \cdots, \mathbb{F}_n\}$, the combined event structure $\mathbb{F}_m = (E_m, \#_m, \prec_m, \lambda_m)$ is defined as follows:

$$
\begin{aligned}
E_m =& E_1 \cup \cdots \cup E_n \\
\#_m =& \#_1 \cup \cdots \cup \#_n \\
& \cup \{(e_a, e_b) \mid e_a \in E_a, e_b \in E_b, \mathbb{F}, \mathbb{F}_b \in F, \mathbb{F}_a \neq \mathbb{F}_b\} \\
\prec_m =& \prec_1 \cup \cdots \cup \prec_n \\
\lambda_m =& \lambda_1 \cup \cdots \cup \lambda_n
\end{aligned}
$$

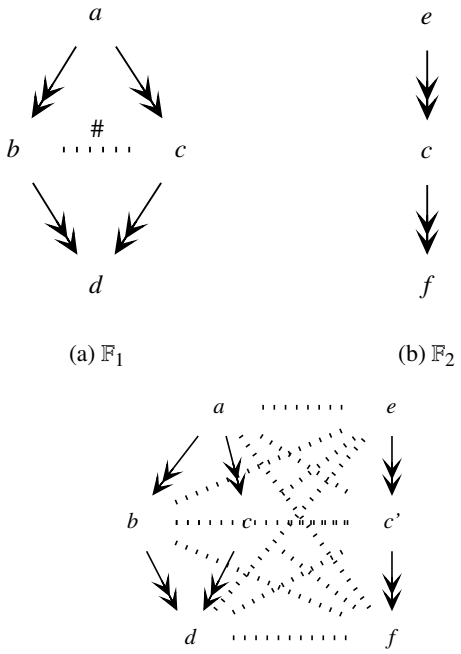
Then, the following lemma holds.

Lemma 1 Let $\mathbb{F}_m$ be the combined event structure from $F = \{\mathbb{F}_1, \cdots, \mathbb{F}_n\}$, then $\mathrm{Conf}(\mathbb{F}_1) \cup \cdots \cup \mathrm{Conf}(\mathbb{F}_n) = \mathrm{Conf}(\mathbb{F}_m)$ holds.

Proof 1 Suppose that there exists $C \in \mathrm{Conf}(\mathbb{F}_1) \cup \cdots \cup \mathrm{Conf}(\mathbb{F}_n)$ such that $C \notin \mathrm{Conf}(\mathbb{F}_m)$. There must exist $e_a, e'_a \in E_a$ for some $\mathbb{F}_a$such that $e_a, e'_a \in C$ and $e_a \#_m e'_a$. Since $C \in \mathrm{Conf}(\mathbb{F}_a)$, $\neg(e_a \#_a e'_a)$. However, the combination procedure does not add any conflict between events in the same event structure. Thus, C must be in $\mathrm{Conf}(\mathbb{F}_m)$, i.e., $\mathrm{Conf}(\mathbb{F}_1) \cup \cdots \cup \mathrm{Conf}(\mathbb{F}_n) \subseteq \mathrm{Conf}(\mathbb{F}_m)$.

Since the combination procedure does not restrict the behavior of each event structure in F. Therefore, $\mathrm{Conf}(\mathbb{F}_m) \subseteq \mathrm{Conf}(\mathbb{F}_1) \cup \cdots \cup \mathrm{Conf}(\mathbb{F}_n)$. □

An example of combination is shown in Fig. 4. The event structures $\mathbb{F}_1$ in Fig. 4(a) and $\mathbb{F}_2$ in Fig. 4(b) are combined; the resulting event structure $\mathbb{F}_m$ is shown in Fig. 4(c). The set of configurations of $\mathbb{F}_1$ is $\mathrm{Conf}(\mathbb{F}_1) = \{\emptyset, \{a\}, \{a, b\}, \{a, c\}, \{a, b, d\}, \{a, c, d\}\}$, that of $\mathbb{F}_2$ is $\mathrm{Conf}(\mathbb{F}_2) = \{\emptyset, \{e\}, \{e, c\}, \{e, c, f\}\}$, and that of $\mathbb{F}_m$ is $\mathrm{Conf}(\mathbb{F}_m) = \{\emptyset, \{a\}, \{a, b\}, \{a, c\}, \{a, b, d\}, \{a, c, d\}, \{e\}, \{e, c\}, \{e, c, f\}\}$; i.e., $\mathrm{Conf}(\mathbb{F}_1) \cup \mathrm{Conf}(\mathbb{F}_2) = \mathrm{Conf}(\mathbb{F}_m)$.



(a) $\mathbb{F}_1$ (b) $\mathbb{F}_2$

(c) $\mathbb{F}_m$

Fig. 4: Combination of event structures

3.3 Folding event structures

When multiple event structures are combined into one by event structure composition, there may be events with the same label in the combined event structure. A process for merging multiple events with the same label while maintaining the behavior of the event structure has been proposed in [7]. This sections introduces the conditions under which event merging is possible and the event merging process, and describes event merging when there are multiple pairs of events that can be merged.

Let $\mathbb{F} = (E, \#, \prec, \lambda)$, a set $X \subseteq E$ of events is called combinable when the following conditions holds [7]:

1. $\forall x, x' \in X : \lambda(x) = \lambda(x')$ and $x \# x'$
2. $\forall x, x' \in X, \forall e \in E : x \#_\delta e \Rightarrow x' \# e$
3. $\forall x, x' \in X, \forall e \in E : x \prec e \Rightarrow x' \prec e \vee x' \# e$
4. $\forall x, x' \in X, \forall e \in E : [e \prec x \Rightarrow \bullet x' \neq \emptyset \wedge \{e \prec x' \vee (\forall e' \prec x' \wedge e' \notin \bullet x : e \# e')\}]$
5. $\forall x \in X, \forall e, e' \in E : [x, e' \in \bullet e \wedge x \# e' \wedge \neg(X \# e') \Rightarrow \forall Y \in mc(\bullet e) : \{(x \in Y \Rightarrow \exists e'' \in Y \backslash \{x\}, e'' \# e') \wedge (X \cap Y = \emptyset \Rightarrow \exists e'' \in Y, X \# e'')\}]$

Let $\mathbb{F}_{/X} = (E_{/X}, \#_{/X}, \prec_{/X}, \lambda_{/X})$ be the event structure after merging all events in a combinable set X, e with $e \#^\forall X$ be an event that is in conflict with every events in X, and e with $e \prec^\exists X$ be an event that precedes at least one event in X; then, each element of $\mathbb{F}_{/X}$ is calculated as follows:

$$E_{/X} = (E \backslash X) \cup \{e_X\}$$

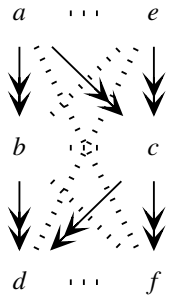
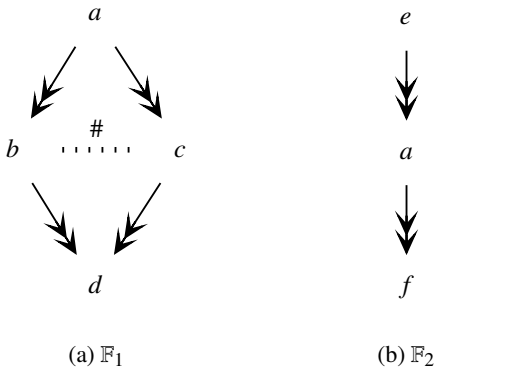
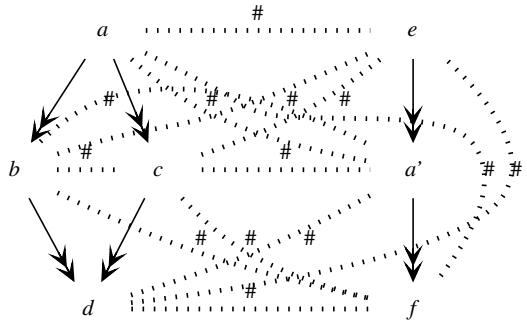


Fig. 5: The event structure $\mathbb{F}_{/X}$ after merging c and c' of $\mathbb{F}_m$ in Fig. 4(c)

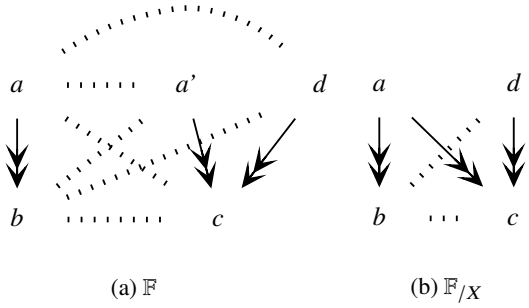


(a) $\mathbb{F}_1$ (b) $\mathbb{F}_2$



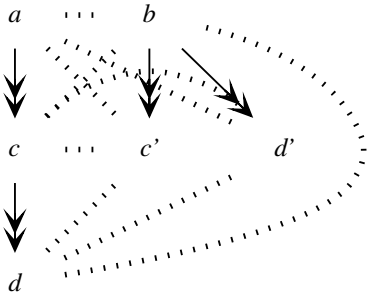
(c) $\mathbb{F}_m$

Fig. 6: $X = \{a, a'\}$ in $\mathbb{F}_m$ is not combinable

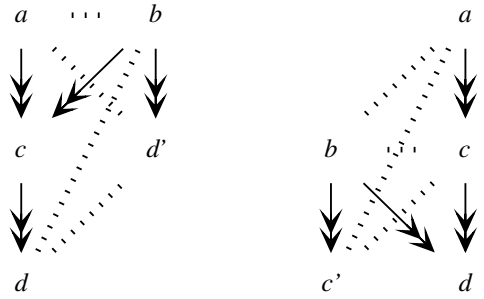


(a) $\mathbb{F}$ (b) $\mathbb{F}_{/X}$

Fig. 7: The set $X = \{a, a'\}$ in $\mathbb{F}$ is not combinable, however, $\mathrm{Conf}(\mathbb{F}) = \mathrm{Conf}(\mathbb{F}_{/X})$



(a) $\mathbb{F}_m$



(b) $\mathbb{F}_{/X_c}$ (c) $\mathbb{F}_{/X_d}$

Fig. 8: The order in which the events are merged produces differ event structures

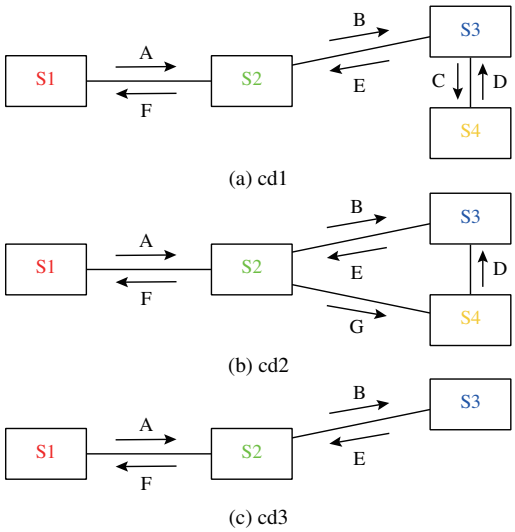
$$\#_{/X} = \#_{|(E\setminus X)} \cup \left\{(e, e_X) \mid e \#^{\forall} X\right\}$$
$$\prec_{/X} = \prec_{|(E\setminus X)} \cup \left\{(e, e_X) \mid e \prec^{\exists} X\right\} \cup \left\{(e_X, e') \mid X \prec^{\exists} e'\right\}$$
$$\lambda_{/X} = \lambda_{/X}\left[e_X \mapsto \lambda(x), x \in X\right].$$

The set $X = \{c, c'\}$ of $\mathbb{F}_m$ in Fig. 4(c) satisfy the conditions of combinable sets. The event structure after merging c and c' is shown in Fig. 5.

Let us see another example. The event structure $\mathbb{F}_1$ in Fig. 6(a) and $\mathbb{F}_2$ in Fig. 6(b) are combined; the resulting event structure $\mathbb{F}_m$ is shown in Fig. 6(c). The set $X = \{a, a'\}$ is not combinable since the condition 4) is not satisfied.

The set $X = \{a, a'\}$ in $\mathbb{F}$ in Fig. 7 is not combinable because the condition 2) is not satisfied, however, $\mathrm{Conf}(\mathbb{F}) = \mathrm{Conf}(\mathbb{F}_{/X})$. The example shows that the condition of combinable set is a sufficient condition. Therefore, there is a room to improve the the condition. This is included in our future research topics.

When multiple combinable sets of events exist in an event structure, the folded event structure may differ depending on the order in which the events are merged. In the event structure $\mathbb{F}_m$ of Fig. 8(a), the sets $X_c = \{c, c'\}$ and $X_d = \{d, d'\}$ are combinable. The event structure merging events in X_c is shown in Fig. 8(b) and the event structure merging events in X_d is shown in Fig. 8(c). In this case, the event set X_d in Fig. 8(b) does not satisfy the condition (2, and the event set X_c does not satisfy the condition (3. There-

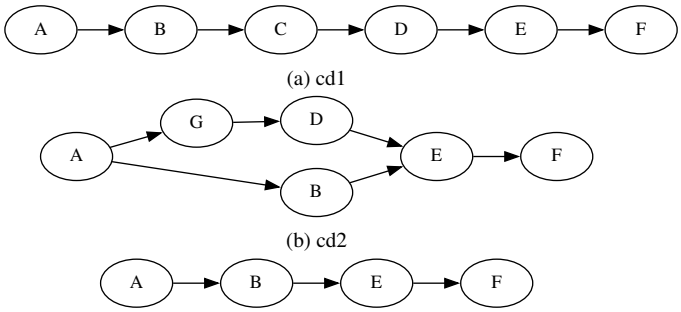


(a) cd1

(b) cd2

(c) cd3

Fig. 9: Choreography of Ex.1



(a) cd1

(b) cd2

(c) cd3

Fig. 10: Message relation of Ex.1

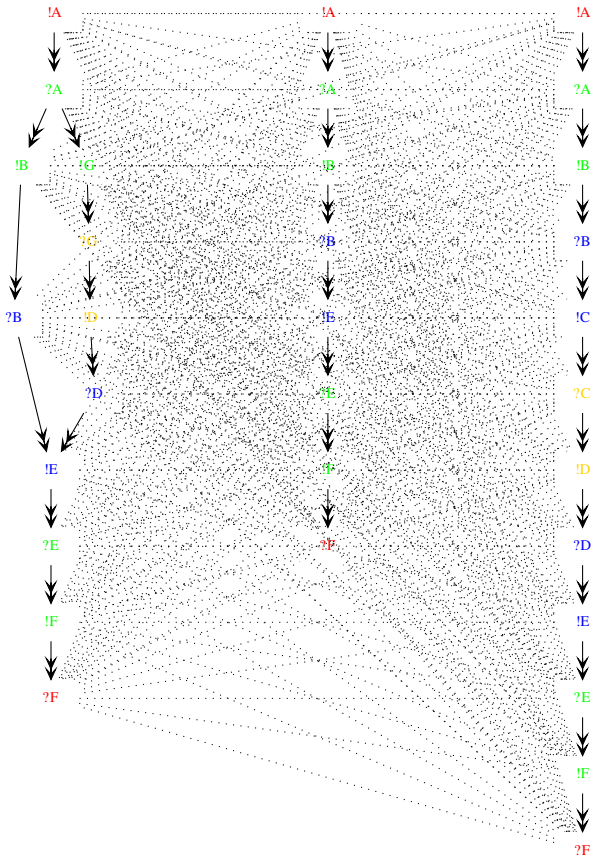


Fig. 11: Combined event structure of Ex.1

fore, in this case, the event structure after merging differs depending on the order of merging events.

4. Case study

Let us consider a system with four services: S1, S2, S3, and S4.

The abstract specification for this system is given by the communication diagram shown in Fig. 9 and the ordering relationship between messages shown in Fig. 10. There are three scenarios for the execution of this system.

In the first scenario, when S2 receives message A sent by S1, S2 sends message B to S3, and S3 sends message C to S4 after receiving B. S4 sends reply message D to S3, S3 sends message E to S2 upon receiving D, and S2 sends F to S1 after receiving E.

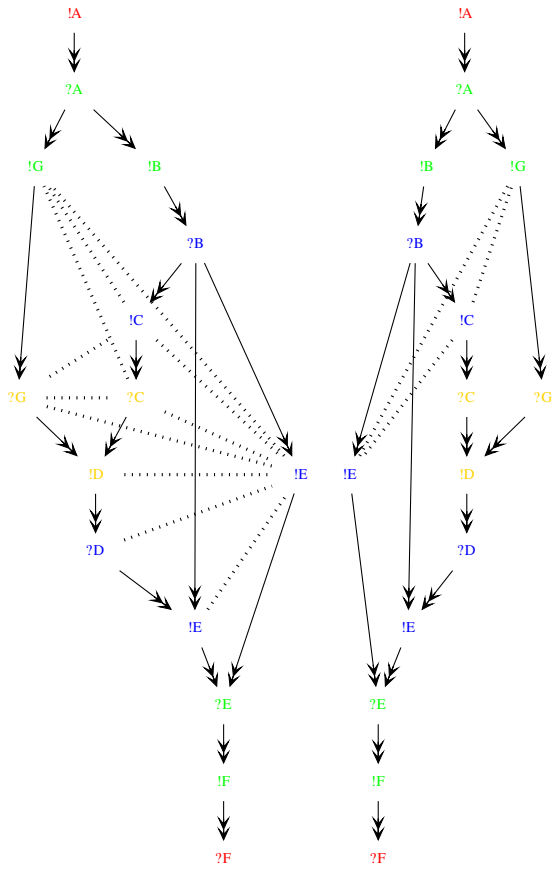
In the second scenario, when S2 receives message A sent by S1, S2 sends message B to S3 and message G to S4; when S4 receives G, it sends message D to S3; when S3 receives B and D, it sends E to S2; after receiving E, S2 sends F to S1; after receiving E, S2 sends F to S1.

In the third scenario, when S2 receives message A sent by S1, S2 sends message B to S3; S3 receives B and sends message E back to S2; S2 sends F to S1 after receiving E.

If we transform the ordering relation between messages shown in Fig. 10 to the ordering relation between sending and receiving events of a message by the method shown in [11] and combine them by the method shown in 3.2, we obtain the event structure in Fig. 11. The sending event of message m is denoted by $!m$ and the receiving event by $?m$. The colors in the figure indicate the service in which the event occurs. For example, red events occur in service S1.

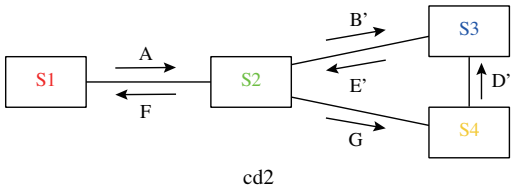
As described in Sect. 3.3, different event structures are generated depending on the order of merging. Therefore, we randomly determined the order of merging in the event structure in Fig. 11 and tried multiple times; then, we obtained two types of event structures. One of them is shown in Fig. 12 (a). In the event structure conflict-reduction by the method shown in [9] is performed, and the reduced event structure is shown in Figure 12 (b). In the event structure, the conflict !G#!E and !G#!C exist across different services S2 and S3. Since such choreography is not realizable, the choreography needs to be modified.

Figure 13 shows the communication diagram cd2 in the modified choreography (Ex.2). cd1 and cd3 are omitted because they are the same as before the modification. In the modified choreography, the message in cd2 is changed from B, D, E to B', D', E'. This allows the receiving service to



(a) (b)

Fig. 12: (a) Folded event structure of Ex.1, and (b) the reduced event structure



cd2

Fig. 13: Choreography of Ex.2

understand the difference in messages.

The event structure that is synthesized and folded in the same way from the modified choreography is shown in Fig. 14 and the reduced event structure is shown in Fig. 15. The event structure in Figure 15 is not unrealizable because the two conflicts exist in the same service. Note that the literature [9] only shows the sufficient condition for unrealizability, and the condition for feasibility has not been derived yet.

5. Conclusions

In this paper, we study the method to generate event structure from choreography; we demonstrated the proposed method via case studies. Our next research topic includes the method to convert the event structure to state machines for each service.

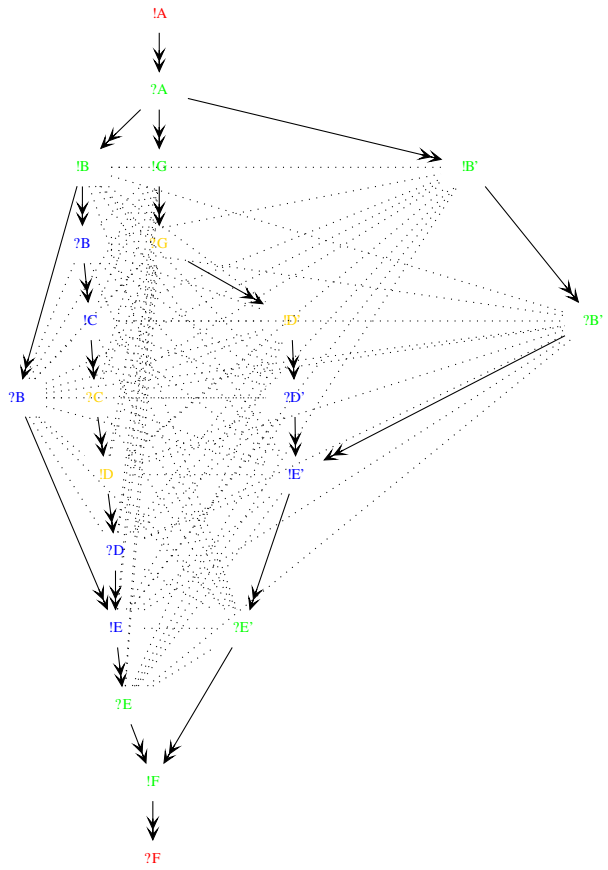


Fig. 14: Folded event structure of Ex.2

Acknowledgment

This work is supported by JSPS KAKENHI Grant Number JP20K11746.

References

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

[2] A. Stutz, A. Fay, M. Barth, and M. Maurmaier, "Orchestration vs. Choreography Functional Association for Future Automation Systems," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 8268–8275, 2020.

[3] 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, Apr. 2008.

[4] T. Miyamoto, "Choreography Realization by Re-Constructible Decomposition of Acyclic Relations," *IEICE Transactions on Information and Systems*, vol. E99.D, no. 6, pp. 1420 – 1427, 06 2016.

[5] T. Kinoshita and T. Miyamoto, "Realizability of Choreography Given by Two Scenarios," *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, vol. E101.A, no. 2, pp. 345 – 356, 00 2018.

[6] M. Nielsen, G. Plotkin, and G. Winskel, "Petri nets, event structures and domains, part I," *Theoretical Computer Science*, vol. 13, no. 1,

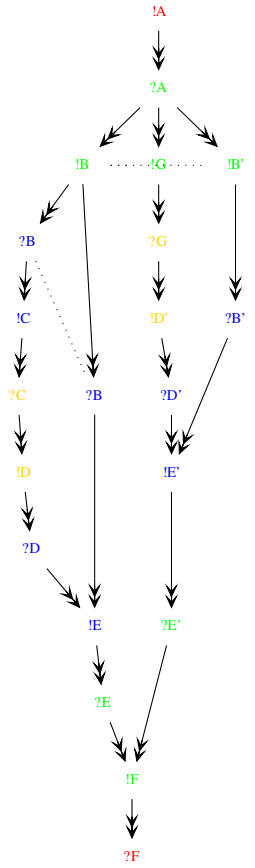


Fig. 15: Reduced event structure of Ex.2

pp. 85–108, Jan. 1981.
[7] A. Armas-Cervantes, P. Baldan, and L. García-Bañuelos, "Reduction of event structures under history preserving bisimulation," *Journal of Logical and Algebraic Methods in Programming*, vol. 85, no. 6, pp. 1110–1130, Oct. 2016.
[8] M. Izawa and T. Miyamoto, "A study on re-constructibility of event structures," *IEICE Transactions on Information and Systems*, vol. E103.D, no. 8, pp. 1810–1813, 2020.
[9] T. Miyamoto and M. Izawa, "Conflict reduction of acyclic flow event structures," *IEICE Trans. on Fund. of Elec., Comm, and Comput. Sci.*, vol. E106-A, no. 5, 2023.
[10] "OMG Unified Modeling Language," Tech. Rep., 2017.
[11] T. Miyamoto, Y. Hasegawa, and H. Oimura, "An approach for synthesizing intelligible state machine models from choreography using Petri nets," *IEICE Transactions on Information and Systems*, vol. E97.D, no. 5, pp. 1171–1180, May 2014.
[12] G. Boudol and I. Castellani, "Permutation of transitions - An event structure semantics for CCS and SCCS," *LNCS*, vol. 354, pp. 411–427, 1989.
[13] G. Boudol, "Flow event structures and flow nets," *LNCS*, vol. 469, pp. 62–95, 1990.

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 a Professor with the Faculty of Information Science and Technology, Osaka Institute of Technology. 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.

Hiroki Akamatsu received the B.E. and M.I.S.T. degrees from Osaka University, Osaka, Japan in 2022 and 2024, respectively.