CSCB Tools: A Tool to Synthesize Pareto Optimal State Machine Models from Choreography Using Petri Nets*

Toshiyuki Miyamoto[1]

Graduate School of Engineering, Osaka University,
Suita, Osaka 565-0871, Japan
miyamoto@eei.eng.osaka-u.ac.jp

Abstract. Application of service-oriented architecture, in which the entire system is built by a combination of independent software components, to a wide variety of computer systems is expected. The problem to synthesize state machine models of the services from a communication diagram representing the overall specifications of service interaction is known as the choreography realization problem. It should be minded on automatic synthesis that software models should be understandable to software engineers. We have proposed a method to synthesize hierarchical state machine models for the choreography realization problem in former PNSEs. We have implemented the method as a plug-in of Rational Software Architect. In this paper, we present the prototypical tool.

1 Introduction

In recent years, the internationalization of activities and information technology in enterprises has intensified competition between companies. Under such circumstances, service-oriented architecture (SOA)[13] has been attracting attention as the architecture of information systems in enterprises. In SOA, an information system is built by composing independent software units called services.

In SOA, the problem to synthesize the concrete model from an abstract specification is known as the choreography realization problem[12], in which the abstract specification, called *choreography*, is defined as a set of interactions among services, which are given in a dependency relation of messages sent and received; the concrete model is called the *service implementation* and defines the behavior of the service. This paper utilizes the communication diagram and the state machine of UML 2.x[11] to describe the choreography and the service implementation, respectively.

Bultan and Fu formally introduced the choreography realization problem in [2]. They used collaboration diagrams of UML1.x and showed some conditions for a given choreography to be realizable. In addition, they showed a method

* This work was supported by KAKENHI (26330083).

to represent the service implementation as the state space in which a state was defined as a set of unsent messages, and they also showed a method to map to a set of finite state machines. However, it is not intelligible because the number of states increases exponentially as the number of messages increases.

Cruz-Lemus et al. experimentally evaluated the relationship between some metrics of state machines and the time needed to understand them[1]. According to the result, state machines are intelligible the smaller the following metrics: the number of simple states (NSS), the number of transitions (NT), and the number of guards (NG).

Miyamoto et al. proposed the Construct State-machine Cutting Bridges (CSCB) method, a method for synthesizing hierarchical state machines from a communication diagram[6, 8, 7]. In this method, dependency relations among sent and received message events are represented by Petri nets[9]; state machines are then synthesized. We have implemented the CSCB method as a plug-in of Rational Software Architect (RSA) [3]

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[4]. Using the re-constructibility, we have proposed an extended version of the CSCB method in [5]. In this paper, we report the extension of the plug-in of RSA so as to support the extended CSCB method.

2 CSCB Tools

2.1 CSCB method

The CSCB method synthesizes state machines for each service from choreography defined by a communication diagram. At first, the method derives an acyclic relation for each service. Second, the relations are transformed to Petri nets. Third, when a Petri net has T-T bridges, it is modified by cutting bridges while keeping the behavior. Finally, each Petri net is converted into a hierarchical state machine. We use RSA as the editor for communication diagrams and the repository to store synthesized state machines.

2.2 Extended CSCB method

The result in [4] allows us to use any acyclic relation in some range. In the extended CSCB method, a set of acyclic relations for each service is derived at the first step. A set of state machines are synthesized and our tool selects more intelligible state machines using the following metrics for intelligibility: NSS, NT, NG, number of depths (ND), and sum of number of partners for all regions (NP). Because we use several metrics, a set of Pareto optimal state machines is selected.

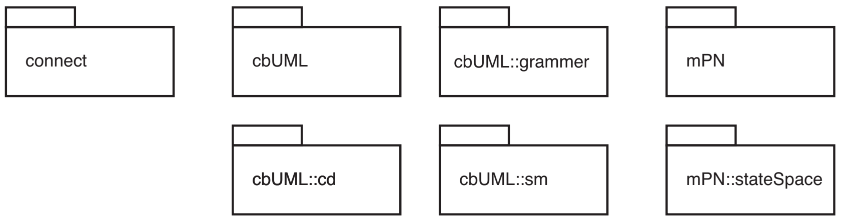

Fig. 1. Packages of CSCB Tools

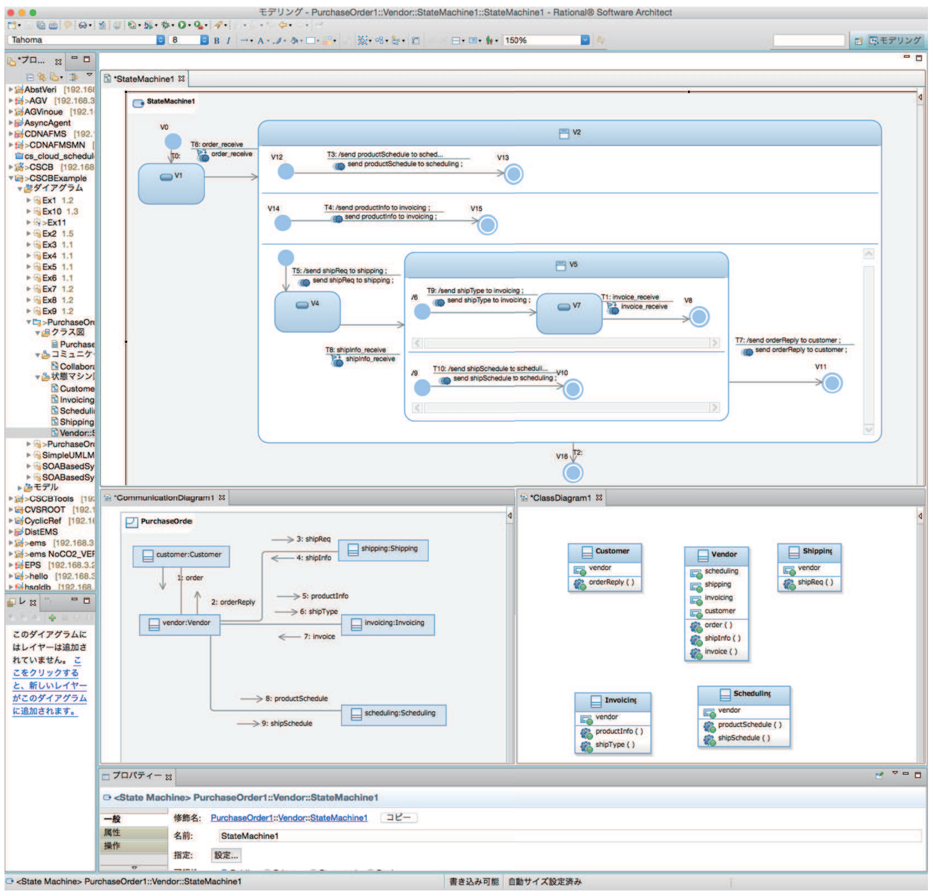
モデリング - PurchaseOrder1::Vendor::StateMachine1::StateMachine1 - Rational® Software Architect
*StateMachine1
StateMachine1
*CommunicationDiagram1
*ClassDiagram1
Customer
Vendor
Shipping
Invoicing
Scheduling
プロパティー
<State Machine> PurchaseOrder1::Vendor::StateMachine1

Fig. 2. Screen shot

2.3 Architecture of CSCB Tools

We have implemented the synthesis function as a plug-in of RSA; one can download the plug-in from the following URL:

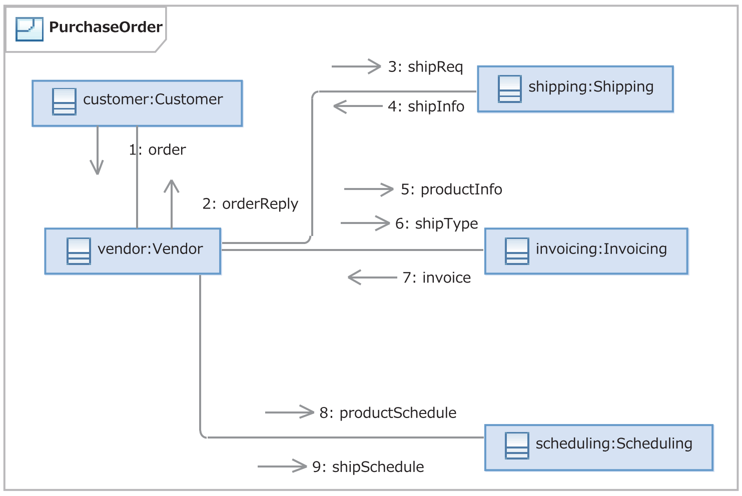

Fig. 3. Purchase Order System

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

Fig. 1 shows packages of the CSCB Tools. We have proposed an UML subset, called the *subset of UML for formally describing choreography and behavioral feature* (cbUML), to discuss the choreography realization problem, and an extended Petri net, called the message mark graph (MMG) [7]. `cbUML` is the main package for cbUML models, `cbUML::cd` is the package for communication diagrams, `cbUML::sm` is the package for state machines, and `cbUML::grammer` defines the grammer of statements in cbUML models. `mPN` is the package for MMGs and `mPN::stateSpace` is the package to generate reachability spaces of MMGs, which is used in the projection method[2]. The package `connect` connects cbUML model with RSA. Retrieving and storing UML models from and to RSA are done through `connect`. Fig. 2 shows a screen-shot of the tool. We use class and communication diagrams to define choreography.

2.4 Example

Fig. 3 shows an example choreography, which is taken 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

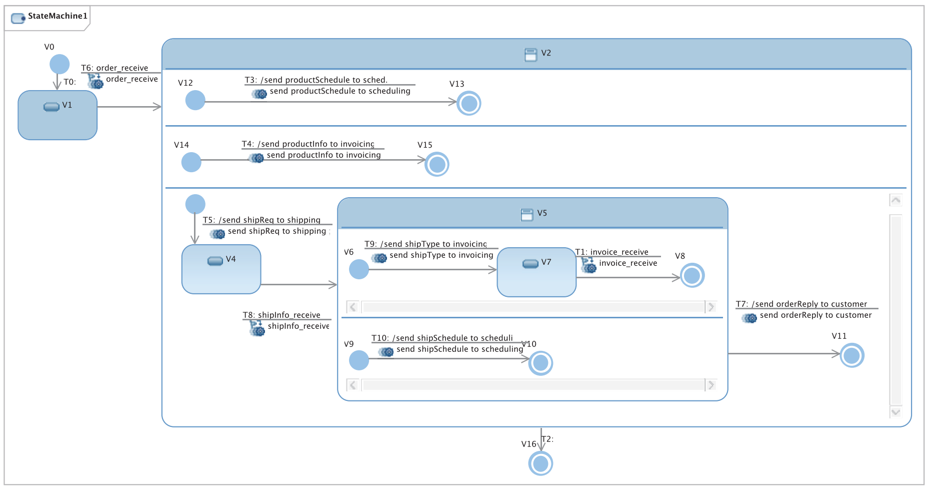
StateMachine1
V0
T6: order_receive
order_receive
T0:
V1
V2
V12
T3: /send productSchedule to sched.
send productSchedule to scheduling
V13
V14
T4: /send productInfo to invoicing
send productInfo to invoicing
V15
T5: /send shipReq to shipping
send shipReq to shipping
V4
V5
V6
T9: /send shipType to invoicing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

Fig. 4. State machine of `Vendor` service

reply. The request to `Scheduling` are `productSchedule` and `shipSchedule`. `Vendor` sends `productSchedule` after receiving `order`; it sends `shipSchedule` after receiving `shipInfo`. `Vendor` sends `orderReply` to `Customer` after receiving `shipInfo` and `invoice` and sending `shipSchedule`.

From this communication diagram, twelve state machines were constructed. Among the state machines, the tool selected the state machine shown in Fig. 4 as the best. 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`.

Table 1. 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

Table 1 shows the comparison result. Compared to the projection method[2], CSCB and extended CSCB methods succeeded in reducing NSS and NT. Compared to the CSCB method[7], the extended CSCB methods succeeded in synthesizing more intelligible state machine. Specially, the state machine does not have any guards and, thus, it has no implicit control flow.

3 Conclusions

In this paper, we extended the function for synthesizing state machines of our tool. The function has been implemented as a plug-in of Rational Software Architect. Currently, our tool can synthesize hierarchical state machines of services from one communication diagram. Synthesizing state machines from multiple communication diagrams is one of our interests. State machines may be modified by designers; conformance checking of state machines should also be developed. Our further interest is developing model checking and simulation functionalists of communication diagrams or state machines for checking some properties of the system.

References

1. Antonio Cruz-Lemus, J., Genero, M., Piattini, M.: Metrics for UML Statechart Diagrams. In: Genero, M., Piattini, M., Calero, C. (eds.) Metrics for Software Conceptual Models, pp. 237–272. Imperial College Press, London (2005)
2. Bultan, T., Fu, X.: Specification of realizable service conversations using collaboration diagrams. Service Oriented Computing and Applications 2(1), 27–39 (2008)
3. Miyamoto, T.: CSCB Tools: Tool for supporting the design of systems based on soa. In: IEEE GCCE. pp. 42–43 (2015)
4. Miyamoto, T.: Choreography realization by re-constructible decomposition of acyclic relations. IEICE Transactions on Information and Systems E99.D(6), 1420–1427 (2016)
5. Miyamoto, T.: Choreography realization by re-constructible decomposition of acyclic relations. Tech. Rep. MSS2015-37, IEICE (2016)
6. Miyamoto, T., Hasegawa, Y.: A Petri net approach to synthesize intelligible state machine models from choreography. In: PNSE. pp. 222–236 (2012)
7. Miyamoto, T., Hasegawa, Y., Oimura, H.: An approach for synthesizing intelligible state machine models from choreography using petri nets. IEICE Transactions on Information and Systems E97.D(5), 1171–1180 (2014)
8. Miyamoto, T., Oimura, H.: A tool to synthesize intelligible state machine models from choreography using petri nets. In: Joint Proc. of PNSE and ModBE. pp. 257–258 (2013)
9. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE 77(4), 541–580 (Apr 1989)
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. OMG: Unified modeling language, http://www.uml.org/
12. Su, J., Bultan, T., Fu, X., Zhao, X.: Towards a theory of web service choreographies. In: Proceedings of the 4th international conference on Web services and formal methods. pp. 1–16 (2008)
13. Thomas, E.: Service-Oriented Architecture. Prentice Hall (2004)