Journal of Computing and Security

July 2015, Volume 2, Number 3 (pp. 209–222)

http://www.jcomsec.org

Toward Automatic Transformation of Service Choreography Into UML State Machine

Yousef Rastegari [a,*], Fereidoon Shams [a]

[a] *Department of Computer Engineering and Science, Shahid Beheshti University, Tehran, Iran.*

ARTICLE INFO.

Article history:
Received: 24 April 2015
Revised: 17 January 2016
Accepted: 04 July 2016
Published Online: 24 August 2016

Keywords:
Service Choreography, Model-Driven Transformation, Adaptation, UML State Machine, Collaborative Business Process

ABSTRACT

An adaptive process consists of dynamic elements, and management rules which govern their run-time behaviors. The WS-CDL describes collaborative business processes between service consumers and providers. Adapting the processes to runtime changes becomes a demanding challenge, because the WS-CDL has static technology-dependent structure, and does not support the separation of concerns. Here, we propose a model-driven approach to transform WS-CDL into UML state machine (behavioral and protocol models), and subsequently into implementation code. Besides separating the business logic from the implementation, the state machine has a dynamic structure which is verifiable and adaptable. As a result, we can easily modify the process flow or change the management rules at run-time, and reflect their effects on the running process instances. We present an 'itinerary purchase' case study for prototyping the transformation rules.

1 Introduction

Choreography addresses the interaction that implements the collaboration between services. Choreography describes collaborative business processes (CBP) to achieve common goals among multiple distributed partners. It shows a global view of all interactions, and specifies the potential (observable) behaviors a partner can exhibit in order to interact. Orchestration refers to a composed business process which use both internal and external web services to fulfill its task [1]. Moreover, using service orchestration, each partner can provide its own internal realization of observable behaviors [2]. Figure 1 shows a conceptual model of the CBP complementary concepts including service choreography, observable behavior, and service orchestration.

The Web Service Choreography Description Language (WS-CDL) [3] is the W3C recommended language for specifying service-based CBPs. An adaptive CBP consists of dynamic elements and management rules which govern their run-time behaviors [4]. Since WS-CDL has a static structure, it is necessary to transform the service-based CBPs into adaptive models. Meanwhile, when a new requirement arises at choreography-level, it must be realized at orchestration-level. Therefore, the adaptive model must cover all choreography, and orchestration entities in different abstraction levels, and also consider the interoperability between them.

WS-CDL transformation is addressed in the literature, but mostly with the goal of choreography verification. We selected UML state machine (UML-SM) as an adaptive model to achieve the adaptation requirements of CBPs. The adaptation might be required in case of context (*e.g.*, computational or environmental)

* Corresponding author.
Email addresses: y_rastegari@sbu.ac.ir (Y. Rastegari), f_shams@sbu.ac.ir (F. Shams)
ISSN: 2322-4460

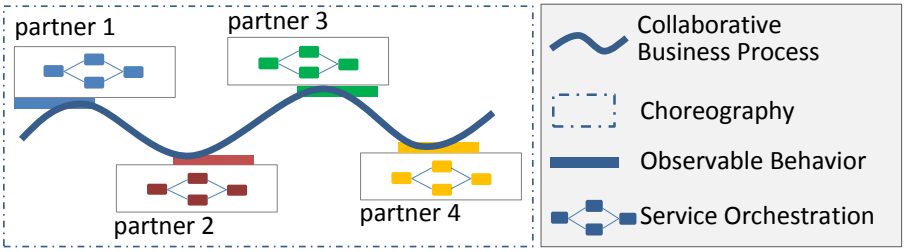

Figure 1. A Conceptual Model of a Service-based CBP

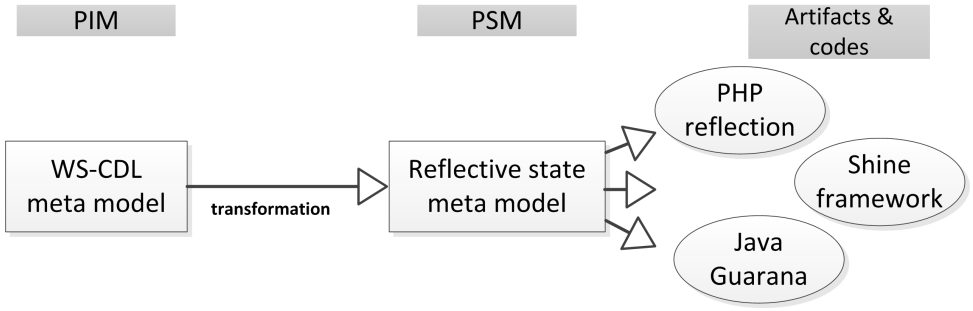

Figure 2. Model-driven Transformation Overview

changes, user preference changes, or business rules changes. In this regard, it is possible to transform both WS-CDL and WSBPEL documents into state-based models, and then integrate them by using nested property of UML-SM. A preliminary version of WS-CDL to UML-SM transformation was presented in [5], [6]. In this paper, we propose the transformation rules for converting WS-CDL into UML-SM regarding Reflective state meta model.

The rest of the paper is organized as follows. Section 2 explains the overview of proposed model-driven transformation. In Section 3 we describe an overview of WS-CDL specification. WS-CDL is corresponded with UML state machine and the rationale behind the transformation is discussed in Section 4. We prototype the trans-formation rules based on an itinerary purchase case study in Section 5. Section 6 provides the semantic preservation of transformation. Section 7 presents related studies and compares them regarding adaptation issues. The final section of the paper provides the conclusions.

2 Model-driven Transformation

The overview of proposed model-driven transformation is illustrated in Figure 2. The platform independent model (PIM) is WS-CDL meta model [7] and the platform specific model (PSM) is Reflective state meta model which includes states and transitions. An implementation of Reflective state models could be realized by the Shine framework , or by the PHP reflection , or by the Java Guarana library . This paper presents the transformation rules for converting WS-CDL meta model to UML-SM entities. UML-SM entities are used to create the Meta level of Reflective state pattern.

Reflective state pattern [8] is a refinement of State design pattern based on Reflection architectural pattern. The Reflective state pattern is used for maintaining the states of applications and providing state-dependent services to users. It uses delegation mechanism to pass user requests to meta-objects, which in turn find and consume state-dependent services. The Reflective state pattern applies the Reflection architectural pattern to implement a state machine in the Meta level, by means of meta-objects that represent state and transitions, and use the interception and materialization mechanisms for implementing the control aspects in a transparent manner. The Reflective state pattern implements the control aspects in the Meta level, separating them from the functional aspects that are implemented by the context object and the concrete states located at the Base level. As a result, the control aspects do not complicate the application design, and additionally we can modify the meta-objects at runtime and reflect the changes on the running application instances. The Controller instantiates and configure the Meta states and the Meta transitions. The Controller maintains and changes the reference to the current Meta state during transitions.

Figure 3 illustrates the Reflective state meta model. The meta model includes Meta level and Base level entities. Changes to information kept in the Meta

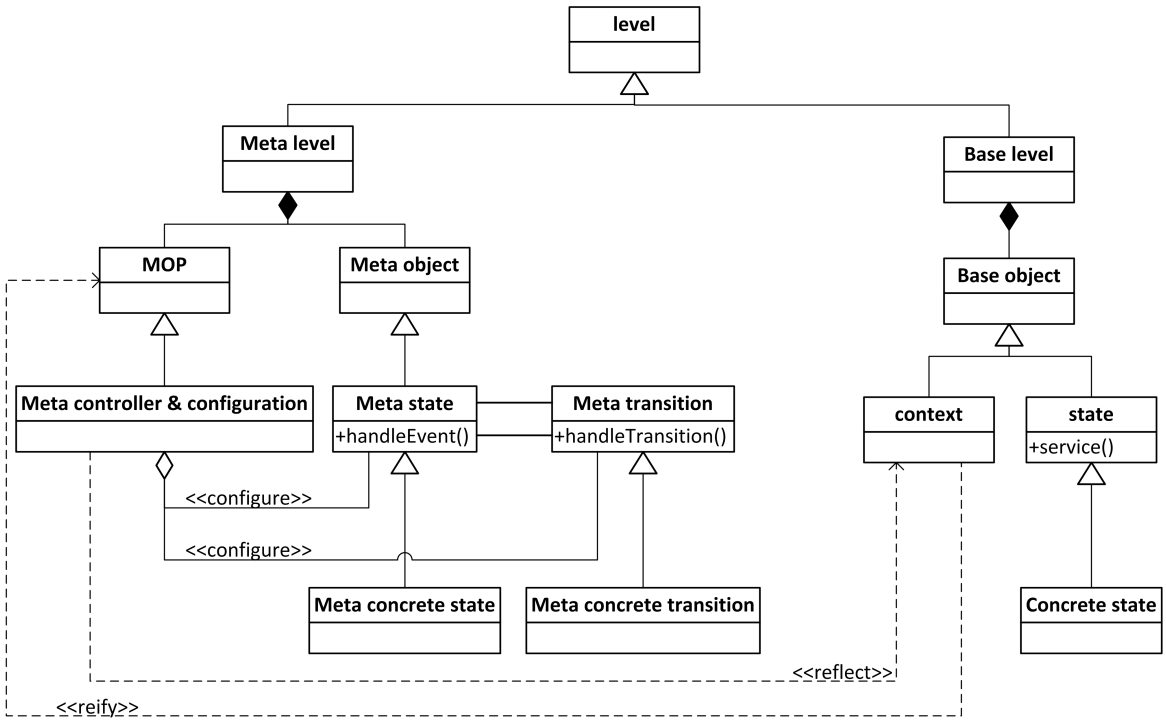

Figure 3. Reflective-state Meta Model

level affect subsequent Base level behavior. The Meta level entity includes Meta Object Protocol (MOP) and Meta object. The Meta object includes Meta states and Meta transitions which define the behaviors of application using Meta concrete states and Meta concrete transitions. Meta concrete states and Meta concrete transitions are similar to UML states and transitions, respectively. Similar to UML state machine, Meta level elements follow the event[guard condition]/action rules. Incoming events are delegated to the corresponding Meta concrete state. If the event is valid and the condition is true, then Meta concrete state performs the action and Meta concrete transition changes the current state.

3 An Overview of WS-CDL

As WS-CDL is an XML-based language that describes peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal. As shown in Figure 4, a choreography element contains activity, exception handling and finalizer parts.

Choreography: The attribute name specifies a distinct name for a choreography element. The root choreography is the only choreography that is enabled by default; it performs other non-root choreographies subsequently.

Activity: Activities describe the actions performed within a choreography. The activity notation is used to define Basic actions, Ordering Structures, and Work Unit of activities. The activity notation provides all required elements for describing services interactions, ordering of interactions, and choreography composition.

Exception handling: The exception block is used to handle performance failures. The failures emerge while an exceptional circumstance or an "error" occurs, like interaction or security failures, timeout or validation errors, etc. When an exception occurs, a work unit within the exception block is performed.

Finalizer: The finalizer block is enabled when a choreography is successfully completed. The activities within a finalizer block are performed to confirm, cancel or modify the effects of completed actions.

4 Transformation

UML 2.4 proposes behavioral and protocol state machines. The behavioral state machine shows discrete behavior of a system through finite state transitions. The protocol state machine expresses the usage protocol of a system. The following nodes and edges are typical state machine elements: behavioral state, behavioral transition, protocol state, protocol transition, and different pseudo-states such as join, fork, entry/exit points, etc. The workflow and service interaction patterns are supported by UML state machine [9].

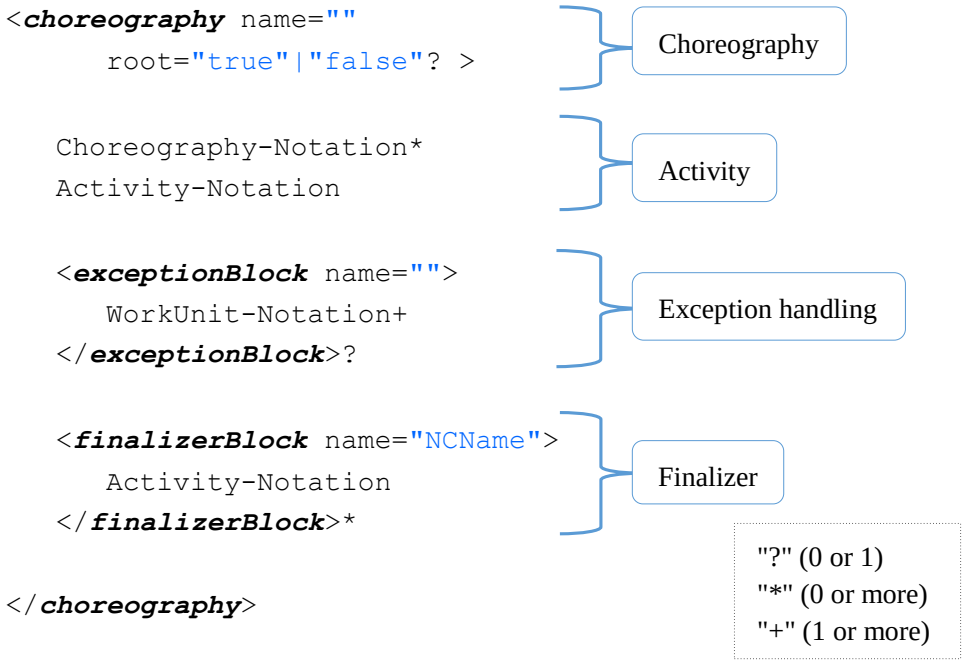

Figure 4. Structure of Choreography Element

4.1 Ordering Structures

Ordering structures are used to combine activities and express the ordering rules of actions. WS-CDL presents the Sequence, Parallel and Choice ordering structures. An ordering structure can include other ordering structures recursively; hence an activity is combined with other ordering structures in a nested way.

Sequence: The activities within a sequence element must be performed one after another. Considering activities as states, two states are performed sequentially, when there is a transition from one state to another.

Parallel: The activities within a parallel element are enabled concurrently. The parallel activity completes successfully, when all its enclosed activities complete successfully. The WS-CDL parallel element is modeled using the UML orthogonal regions. A state may be divided into orthogonal regions containing substates that execute concurrently and independently. The fork pseudo-state splits an incoming transition into two or more transitions entering the orthogonal regions. The join pseudo-state merges the transitions exiting from different orthogonal regions into one transition. As shown in Figure 5, a parallel element was corresponded with a composite state, and consider a concurrent region for each activity within the parallel element. Since the parallel element does not have a name, the composite state is labeled with a temporary name.

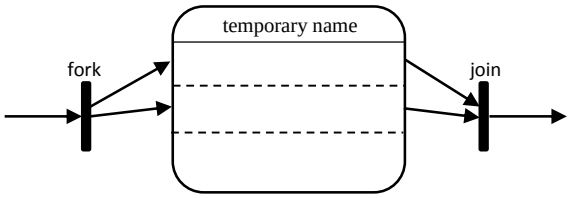

Figure 5. A Composite State with Orthogonal Regions

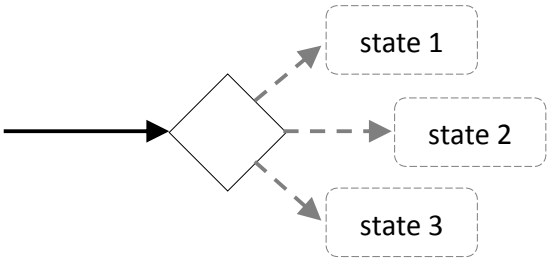

Figure 6. A Choice Pseudo-state

Choice: The choice ordering structure is similar to the UML choice pseudo-state. They both realize a dynamic conditional branch. Although the choice element encompasses one or more activities, only one activity is selected, and the other activities are disabled. The enclosed activities within a choice element are transformed into state-based elements subsequently. For example, Figure 6 is an equivalent state diagram for a WS-CDL choice element with three enclosed activities. Since each activity indicates a separate conditional branch, we transform the activity to the corresponding state-based elements.

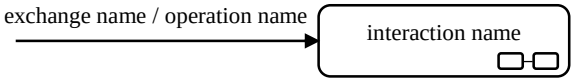

Figure 7. An Interaction is corresponded with a 'Composite State with Hidden Decomposition'

4.2 Basic Activities

A basic activity provides the lowest level actions for service interaction, choreography composition, and describing silent / hidden activities. It also provides building blocks for handling exceptions, and finalizing choreographies.

Interaction: Interaction is the most important activity of the WS-CDL specification. It leads to an information exchange between participants. In fact, an interaction is a pair of message exchanges for delivering data between a consumer and a provider, and defining the actual values of the delivered data. Furthermore, an interaction specifies the service operation that should be consumed to prepare the response mes-sage. An interaction is initiated when the consumer sends a message to the provider. Meanwhile, the provider performs the requested operation, and responds with a normal response message or a fault message. As shown in Figure 7, an interaction activity is transformed into a composite state (with hidden decomposition), and a transition entering the state.

State: To represent a running interaction between participants, we consider a state, which is labeled with the interaction name.

Trigger: When a message exchange with 'action' value equal to 'requested' is per-formed, its enclosing interaction is initiated. Therefore, we match the exception name with the transition trigger. The trigger specifies events that may induce state transition and also execution of actions.

Guard: According to the WS-CDL specification, no attribute guard-condition is specified for an interaction. Therefore, we do not consider guard condition for the state transition. Nevertheless, to define condition expressions for an interaction, the interaction must be enclosed in a work-unit.

Action: Since an operation may be performed during an interaction, the operation was corresponded with a transition's action. The action is executed when the transition is fired.

No-action, Silent-action: The no-action and silent-action activities are used, when a participant does not perform any action, or perform an action without any observable operational details, respectively. The no-action specifies a 'waiting' state for its enclosing choreography. In other words, the choreography does not perform any action, while it is waiting for an expecting event to continue. Similarly, the silent-action specifies a choreography, which is waiting for hidden operations to be completed, and then continue the performance. Consequently, as shown in Figure 8, we consider a basic state (with no action) for a no-action activity, and a composite state (with hidden decomposition) for a silent-action activity.

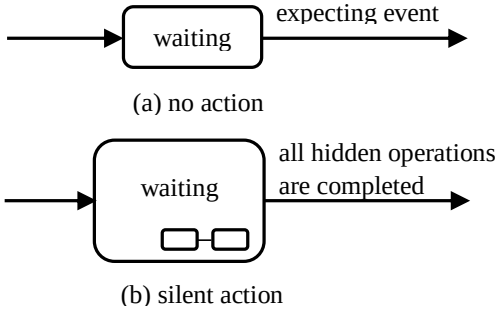

Figure 8. The 'No Action' / 'Silent Action' Activities are transformed to 'Basic State' / 'Composite State with Hidden Decomposition', respectively

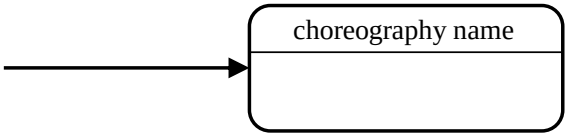

Figure 9. A Choreography Element is corresponded with a Composite State

Perform: The perform activity enables a choreography to reuse and combine other existing choreographies hierarchically. It has 'name' attribute for referencing the name of the choreography to be performed. Similarly, a composite state can include other composite or basic states in a nested way. Therefore, we correspond a perform activity with a composite state. As shown in Figure 9, the composite state is labeled with the value of the 'name' attribute. Since the performed choreography encloses activities independently, the transformation algorithm must continue recursively to convert all enclosed activities to states and display them within/inside the (enclosing/parent) composite state.

Exception block, Finalizer block: We described the exception handling and finalizer blocks in Section 3. The exception block contains one or more work-units, each work-unit handle an exceptional circumstance. The finalizer block contains required activities for finalizing its enclosing choreography performance. These blocks are simply transformed to composite states which have sub-states to cover work-units or activities. As shown in Figure 10, the composite state is labeled with the block name.

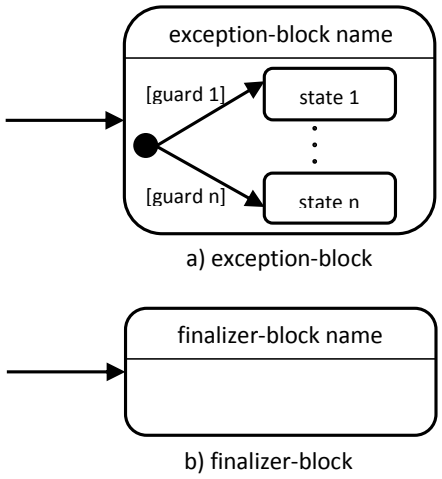

Figure 10. The Exception and Finalizer Blocks are corresponded to Composite States. Nested States should be considered subsequently for the Activities within the Blocks

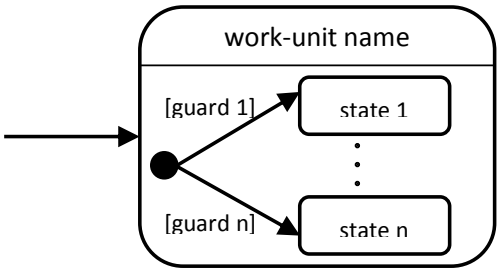

Figure 11. A Work-unit is corresponded with a Composite State. Nested States should be considered subsequently for the Activities within the Work-unit

4.3 Work Unit

A work-unit encloses activities, and defines the constraints that should be fulfilled to perform them. A work-unit has the 'guard' attribute for specifying the condition of variables in XPATH format. If the guard condition of a work-unit is satisfied, then its enclosed activities are enabled. Clearly, a work-unit is equal to a composite state and an entering transition with a guard condition. As shown in Figure 11, the composite state is labeled with the work-unit name. It also has sub-states corresponding to the work-unit activities. Each entering transition has a guard condition similar to the work-unit guard condition.

To sum up, in this section, we described the WS-CDL choreography elements, and tried to model each element's behavior through finite state transitions. In this way, we transformed 'choreography elements and attributes' into 'state-transition and event-condition-action'. Table 1 summarizes the transformation rules.

5 Prototype

Here, we adopt and extend the 'itinerary purchase' example [10], [11] for prototyping the transformation rules. The itinerary purchase process is handled by the following independent and collaborating parties: Customer, Travel Agency, Airline, Hotel, and Payment system. The itinerary purchase scenario is as follows. First, the customer requests the travel agency for available itineraries, and then the travel agency sends all available itineraries to the customer. Next, the customer selects desired itinerary and requests the travel agency for reservation. The travel agency starts two parallel choreographies with the hotel and airline parties, and waits until reservation responses arrive. If both of reservations are done, then the travel agency calculates total cost of itinerary locally (indeed, it calculates the airline, travel agency, hotel and other commissions plus the base costs). After the total cost is determined, the choreography between the travel agency and the payment system is started. Again, the travel agency waits until the payment is confirmed by the payment system. Finally a choreography is started to notify the customer about the payment and itinerary information. The choreographies of the mentioned 'itinerary purchase' CBP is shown in Figure 12.

According to the transformation rules that we mentioned in previous section, we present a transformation algorithm. A pseudo code of the transformation algorithm is shown in Appendix A. We applied the algorithm to the above itinerary purchase choreography to get its corresponding state machine. As a result, Figures 13 and 14 show the equivalent state machines of the above choreography and its exception block respectively.

6 Semantic Preservation

Providing proof techniques for showing full semantic preservation of model trans-formation is a very difficult problem, on which there has been little work so far [12]. Instead of generally proving total correctness of a transformation, a number of approaches, carry out run-time checks of equivalence between a given source and generated target model, that is, partial correctness. Here we describe semantic preservation of state machine models in order to prove total correctness of proposed transformation rules. The semantic of source models (*i.e.* WS-CDL) is preserved, if transformation rules produce behaviorally equivalent target models (*i.e.* UML-SM). In the following list, we show that our proposed transformation rules preserve ordering of messages and semantic of interactions.

- **Ordering and Composing Structures:** WS-CDL's ordering and composing structures are corresponded with state based elements in a straight-

```
<choreography name="itineraryPurchase" root="true">
  <sequence>                                              Interaction block
/* (1) customer , travel agency */
    <interaction name="itinerary" operation="getItineraries">
      <participate relationshipType="Customer_TravelAgency"
       fromRole="CustomerRole" toRole="TravelAgencyRole" />
      <exchange name="requestItineraries" action="request">
        <send variable="tripProfile"/>
        <receive variable="tripProfile"/>
      </exchange>
      <exchange name="itinerariesList" action="respond">
        <send variable="itinerariesList"/>
        <receive variable="itinerariesList"/>
      </exchange>
    </interaction>

/* (2) customer , travel agency */
    <perform choreographyName="requestReservation"></perform>
/* (3) travel agency , airline | travel agency , hotel */
    <perform choreographyName="itineraryReservation"></perform>
/* (4) travel agency , payment system */
    <perform choreographyName="paymentProcessing"></perform>
  </sequence>

  <exceptionBlock name="exceptionHandling">               Exception block
    <workunit guard="cancel">
      <sequence>
        <perform choreographyName="itineraryCancelation"></perform>
        <perform choreographyName="cancelNotification"></perform>
      </sequence>
    </workunit>
    <workunit guard="handleTimeout">
      <noAction>
    </workunit>
  </exceptionBlock>

  <finalizerBlock>                                        Finalizer block
    <workunit name="finalizing">
/* (5) travel agency , customer */
      <perform choreographyName="successNotification"></perform>
    </workunit>
  </finalizerBlock>
</choreography>

<choreography name="itineraryReservation">
  <parallel>
/* (3.1) travel agency , airline */
    <perform choreographyName="flightReservation"></perform>
/* (3.2) travel agency , hotel */
    <perform choreographyName="roomReservation"></perform>
  </parallel>
</choreography>
```

Figure 12. The Specification of Itinerary Purchase Process in WS-CDL Format

Table 1. WS-CDL to UML State Machine Mapping Table

WS-CDL	UML state machine	Label	Event [Guard condition] / Action
Root choreography	Initial state and composite state	Choreography name	-
Enclosed choreography	Initial state	-	-
</Choreography>	Final state	-	-
Interaction	Composite state with hidden decomposition	Interaction name	Exchange name [null] / operation name
Perform	Composite state	Choreography name	-
No-action	Basic state	Waiting	Exception name [null] / null
Silent-action	Basic state	Waiting	-
Sequence	Transitions	-	-
Parallel	Fork - Orthogonal regions - Join	Temporary name	-
Choice	Choice	-	-
Finalizer block	Composite state	Finalizer block name	-
Exception block	Composite state	Exception block name	-
Work unit	Composite state	Work unit name	Null [guard name] / null

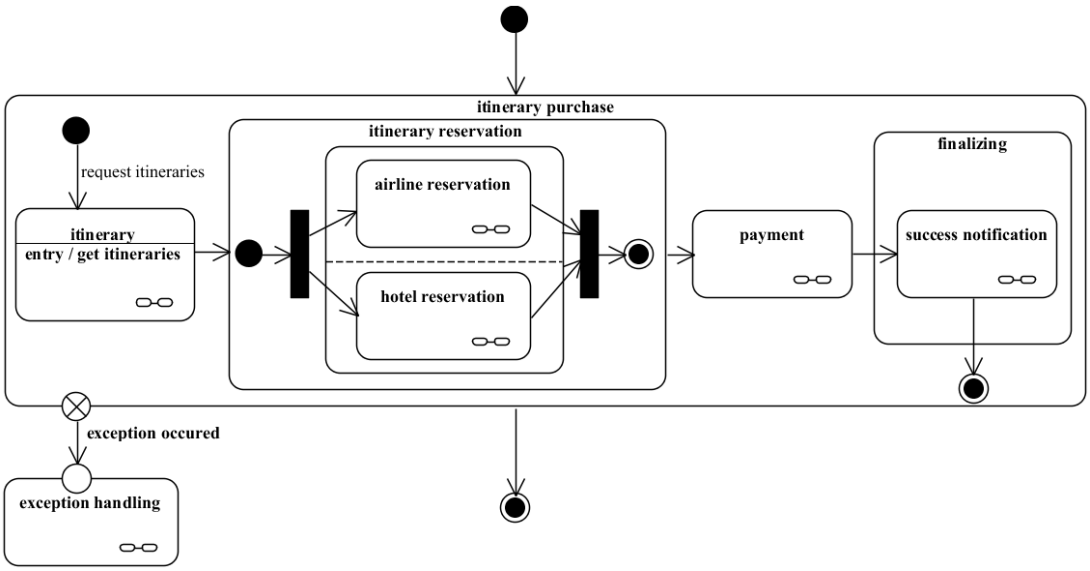

Figure 13. Itinerary Purchase State Machine

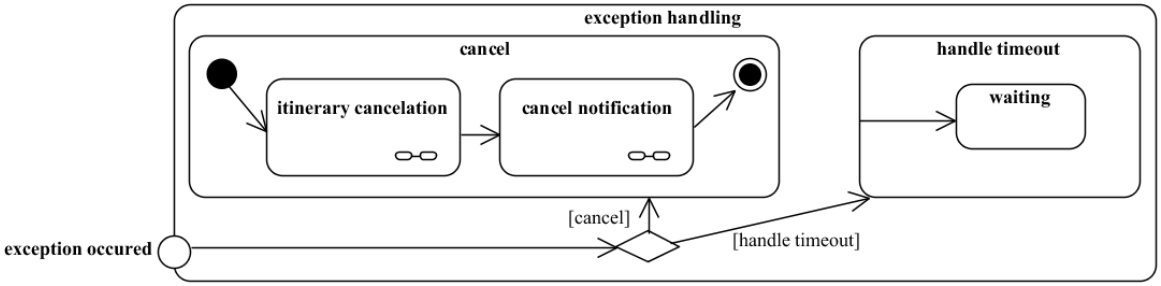

Figure 14. Exception Handling Composite State

forward form (see the Transformation section).

- **Flow Control:** To control flow of messages, WS-CDL uses guard conditions in Exception Block and Work Unit. Similarly, UML-SM controls flow of messages by evaluating guards associated with transitions.
- **Interaction:** Choreography interaction includes service invocation between two partners in which one partner requests for an operation, and the other partner executes the operation and reply. Here we show how to realize service invocation be-tween two partners using UML-SM. In particular, we expand 'interaction compo-site state' shown in Figure 7, to expose its enclosed states, transitions and series of events. As shown in Figure 15, a consumer enters 'waiting for response' state while sending a request event to provider. The request event is labeled with operation name and induces the provider to exit 'waiting for request' state and execute requested state machine. Upon receipt of response event, the consumer exits from 'waiting for response' state and continues its state machine.

7 Related Work

There are two types of transformation including model-driven (with the goal of adaptation) and formal (with the goal of verification) in the literature. The model driven approaches translate a WS-CDL element to its respective replacement in terms of BPEL as well as WSCDL. This enables tracing down changes from choreography to orchestration and vise versa which is an important issue in the choreography adaptation scope. On the other hand, some studies formalize the WS-CDL elements. They tried to verify several aspects of service choreography like protocol compatibility, time constraints, and message ordering. It might also be observed that these works are limited to a specific subject and does not check whether the committed choreography is realizable by the existing services protocols at the orchestration level.

Mendling & Hafner [13] propose a model driven transformation approach to drive BPEL process definitions from a global WS-CDL model. It proposes a mapping between WS-CDL and WSBPEL building blocks. In addition, the mapping can be used to generate WS-CDL description from existing WSBPEL processes. In another model-driven approach, CDL2BPEL [14] algorithm translates WS-CDL to BPEL and WSDL elements, according to a knowledge base. The knowledge base contains generic patterns to translate a WS-CDL entity to its respective replacements in terms of BPEL as well as optional WSDL. The algorithm extracts WSDL interfaces from interactions and tokens / token locators. BPEL4Chor [15] is an intermediary language to align choreography and orchestration. BPEL4Chor is a non-executable choreography language forming an additional layer on top of the BPEL standard [16]. The idea of mapping WS-CDL to UML-SM was firstly proposed by Zakerfar et al. [6]. The interoperability between choreography and orchestration was not considered in their work. Therefore the Perform element, the noAction element, the silentAction element have no corresponding UML-SM entity, and the finalizer block is corresponded with the final state. They did not convert the exceptionHandling block for adaptation, neither did they expanded the interaction activity at orchestration level. We also mentioned the semantic preservation of transformation that has not been described in related studies.

CDL [17] was introduced to formalize the WS-CDL's participant roles, and the collaborations among roles. They used SPIN model-checker to reason about properties that should be satisfied by the specified system automatically. Furthermore, in order to verify WS-CDL protocol mismatches, the transformation rules were proposed to correspond the WS-CDL entities with Timed automata [18], and Colored Petri-net [19], [20] elements. A formal specification of WS-CDL was proposed by Le & Truong [21], using Event-B formalism. They used Rodin studio to verify the properties of translated models such as deadlock and ordering of messages. Yu et al. [22] proposed an algorithm that transforms service choreography model and domain rules into description logic ontology. They verified the output model by semantic reasoners such as Pellet. These formal languages are suitable for choreography

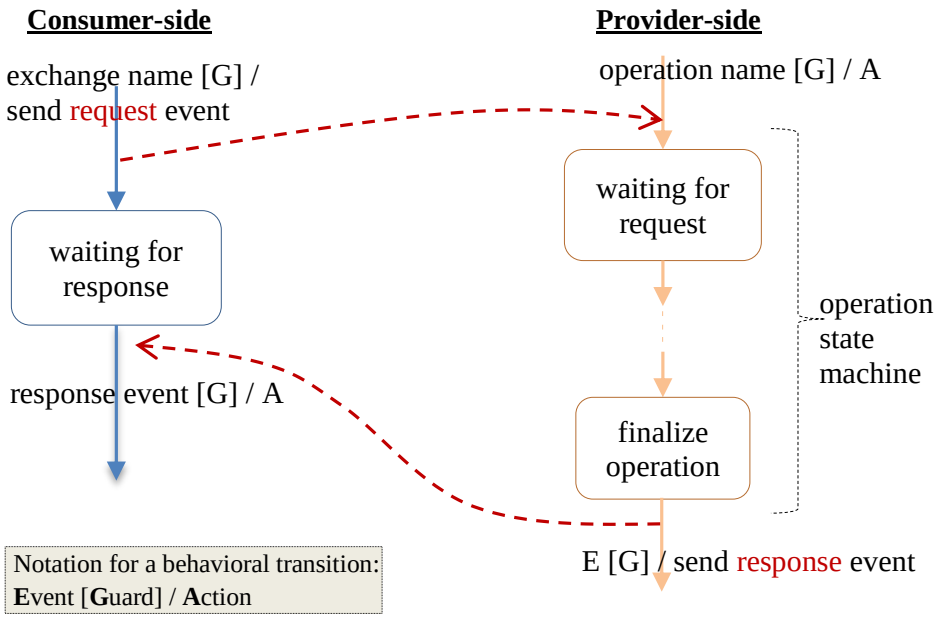

Figure 15. A Realization of Service Invocation using UML-SM (Expanded Version of Figure 7)

verification, but they cannot realize the requirements of an adaptive process. For example, CDL and timed automata do not support all workflow patterns; Colored petri-net does not support the separation of business logic and implementation code, nor abstract modeling, nor distinct control model. From the adaptation point of view, we consider the below attributes to compare the choreography description languages/models. The comparison results are depicted in Table 2.

- **Structure**
  - Dynamic. Dynamic structure means that the structure of a process must be flexible to being reconfigured and regulated dynamically in response to commands of management activities.
  - Workflow support. It refers to supporting both workflow and services interaction patterns (*e.g.*, sequence, parallel, synchronization, send, receive, etc.).
  - Hierarchical (nested). A hierarchical process is designed level by level, for hiding unnecessary details at each abstraction level. At each level, there is a composite operation that may be broke down at the next lower level.
  - Separation of concerns. Separation of concerns [23] enables the separate development of the business logic and the cross-cutting concerns of a process (*e.g.*, quality of service, implementation code).
- **Management**
  - Manageable. A process is manageable, when its structure is reconfigured by, and its runtime behaviors are regulated by, management rules and protocol.
  - Verifiable. Choreography verification consists of two main types of protocol mismatches. Service interoperability verification including message ordering and time constraints at design time [20]. Deadlock, in which both parties are mutually waiting to receive some message from the other [24].

This section could be concluded by highlighting the differences of our work as follows. 1) This work proposed a transformation of WS-CDL to UML-SM with the goal of adaptation. Therefore, the transformation rules provide the interoperability between WS-CDL and WSBPEL and support the traceability of changes from choreography to orchestration and vice versa. 2) To fulfill the adaptation concern, we used Reflective state pattern which hold state machines in its Meta level. Consequently, after transforming WS-CDL to UML-SM, the output models are deployed at the Meta level and the implementation code for each action is deployed at the Base level. 3) We described the semantic preservation of transformation rules in Section 6. We also expanded the WS-CDL's Interaction activity to describe its enclosing states at the orchestration level. 4) We surveyed the most recent related studies from the adaptation point of view. We concluded that there are two types of transformation including model-driven (with the goal of adaptation) and formal (with the goal of verification) in the literature. The comparison results are depicted in Table 2.

Table 2. Comparison of Choreography Modeling and Description Languages

Language/Model	Goal	Dynamic	Workflow Support	Hierarchical	Separation of Concerns	Manageable	Verifiable
WS-CDL [3]	Specification	-	●	✓	-	-	-
WSCI [25]	Specification	-	●	-	-	-	-
BPEL4Chor [15]	Specification, Execution	-	●	✓	-	-	-
CDL [17]	Specification, Verification	-	-	-	-	●	✓
UML state machine [6], and this work	Modeling, Specification, Verification	✓	✓	✓	✓	✓	●
Timed automata [18]	Verification	✓	●	-	-	●	✓
Colored Petri net [19], [20], Event-B [21], Description Logic [22]	Verification	✓	✓	●	●	●	✓

Complete support (✓) , Partial support (●) , Lack of support (-)

8 Conclusion

In this paper, we studied the required characteristics of an adaptive service-based CBP such as dynamicity, verifiability, manageability, etc. As a result, we selected UML state machine for modeling adaptive CBPs in form of Reflective state pattern. We proposed a model-driven approach to transform the WS-CDL specification into UML state machine. We presented the transformation rules and the rationale behind each rule. The benefits of the transformation include:

- The adaptation strategies like reconfiguration, reselection are easily realized by modifying the Meta level's state machines. For example, by adding / removing / merging / replacing states and transitions.
- We can suspend the failed instance of a process for adaptation, and then, resume it without interrupting other process instances.
- We can separately develop the business process and policies, and reflect their effects on the running process instances.

The limitations of the transformation include:

- As mentioned in Section 7, UML-SM is partially suitable for verifying choreography mismatch patterns like deadlock, unspecified reception, or order of messages. Indeed, our model is mostly suitable for process adaptation rather than process verification.
- To perform the output models, both service consumer and service provider must be equipped with a middle-ware that follows the Reflective state design pattern.
- Describing processes in state-based models like UML-SM is more complex than BPMN or UML activity diagram which simply shows the order of activities.

In future, we will develop a software tool that implements the pseudo-code of transformation algorithm. We will also propose the transformation of WSBPEL (*i.e.*, an orchestration language) into UML state machine. Then, we can transform both WS-CDL and WSBPEL into their corresponding state machines, and integrate them in a nested way. According to the reflective-state design pattern [8], we will deploy the state machines on Meta level, and their implementation on Base level. We will consider concrete states and services to realize the functionalities that are defined at Meta level. Consequently, the adaptation designer (or an automatic adaptation unit) can easily modify Meta level which mirrors the behaviors of each process instance distinctly.

References

[1] Prachet Bhuyan, Abhishek Ray, and Durga Prasad Mohapatra. A service-oriented architecture (soa) framework component for verification of choreography. In *Computational Intelligence in Data Mining-Volume 3*, pages 25–35. Springer, 2015.

[2] Johann Eder and Amirreza Tahamtan. Temporal conformance of federated choreographies. In

Database and Expert Systems Applications, pages 668–675. Springer, 2008.

[3] W3C. Web services choreography description language version 1.0, 2005. URL `https://www.w3.org/TR/ws-cdl-10/`.

[4] Zhen Li and Manish Parashar. Enabling dynamic composition and coordination for autonomic grid applications using the rudder agent framework. *The Knowledge Engineering Review*, 21(03):221–230, 2006.

[5] Yousef Rastegari and Fereidoon Shams. Toward automatic transformation of service choreography into uml state machine (poster). In *poster presented at 6th IPM International Conference on Fundamentals of Software Engineering*, Tehran, Iran, Apr. 22-24 2015. URL `http://fsen.ir/2015/files/Preproceedings.pdf`.

[6] Seyed Mohammad Javad Zakerfar, Naser Nematbakhsh, Farhad Mardukhi, and Mohammad Naderi Dehkordi. Mapping uml state machine diagram and ws-cdl for modeling participants behavioral scenarios.

[7] Alistair Barros, Marlon Dumas, and Phillipa Oaks. A critical overview of the web services choreography description language. *BPTrends Newsletter*, 3:1–24, 2005.

[8] Luciane Lamour Ferreira and Cecília MF Rubira. The reflective state pattern. *Proceedings of the Pattern Languages of Program Design, TR# WUCS-98-25, Monticello, Illinois-USA*, 1998.

[9] Azadeh Mellat, Naser Nematbakhsh, Ahmad Farahi, and Farhad Mardukhi. Suitability of uml state machine for modeling choreography of services. *International Journal of Web & Semantic Technology*, 2(4):33, 2011.

[10] Natália Silva, Renata Carvalho, Ricardo Lima, and Cesar Oliveira. Integrating declarative processes and soa: A declarative web service orchestrator. In *Proceedings of the International Conference on Semantic Web and Web Services (SWWS)*, page 5. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[11] Anis Charfi and Mira Mezini. Hybrid web service composition: business processes meet business rules. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 30–38. ACM, 2004.

[12] Mathias Hülsbusch, Barbara König, Arend Rensink, Maria Semenyak, Christian Soltenborn, and Heike Wehrheim. Showing full semantics preservation in model transformation-a comparison of techniques. In *Integrated Formal Methods*, pages 183–198. Springer, 2010.

[13] Jan Mendling and Michael Hafner. From ws-cdl choreography to bpel process orchestration. *Journal of Enterprise Information Management*, 21(5):525–542, 2008.

[14] Ingo Weber, Jochen Haller, and Jutta A Mulle. Automated derivation of executable business processes from choreographies in virtual organisations. *International Journal of Business Process Integration and Management*, 3(2):85–95, 2008.

[15] Gero Decker, Oliver Kopp, Frank Leymann, and Mathias Weske. Bpel4chor: Extending bpel for modeling choreographies. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 296–303. IEEE, 2007.

[16] Andreas Weiß, Dimka Karastoyanova, David Molnar, and Siegfried Schmauder. Coupling of existing simulations using bottom-up modeling of choreographies. In *GI-Jahrestagung*, pages 101–112, 2014.

[17] Hongli Yang, Xiangpeng Zhao, Zongyan Qiu, Geguang Pu, and Shuling Wang. A formal model forweb service choreography description language (ws-cdl). In *null*, pages 893–894. IEEE, 2006.

[18] Gregorio Diaz, Juan-José Pardo, María-Emilia Cambronero, Valentin Valero, and Fernando Cuartero. Automatic translation of ws-cdl choreographies to timed automata. In *Formal Techniques for Computer Systems and Business Processes*, pages 230–242. Springer, 2005.

[19] Valentín Valero, Hermenegilda Macià, Juan José Pardo, María Emilia Cambronero, and Gregorio Díaz. Transforming web services choreographies with priorities and time constraints into prioritized-time colored petri nets. *Science of Computer Programming*, 77(3):290–313, 2012.

[20] Maya Souilah Benabdelhafid and Mahmoud Boufaida. Toward a better interoperability of enterprise information systems: a cpns and timed cpns-based web service interoperability verification in a choreography. *Procedia Technology*, 16: 269–278, 2014.

[21] Hong Anh Le and Ninh-Thuan Truong. Modeling and verifying ws-cdl using event-b. *ICCASA*, 109: 290–299, 2012.

[22] Minggang Yu, Zhixue Wang, and Xiaoxing Niu. Verifying service choreography model based on description logic. *Mathematical Problems in Engineering*, 2016, 2016.

[23] Philip K McKinley, Seyed Masoud Sadjadi, Eric P Kasten, and Betty HC Cheng. A taxonomy of compositional adaptation. *Rapport Technique numéroMSU-CSE-04-17*, 2004.

[24] Woralak Kongdenfha, Hamid R Motahari-Nezhad, Boualem Benatallah, and Regis Saint-Paul. Web service adaptation: Mismatch patterns and semi-automated approach to mismatch identification and adapter development. In *Web*

Services Foundations, pages 245–272. Springer, 2014.

[25] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, et al. Web service choreography interface (wsci) 1.0. *Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems*, 2002.

A Pseudo-code of Transformation Algorithm

```
#= ABBREVIATIONS =
/* CS (Composite State),
HCS (a Composite State with Hidden decomposition),
BS (Basic State),
F (Fork),
J (Join),
IS (Initial State),
FS (Final State)
*/

#= START TRANSFORMATION =
SET query to '/package/choreography/[@root=true]'
CALL findTag with query
CALL TRANSFORM

#= SUB-MODULES =
string FUNCTION readTag ()
Extract the tag referred by read-pointer
Update read-pointer to next tag

void FUNCTION findTag (query)
Find the tag according to query
Update read-pointer

void FUNCTION draw (object, name, event, guard,
action)
Find the tag according to query
Update read-pointer

#= MAIN MODULE =
void FUNCTION TRANSFORM ()
CALL readTag RETURNING tag

IF tag EQUAL 'choreography' THEN
CALL draw with 'IS'
CALL TRANSFORM
ENDIF

IF tag EQUAL '</choreography>' THEN
CALL draw with 'FS'
RETURN
ENDIF

IF input EQUAL tag THEN
RETURN '</'+tag+'>'
ENDIF

/* basic activities */
CASE tag OF

'perform':
IF nested states must be shown THEN
SAVE read-pointer
SET query to '/package/choreography/ [@name =
perform.choreographyName]'
CALL findTag with query
CALL draw with 'CS'
CALL TRANSFORM
RESUME read-pointer
ELSE
CALL draw with 'HCS'
ENDIF

'interaction':
CALL draw with 'HCS'

'noaction':
CALL draw with 'BS'

'silentaction':
CALL draw with 'BS'

ENDCASE

/* ordering structures */
IF tag EQUAL 'sequence' OR 'parallel' OR
'choice' THEN

IF tag EQUAL 'parallel' THEN
CALL draw with 'F'
ENDIF

IF tag EQUAL 'choice' THEN
CALL draw with 'C'
ENDIF

REPEAT
CALL TRANSFORM RETURNING output
UNTIL output NOT EQUAL '</'+tag+'>'

IF tag EQUAL 'parallel' THEN
CALL draw with 'J'
ENDIF

ENDIF
```

Yousef Rastegari is PhD candidate at Department of Computer Engineering and Science, Shahid Beheshti University. He is member of two research groups namely ASER (Automated Software Engineering Research) (aser.sbu.ac.ir) and ISA (Information Systems Architecture) (isa.sbu.ac.ir).

Fereidoon Shams has received his PhD in Software Engineering from the Department of Computer Science, Manchester University, UK, in 1996 and his M.S. from Sharif University of Technology, Tehran, Iran, in 1990. His major interests are Software Architecture, Enterprise Architecture, Service Oriented Architecture, Agile Methodologies, Ultra-Large-Scale (ULS) Systems and Ontological Engineering. He is currently an Associate Professor of Software Engineering Department, Shahid Beheshti University of Iran. Also, he is heading two research groups namely ASER (Automated Software Engineering Research) (aser.sbu.ac.ir) and ISA (Information Systems Architecture) (isa.sbu.ac.ir) at Shahid Beheshti University.