Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 8947157, 20 pages
http://dx.doi.org/10.1155/2016/8947157

*Research Article*

# Verifying Service Choreography Model Based on Description Logic

**Minggang Yu, Zhixue Wang, and Xiaoxing Niu**

*Institute of Command Information System, PLA University of Science and Technology, Nanjing 210007, China*

Correspondence should be addressed to Zhixue Wang; wzxcx@163.com

Received 9 May 2015; Revised 23 June 2015; Accepted 14 July 2015

Academic Editor: Jean-François Monin



Web Services Choreography Description Language lacks a formal system to accurately express the semantics of service behaviors and verify the correctness of a service choreography model. The paper presents a new approach of choreography model verification based on Description Logic. A metamodel of service choreography is built to provide a conceptual framework to capture the formal syntax and semantics of service choreography. Based on the framework, a set of rules and constraints are defined in Description Logic for choreography model verification. To automate model verification, the UML-based service choreography model will be transformed, by the given algorithms, into the DL-based ontology, and thus the model properties can be verified by reasoning through the ontology with the help of a popular DL reasoner. A case study is given to demonstrate applicability of the method. Furthermore, the work will be compared with other related researches.

## 1. Introduction

Web service technology has been popularly applied due to its power of interoperation, which allows various applications to run on heterogeneous platforms. Standards for web service composition cover two different levels of view: choreography and orchestration [1, 2]. The choreography view describes the interactions between services from a global perspective, while the orchestration view focuses on the interactions between one party and others. The web service choreography description language (WS-CDL) [3] is an XML-based language for the description of peer-to-peer collaborations of participants from a global viewpoint. However, WS-CDL is a declarative language and the specified concepts are weakly constrained. It lacks a formal system to accurately express the semantics of service behaviors and verify the correctness of a service choreography model. As a result, the built models may suffer from the problem of inconsistency, conflict, and realizability.

A number of approaches are suggested for formally modeling and verifying web services composition. Xiao et al. [4] proposed a process algebra called probabilistic priced process algebra (PPPA) for modeling and analyzing web service composition from both functionality and nonfunctionality, such as reliability and performance. Moreover, they provided a united method based on PPPA to model and analyze both functionality and QoS of web service composition. Cambronero et al. [5] presented an approach to validation and verification of web services choreographies and more specifically for composite web services systems with timing restrictions. They defined operational semantics for a relevant subset of WS-CDL and then provided a translation of the considered subset into a network of timed automata for the validation and verification using the UPPAAL tool. Zhou et al. [6] put forward an approach to testing WS-CDL programs automatically. The dynamic symbolic execution technique was used to generate test inputs, and assertions are treated as the test oracles. An engine that can simulate WS-CDL is used to execute the WS-CDL programs during symbolic execution. Besson et al. [7] adapted the automated testing techniques used by the Agile Software Development community to the SOA context to enable test-driven development of choreographies. They present the first step in that direction, a software

prototype composed of ad hoc automated test case scripts for testing a web service choreography. Gu et al. [8, 9] originally advocated for a formal modeling framework, called Abstract WS-CDL. This includes grammar, congruence relations, and operational semantics. They defined a set of mappings of the Abstract WS-CDL global model to the Pi calculus-based local model and subsequently suggested a set of deductive reasoning rules of state reachability and terminability.

One of the crucial questions in choreography-based development is to check the realizability and conformance properties. Hallé and Bultan [10] proposed a novel algorithm for deciding realizability by computing a finite state model that keeps track of the information about the global state of a conversation protocol that each peer can deduce from the messages it sends and receives. McNeile [11] provided a new technique that uses compositions of partial descriptions to define a choreography, and he demonstrated that realizability of a choreography defined as a composition only needs to be established individually for the components of the composition. Basu et al. [12] gave necessary and sufficient conditions for realizability of choreographies and implemented the proposed realizability check on three granularities: (1) web service choreographies, (2) singularity OS channel contracts, and (3) UML collaboration (communication) diagrams. Yeung [13] put forward a formal approach to web service composition and conformance verification based on WS-CDL and WS-BPEL. The main contributions included a precise notion of choreography conformance upon which verification is based and support for the complementary use of visual modeling (e.g., UML) and standard WS-$^*$ notations in composition.

We argue that checking the realizability and conformance properties should be approached in two ways. One is to check the completeness and consistency of a choreography specification to guarantee there are no logical conflicts in the specification. The other is to check whether the specified choreography behaves in a correct way or whether the system functions well to complete its jobs. Having investigated the recent research results, we found that most of them addressed only one side rather than both.

To solve the problem above, we propose an approach of choreography model verification based on Description Logic, or CMV-DL. A service choreography modeling framework is provided to support UML-based modeling. The metamodel of service choreography extends the WS-CDL specification. Two algorithms are given to transform the UML-based service choreography model into the DL-based ontology through which the verification can be made using a DL reasoner.

To enable automatic verification, we introduce Description Logic (DL) [14] to formalize the choreography model and define a set of rules and constraints for verification. DL is a knowledge representation language and is a decidable subset of first-order predicate logic. It contains a set of basic concept constructors, such as concept consumption ($\sqsubseteq$) and universal constraint ($\forall$). As a subset of DL, *SHOIN*(D) [15] is the logical basis of web ontology language (OWL) [16]. It is powerful in knowledge description ability, reasoning decidability, and knowledge reusability and more importantly there are available supporting reasoners such Pellet [17] and Racer.

The rest of the paper is organized as follows. Section 2 introduces the service choreography modeling framework, highlighting the metamodel of service choreography. Section 3 discusses CMV-DL further on model verification mechanism and UML-DL conversion and introduces the deductive reasoning rules specified in Semantic Web Rule Language (SWRL) [18]. Section 4 provides a case study to show the potential usage of CMV-DL. Section 5 investigates the related work and draws a comparison between CMV-DL and other choreography verification methods. The final section brings a conclusion and foresees our future work.

## 2. Service Choreography Modeling Framework

According to the OMG's four-layered metamodel architecture, UML can be extended by defining new stereotypes at metamodel level (M2) [19] and thus becomes a domain-specific language. The modeling framework for service choreography is accordingly defined in two levels, as shown in Figure 1. The metamodel of service choreography is built by extending the concepts of WS-CDL to provide the syntax and semantics for visually presenting a choreography and define a set of domain rules for model checking. The application model specifies the concepts of service choreography of an application (or a system) by instantiating the concepts of the metamodel.

*2.1. Metamodel of Service Choreography.* To build the metamodel of service choreography, we define the main concepts of service choreography by referring to the WS-CDL specification.

*Definition 1* (session (*S*)). A session is composed of a set of basic interactive activities which are performed by one or more participants in order to complete a specific function. Sessions can be denoted as a four-tuple structure: $S = \langle N, Act, Ro, Pr\rangle$, where $N$ is the name, $Act$ refers to the activities that realize this specific function, $Ro$ is a set of the roles that participate in the activities, and $Pr$ is a set of the preconditions for carrying out this session.

*Definition 2* (choreography (*Cho*)). A choreography defines the collaboration contracts between the participants and the interoperations of cross-system behaviors. It can be denoted as a three-tuple structure: $Cho = \langle N, Ro, S\rangle$, where $N$ is the name, $Ro$ is a set of the roles that participate in the interoperations, and $S$ represents a set of sessions to be executed in the interoperation processes.

*Definition 3* (metamodel). The metamodel of service choreography defines the syntax and semantics for visually presenting a choreography and provides a set of domain rules for checking the correctness properties of a choreography model. It is composed of three parts: $\langle$*MetaConcept*, *MetaRelation*, *DomainRule*$\rangle$, where *MetaConcept* and *MetaRelation* are sets of metaconcepts and metarelations which are defined by inheriting the counterpart concepts from the WS-CDL

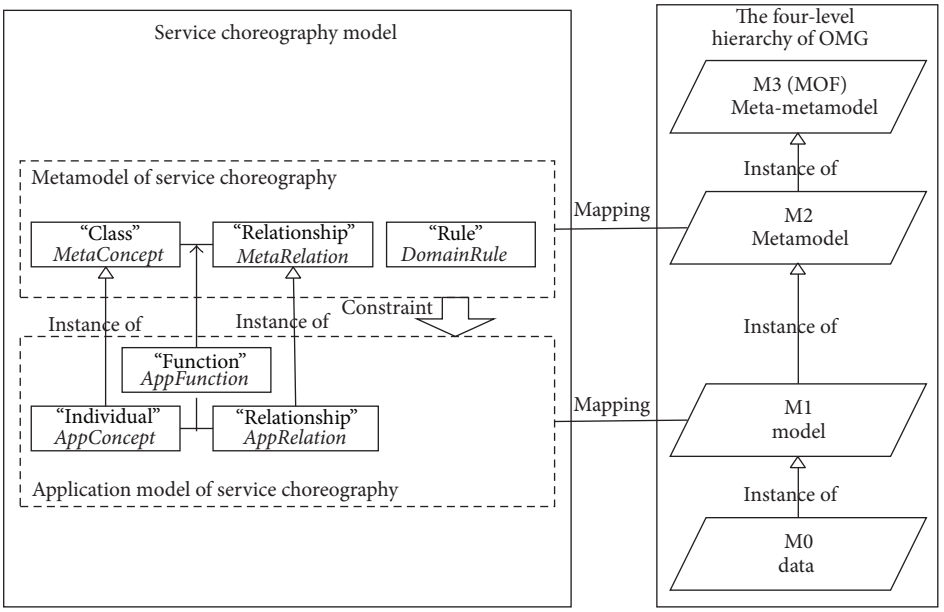



Figure 1: Service choreography modeling framework.

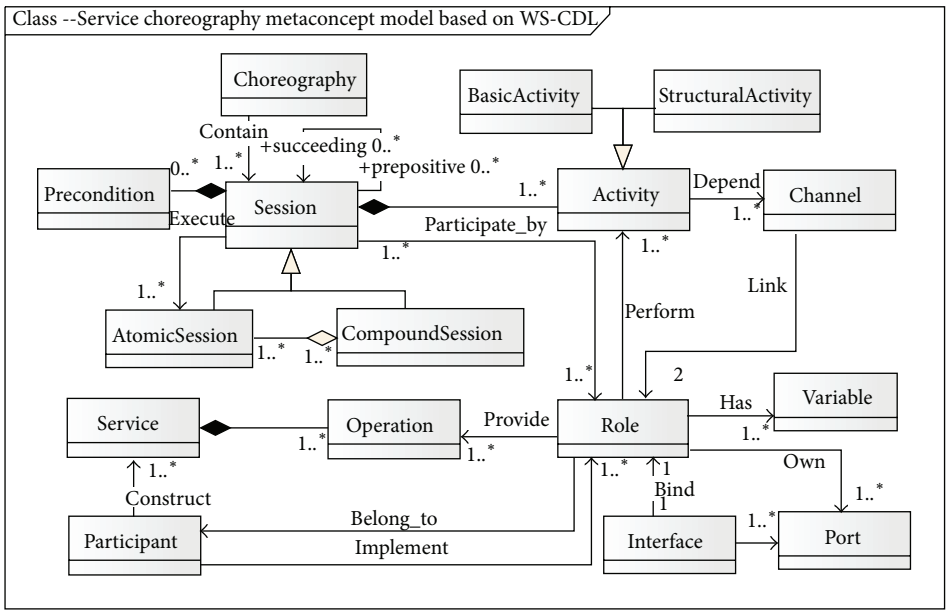



Figure 2: Metamodel of service choreography based on WS-CDL.

specification. The core elements of the metamodel are shown in Figure 2.

*2.1.1. MetaConcept.* *MetaConcept* is a finite set of metaconcepts that originate from WS-CDL but are not limited to it. New concepts, such as Session, AtomicSession, and CompoundSession, are extended for the purpose of formal modeling and verification. Table 1 provides detailed definitions and descriptions of the metaconcepts.

The concept activity can be divided into two kinds: BasicActivity (*Abas*) and StructuralActivity (*Astr*):

$$Act ::= Abas \mid Astr \tag{1}$$

*Abas* is defined as

$$Abas ::= NoAction \mid SilentAction \mid Interaction \mid Assign \mid Perform \tag{2}$$

*Astr* is used to describe a compound activity in following syntax and semantics:

$$\begin{aligned} Astr ::= & \ [p]\, Act && (condition) \\ & \mid [p] * Act && (repeat) \\ & \mid \left[p_g\right]\left[p_{rep}\right] * Act && (workunit) \\ & \mid Act_1 \cdot Act_2 && (sequence) \\ & \mid Act_1 \parallel Act_2 && (parallel) \\ & \mid Act_1 \mathrm{T} Act_2 && (non\text{-}deterministic) \\ & \mid [p_1]\, Act_1 + [p_2]\, Act_2 && (choice) \end{aligned} \tag{3}$$

The structure of a compound activity appears either in workunit pattern or in control-flow pattern. The workunit structure is defined in WS-CDL by three substructures. The condition structure is expressed as $[p]Act$. The repeat structure is expressed as $[p] * Act$. A workunit structure is expressed as $[p_g][p_{rep}] * Act$, which means that the activity will be blocked until the precondition $p_g$ is evaluated to be "true"; that is, the activity is triggered by the precondition. If *Act* terminates successfully and the repetition condition $p_{rep}$ is "true," the workunit will repeat; otherwise, it will finish. A control-flow structure is defined as any combination of sequentially executed activities $Act_1 \cdot Act_2$, parallel executed activities $Act_1 \parallel Act_2$, nondeterministically executed activities $Act_1 \mathrm{T} Act_2$, or selectively executed activities $[p_1]Act_1 + [p_2]Act_2$.

The concept session can also be divided into two kinds: AtomicSession ($S_{atom}$) and CompoundSession ($S_{com}$):

$$S ::= S_{atom} \mid S_{com} \tag{4}$$

An atomic session may appear in any form of the following:

$$\begin{aligned} S_{atom} ::= & \ Ro\,(\emptyset) && (S_{no}) \\ & \mid Ro\,(\tau) && (S_{silent}) \\ & \mid assign\,(Ro \cdot x = e) && (S_{assign}) \\ & \mid request\,(Ro_1 \cdot x \longrightarrow Ro_2 \cdot y, Ch@Ro_2) && (S_{req}) \\ & \mid respond\,(Ro_1 \cdot x \longleftarrow Ro_2 \cdot y, Ch@Ro_1) && (S_{resp}) \\ & \mid req\text{-}resp\,(Ro_1 \cdot x \longrightarrow Ro_2 \cdot y, Ro_1 \cdot v \longleftarrow Ro_2 \cdot u, Ch@Ro_2) && (S_{req\text{-}resp}) \\ & \mid Cho\,(x_1, x_2, x_3, \ldots) && (S_{perform}) \\ & \mid \psi && (NULL) \end{aligned} \tag{5}$$

$S_{no}$ describes that the role *Ro* does not perform any operation; $S_{silent}$ means that *Ro* performs an internal silent action $\tau$; $S_{assign}$ describes an assignment operation that the value $e$ is assigned to the variable $Ro \cdot x$; $S_{req}$ describes a request interaction from $Ro_1$ to $Ro_2$ through $Ch@Ro_2$, where the request message is sent from $Ro_1 \cdot x$ to $Ro_2 \cdot y$; $S_{resp}$ describes a response interaction; $S_{req\text{-}resp}$ describes request-response interaction; $S_{perform}$ describes invoking operation between sessions; and *NULL* denotes the termination state of a session.

Two or more atomic sessions may be combined with one another by structural connectors to become a compound session $S_{com}$. The combination follows the following syntax and semantics:

$$\begin{aligned} S_{com} ::= & \ S_1 \parallel S_2 && (S_{parallel}) \\ & \mid S_1 \cdot S_2 && (S_{sequence}) \\ & \mid [p_1]\, S_1 + [p_2]\, S_2 && (S_{choice}) \\ & \mid \left[p_g\right]\left[p_{rep}\right] * S && (S_{workunit}) \end{aligned} \tag{6}$$

$S_{parallel}$ describes two parallel executed sessions; $S_{sequence}$ describes two sequentially executed sessions where $S_1$ is the predecessor and $S_2$ is the successor; $S_{choice}$ describes two selectively executed sessions that either $S_1$ or $S_2$ will be executed depending on whether the precondition $p_1$ or $p_2$ is met; $S_{workunit}$ describes a repeatedly executed session, where *S* is triggered by the precondition $p_g$ and it will be repeatedly executed until $p_{rep}$ becomes "false."

*2.1.2. MetaRelation.* *MetaRelation* is a finite set of semantic associations between the metaconcepts appearing in Table 1. It inherits the concepts of semantic verbs, such as *Support*, *Perform*, and *Depend*, from WS-CDL and is supplemented by some new relations such as *SessionDeduction*, *Congruence*, and *Sequence*, to enable comprehensive model checking.

Table 2 provides a set of core metarelations accompanied with their formal semantics specified in DL. The universal constraint $\forall$ is a DL symbol interpreted by $(\forall R \cdot C)^I = \{a \in \Delta^I \mid \forall b((a,b) \in R^I \rightarrow b \in C^I)\}$, where $C$ is a set of metaconcepts, $R$ is a set of metarelations, and $\Delta^I$ denotes nonempty set of discourse domains.

*Definition 4* (*SessionDeduction*). *SessionDeduction* is defined by a labeled transition system: $S \xrightarrow{p,a} S'$, meaning that the session *S* will become $S'$ in the future when the precondition $p$

Table 1: A summary of metaconcepts of service choreography for the metamodel (a fragment).

| Concepts | Definition | Description |
|---|---|---|
| Participant | $Par = \langle N, Ro \rangle$ | The business entities or web services that participate in service interaction. $N$ is the name and $Ro$ denotes the roles it implements. |
| Role | $Ro = \langle N, Op, Ch, Va \rangle$ | The observable behavior a participant exhibits in order to collaborate. $Ro$ initiates the collaborative operations ($Op$) through channels ($Ch$) with other roles, and the local variables ($Va$) will be affected. |
| Activity | $Act = \langle N, Ca \rangle$ | The actual functions performed in the choreography. They ($Ca$) can be categorized into basic activities and structural activities. |
| Precondition | $P = \langle N, \text{Boolean} \rangle$ | The preconditions of session execution, specified in Boolean type. |
| Guard | $G = \langle P, S \rangle$ | The bindings of preconditions to sessions. |
| Variable | $Va = \langle N, Ro \rangle$ | The variables a role has. It is a component of a role specification. |
| Operation | $Op = \langle N, Ro \rangle$ | The operations a role provides. It is a component of a role specification. |
| Channel | $Ch = \langle N, Ro, Loc, Int \rangle$ | The locations and manners through which information is exchanged between roles. $Loc$ and $Int$ denote the channel locations and interaction activities, respectively. |
| Interface | $In = \langle N, Par \rangle$ | The declaration of the participants ($Par$) that collaborate. |
| Port | $Po = \langle N, In \rangle$ | The way through which roles interact. One interface may correspond to one role and multiports. |

Table 2: A summary of metarelations of service choreography for the metamodel (a fragment).

| Relations | Correlative concepts | Formal semantics |
|---|---|---|
| SessionDeduction | Session × Session | Session ⊑ ∀ SessionDeduction. Session |
| Congruence | Session × Session | Session ⊑ ∀ Congruence. Session |
| Implement | Participant × Role | Participant ⊑ ∀ Implement.Role |
| Contain | Choreography × Session | Choreography ⊑ ∀ Contain.Session |
| Link | Channel × Role | Channel ⊑ ∀ Link.Role |
| Belong_to | Role × Participant | Role ⊑ ∀ Belong_to.Participant |
| HasName | Role × Name | Role ⊑ ∀ HasName.Name |
| Provide | Role × operation | Role ⊑ ∀ Provide.operation |
| Executing | Session × $S_{atom}$ | Session ⊑ ∀ Executing. $S_{atom}$ |
| Sequence | Session × Session | Session ⊑ ∀ Sequence. Session |
| Parallel | Session × Session | Session ⊑ ∀ Parallel. Session |
| Choice | Session × Session | Session ⊑ ∀ Choice. Session |
| WorkUnit | Session × Session | Session ⊑ ∀ WorkUnit. Session |

becomes true and the atomic session $a$ of $S$ has been executed. *SessionDeduction* is defined for state reachability reasoning.

*Definition 5* (congruence). If the two sessions $S$ and $S'$ behave exactly in same way, there is a congruence relation between them, marked with the symbol ≡. The conditions of congruence are listed as follows:

$$
\begin{aligned}
&C_1 : [p_1]S_1 + [p_2]S_2 \equiv [p_2]S_2 + [p_1]S_1 \\
&C_2 : ([p_1]S_1 + [p_2]S_2) + [p_3]S_3 \equiv [p_2]S_2 \\
&\quad + ([p_1]S_1 + [p_3]S_3) \\
&C_3 : [p_1]S_1 + [p_2]S_2 \equiv [p_1 \vee p_2]S \\
&C_4 : S_1 \| S_2 \equiv S_2 \| S_1 \quad (S_1 \| S_2) \| S_3 \equiv S_1 \| (S_2 \| S_3) \\
&C_5 : S \| S_{no} \equiv S \quad S \| S_{silent} \equiv S \quad S \| NULL \equiv S \\
&C_6 : S_{no}.\ S \equiv S \quad S_{silent}.\ S \equiv S \quad NULL.\ S \equiv S
\end{aligned}
\tag{7}
$$

Definitions of *MetaConcept* and *MetaRelation* provide the syntax and semantics for modeling service choreography. In the subsequent sections, we will discuss the operational semantics for session evolution and the deductive domain rules for checking the properties of consistency, completeness, and state reachability of a service choreography model.

*2.1.3. DomainRule.* Domain rules are description of constraints for specific domains [20]. *DomainRule* is a set of rules defined upon the metamodel, providing overall constraints that need to be held by all concepts and relations in the service choreography model.

Domain rules of service choreography can be classified into three categories: consistency, completeness, and deductive reasoning. They are not limited to what we give in the following. They may be continuously enriched and improved as applied.

*Definition 6* (consistency). A service choreography model is consistent provided that (1) the application model is built

consistent with the metamodel and (2) there is not any conflict among the concepts of the application model. A typical set of consistency rules are defined below.

(i) $R_{con1}$: if there is a metarelation $R$ that associates the metaconcept $C_i$ with $C_j$ ($C_i \sqsubseteq \forall R \cdot C_j$) in the metamodel and $r$ is instance of $R$ that associates the application concept $c_i$ with $c_j$ in the application model, then $c_i$ must be an instance of $C_i$ and $c_j$ must be an instance of $C_j$.

(ii) $R_{con2}$: no more than one relation of *Sequence*, *Parallel*, *Choice*, or *WorkUnit* is allowed to associate a pair of sessions.

(iii) $R_{con3}$: if two sessions are associated with the relations $SessionDeduction(S_i, S_j)$ and $Congruence(S_i, S_j)$ ($i \neq j$) at the same time, the atomic session $S_{atom}$ associated with $S_i$ by the *Executing* relation must be in type of $S_{no}$, $S_{silent}$, or *NULL*.

(iv) $R_{con4}$: if two sessions are associated with the relation $Executing(S_i, S_j)$ ($i \neq j$), $S_j$ must be an atomic session, that is, $S_j : S_{atom}$.

(v) $R_{con5}$: if two associated application concepts $a$ and $b$ are declared in the application model, there must be a metaconcept $C$ and a metarelation $R$ that satisfy $a : C$ and $(a, b) : R$.

(vi) $R_{con6}$: the two relations $Sequence(S_i, S_j)$ and $Sequence(S_j, S_i)$, $i \neq j$, are allowed to exist at the same time.

The WS-CDL specification includes many clauses stated with the keywords such as MUST/MUSTNOT, SHOULD/SHOULDNOT, and SHALL/SHALLNOT. Those clauses may relate to the consistency and completeness constraints. For example, "*a Choreography MUST contain one or more sessions*," from which we can formally define a rule *Choreography* $\sqsubseteq\geq$ 1 *Contain.Session* in DL. The DL expression $\geq nR \cdot C$, used for constraining a relation, is interpreted by $(\geq nR \cdot C)^I = \{a \in \Delta^I \mid \#\{b \in \Delta^I \mid (a, b) \in R^I \wedge b \in C^I\} \geq n\}$, where $R$ is a metarelation and $C$ is a metaconcept, and $\Delta^I$ denotes nonempty set of discourse domains.

*Definition 7* (completeness). A service choreography model is complete provided that the number of the relations in the application model satisfies the multiplicity constraint of the corresponding metarelation in the metamodel. A typical set of completeness rules are defined below, according to the WS-CDL specification:

(i) $R_{com1}$: *Role* $\sqsubseteq\geq$ 1 *Belong_to.Participant* $\sqcap$ *Role* $\sqsubseteq\leq$ 1 *Belong_to.Participant*.

(ii) $R_{com2}$: *Session* $\sqsubseteq\geq$ 1 *Support.Activity*.

(iii) $R_{com3}$: *Channel* $\sqsubseteq\geq$ 2 *Link.Role* $\sqcap$ *Channel* $\sqsubseteq\leq$ 2 *Link.Role*.

(iv) $R_{com4}$: *Interaction* $\sqsubseteq\geq$ 1 *Depand.Channel*.

(v) $R_{com5}$: *Participant* $\sqsubseteq\geq$ 1 *Implement.Role*.

(vi) $R_{com6}$: *Choreography* $\sqsubseteq\geq$ 1 *Contain.Session*.

(vii) $R_{com7}$: *Interface* $\sqsubseteq\geq$ 1 *Bind.Role* $\sqcap$ *Interface* $\sqsubseteq\leq$ 1 *Bind.Role*.

Deductive reasoning rules relate to operational semantics that describes the evolution of a session. They are used to verify the state reachability of service choreography. Every deductive reasoning rule is a Horn clause in the following form: IF ⟨antecedent⟩ THEN ⟨consequent⟩, where the antecedent is a conjunction of one or more clauses and the consequent is an assertion of facts.

The relation *SessionDeduction* defines deduction of session evolution. If the precondition $p$ holds and the session $S$ executes an atomic session $a$ making $S$ become the session $S'$, then $S$ is deduced to $S'$. In particular, the two sessions can be identified by two states, before and after evolution.

Choreography model checking may concern two kinds of state. One is the session state that defines whether a session can normally end and successfully complete its job. The other is the application state that defines whether an application that is running on one or more sessions can normally end. As a session will never end until all atomic sessions end, the state space of a session is determined by all possible states of the atomic sessions. Similarly, the state space of an application is determined by all possible states of the sessions.

*Definition 8* (state reachability). A state (session state) is reachable provided that there is an evolving session which can be identified by the state and which can be deduced from its ancestor in its initial state.

*Inference 1.* If the termination states of an application described by a choreography model are reachable, that is, the termination sessions (denoted as *NULL*) can be deduced from the initial sessions, then all states of the application are reachable and the choreography models are thereby proven correct.

The basic deductive reasoning rules are listed as follows:

(i)
$$R_{atom} : \frac{}{a \xrightarrow{true,a} NULL} \tag{8}$$

(ii)
$$R_{struct} : \frac{S \equiv S' \; S \xrightarrow{p,a} T \; T \equiv T'}{S' \xrightarrow{p,a} T'} \tag{9}$$

(iii)
$$R_{seq1} : \frac{}{a \cdot S \xrightarrow{true,a} S} \tag{10}$$

(iv)
$$R_{seq2} : \frac{S \xrightarrow{p,a} S'}{S \cdot T \xrightarrow{p,a} S' \cdot T} \tag{11}$$

(v)

$$R_{parallel}: \frac{S \xrightarrow{p,a} S'}{S \,\|T \xrightarrow{p,a} S' \,\|T} \tag{12}$$

(vi)

$$R_{choice}: \frac{S_1 \xrightarrow{p_1,a} S_1'}{[p_1]\,S_1 + [p_2]\,S_2 \xrightarrow{p_1,a} S_1'} \tag{13}$$

(vii)

$$R_{nonblock}: \frac{S \xrightarrow{\neg p_g, NULL} S}{[p_g]\,[p_{rep}] * S \xrightarrow{\neg p_g, NULL} NULL} \tag{14}$$

(viii)

$$R_{norepeat}: \frac{S \xrightarrow{p_g \wedge \neg p_{rep}, a} S'}{[p_g]\,[p_{rep}] * S \xrightarrow{p_g \wedge \neg p_{rep}, a} S'} \tag{15}$$

(ix)

$$R_{repeat}: \frac{S \xrightarrow{p_g \wedge p_{rep}, a} S'}{[p_g]\,[p_{rep}] * S \xrightarrow{p_g \wedge p_{rep}, a} S' \cdot [p_{rep}] * S} \tag{16}$$

$R_{atom}$ is the rule that holds the indivisibility of an atomic session. $R_{struct}$ is the rule that keeps the congruence in session deduction. $R_{seq1}$ and $R_{seq2}$ are the deduction rules for the sequence sessions. $R_{parallel}$ is the deduction rule for parallel session that two sessions are parallel executed and one of them evolves. $R_{choice}$ is the deduction rule for the choice session that if the precondition is true and the atomic session is executed, one of the two sessions will evolve. $R_{nonblock}$ is the deduction rule for the workunit that if the precondition is false and $[p_g][p_{rep}] * S$ works in nonblocking mode, the workunit will be skipped. $R_{norepeat}$ is the deduction rule for the workunit that if $p_g$ is "*true*" and $p_{rep}$ is "*false*" $S$ will become $S'$ and the workunit will become $S'$ after execution of $a$. $R_{repeat}$ is the deduction rule for the workunit that if both $p_g$ and $p_{rep}$ are "*true*" $S$ will become $S'$ and the workunit will become $S' \cdot [p_{rep}] * S$ after execution of $a$, where $[p_{rep}] * S$ means if $p_{rep}$ is "*true*" then $S$ will be iteratively executed.

*2.2. Application Model of Service Choreography.* The metamodel gives a formal definition of the metaconcepts and relations of service choreography and provides fundamental semantics for service choreography description. An application model, an instantiation of the metaconcept model, gives a UML-compliant description and representation of service choreography. To build the model, software engineers would start with analysis of the objectives to be achieved in the choreography and then describe the roles that participants implement, the activities that the roles perform, and the session execution patterns.

*Definition 9* (application model of service choreography). The application model of service choreography formally describes the design of service choreography for a distributed application in a UML-compliant representation within the constraint of the metamodel. It comprises three parts: ⟨*AppConcept*, *AppRelation*, *AppFunction*⟩.

*AppConcept* is a finite set of application concepts which are defined by instantiating the metaconcepts of the metamodel. *AppRelation* is a finite set of application relations which are defined by instantiating the metarelations of the metamodel. *AppFunction* is a set of functions that map *AppConcept* to *MetaConcept* or *AppRelation* to *MetaRelation* in order to trace the types of application concepts and relations in the metamodel. For example, Buyer is an *AppConcept* and Role is the corresponding *MetaConcept*, and thus the function is built as follows: *AppFunction(Buyer) = Role*.

The application model, if built with a universal UML tool, may suffer from the problems of inconsistency, incompleteness, and unreachable states, as mentioned in Section 2.1.3. The next section will discuss the technique of transforming the UML-based model into DL ontology to verify the correctness of the model with the help of a popular reasoner.

## 3. Model Transformation and Verification

UML is a semiformal specification language and it does not by itself support logic inference for model checking. A popular solution is to use OCL (Object Constraint Language), a subset of UML for defining domain constraints in first-order predicate logic for model checking. Unfortunately, it is well known that the full expressiveness of OCL may lead to undecidability of reasoning [21]. As a result, the existing methods have to either limit the UML/OCL constructs or decrease the level of automation or balance between the two.

To solve the problem, we suggest using *SHOIN*(D), the subsystem of DL, to formalize the models. DL is proven powerful in expressibility and decidability for knowledge engineering and allows for making use of some handy reasoning engines, such as Pellet and Racer. But engineers may worry about the fact that it is hard to learn a formal language, hoping that the formal language would be hidden by a software tool.

This section will discuss the algorithms for model transformation, the mechanism of model verification based on *SHOIN*(D), and the way of implementing prototype.

*3.1. Mechanism for Service Choreography Model Verification Based on SHOIN(D).* In the early stage of our research, we obtained some meaningful achievements in DL-based reasoning. Dong et al. [22, 23] checked the C4ISR (Command, Control, Communication, Computer, Intelligence, Surveillance, and Reconnaissance) domain models to guarantee the consistency and completeness through converting the UML models into the Description Logic ontology and making use of inference engine Pellet. He et al. [24] presented a method of UML behavioral model verification based on Description Logic system. He et al. transformed UML behavioral models to OWL DL ontology, and hence model consistency can be

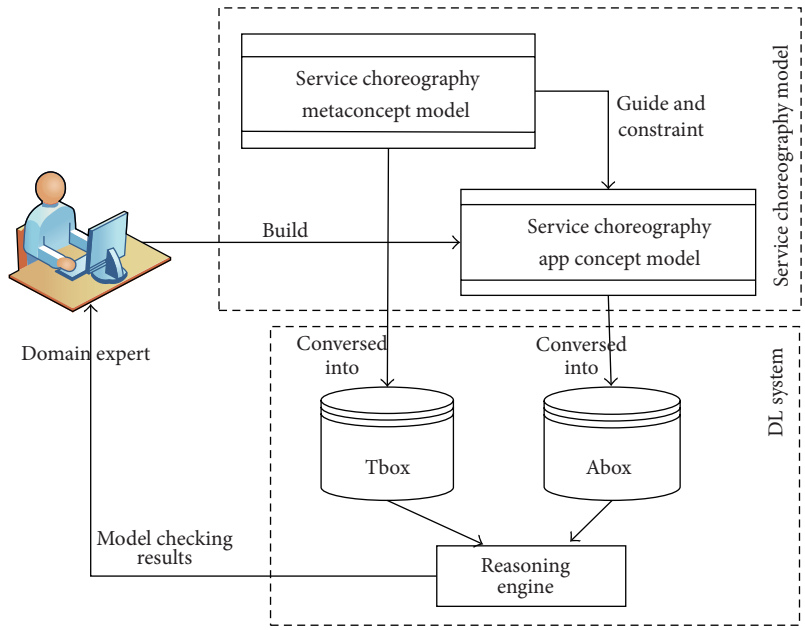



Figure 3: Mechanism for service choreography model verification based on *SHOIN*(D).

verified with DL supporting reasoner. Zhang [25] tried to transform the service-oriented application models to OWL DL ontology. Based on the above research, we present, as shown in Figure 3, a mechanism for service choreography model verification based on *SHOIN*(D).

The principle of conversion and verification is as follows: (1) convert the metaconcepts of service choreography model and domain rules into the axiom sets in *SHOIN*(D) Tbox and the application concept model into the assertion sets in *SHOIN*(D) Abox; (2) verify the consistency, completeness, and state reachability with the help of a reasoning engine like Pellet that supports logical reasoning through the *SHOIN*(D) ontology. The automatic conversion can be realized by Algorithms 1 and 2.

Step (1) initializes Tbox and Abox. Steps (2)–(10) convert the metaconcepts and relations of the metaconcept model into axiom sets in Tbox. Steps (11)–(17) add multiplicity constraints of metarelations as axiom sets in Tbox. Subsequently, steps (18)–(24) convert application concepts and relations of the application model into assertion sets in Abox. For a more detailed explanation for each step, refer to our previous work in [22, 24]. Our previous work [23] showed that the conversion is correct and there is no semantics loss.

The domain rules need to be transformed into axiom sets in Tbox. The consistency and completeness rules are specified in DL and can be added to Tbox through steps (11)–(17), while the deductive reasoning rules need to be specified in SWRL. SWRL combines RuleML and OWL DL. It is popularly applied for formal representation of semantic rules and knowledge-based reasoning, supported by the algorithm Tableau [15].

Algorithm 2 is provided to convert the deductive reasoning rules in the choreography model into the formally specified rules in SWRL.

The concepts and relations appearing in the prerequisite of the deductive reasoning rules are converted to Horn clauses that combine to form the antecedent of SWRL rules, and the concepts and relations appearing in the conclusion are converted to Horn clauses that combine to form the consequent of SWRL rules.

For example, the deductive reasoning rules $R_{atom}$ and $R_{struct}$ are converted into following SWRL expressions:

$$
\begin{aligned}
&\text{(i)}\quad Guard\,(Session\ S, Precondition\ p)\\
&\quad \wedge Excuting\,(Session\ S, S_{atom}\ a)\\
&\quad \wedge Boolean\,(Precondition\ p, true)\\
&\quad \longrightarrow SessionDeduction\,(S_{atom}\ a, NULL)\\
&\text{(ii)}\quad SessionDeduction\,(Session\ S, Session\ T)\\
&\quad \wedge Congruence\,(Session\ S, Session\ S')\\
&\quad \wedge Congruence\,(Session\ T, Session\ T')\\
&\quad \wedge Guard\,(Session\ S, Precondition\ p)\\
&\quad \wedge Executing\,(Session\ S, S_{atom}\ a)\\
&\quad \wedge Boolean\,(Precondition\ p, true)\\
&\quad \longrightarrow SessionDeduction\,(Session\ S', Session\ T')\\
&\quad \wedge Guard\,(Session\ S', Precondition\ p)\\
&\quad \wedge Executing\,(Session\ S', S_{atom}\ a)
\end{aligned}
\tag{17}
$$

**Input**: the meta model and the application model
**Output**: Tbox, Abox
**begin**
(1) Tbox = { }, Abox = { };
(2) **for all** *MetaConcepts* $C$ in meta concept model, **do**
(3) Tbox = Tbox $\cup$ $\{c\}$;
(4) **if** $C_2$ is the superclass of $C_1$, **then**
Tbox = Tbox $\cup$ $\{c_1 \sqsubseteq c_2\}$;
(5) **else if** $C$ has a data attribute $X$ (data type is $t$), **then**
Tbox = Tbox $\cup$ $\{c \sqsubseteq x \cdot t\}$;
(6) **else if** $C_1 \neq C_2$ and $C_1 \sqsubseteq C_2 \notin$ Tbox
and $C_2 \sqsubseteq C_1 \notin$ Tbox, **then**
Tbox = Tbox $\cup$ $\{c_1 \sqcap c_2 = \varnothing\}$;
(7) **end if**;
(8) **end for**;
(9) **for all** *MetaRelation* $R = (C_1, C_2)$ in meta concept model, the relation $r \sim$ is the inverse relation of $r$, **do**
Tbox = Tbox $\cup$ $\{c_1 \sqsubseteq \forall r \cdot c_2,\ c_2 \sqsubseteq \forall r^{\sim} \cdot c_1\}$;
(10) **end for**;
(11) **for all** the multiplicity constraints in range and domain of every relation $R = (C_1, C_2)$ in meta concept model, the relation $r \sim$ is the inverse relation of $r$, **do**
(12) **if** multiplicity constraints is "$0 \cdots 1$" **then**
Tbox = Tbox $\cup$ $\{c_1 \sqsubseteq\leq 1r \cdot c_2\} \cup \{c_2 \sqsubseteq\leq 1r^{\sim} \cdot c_1\}$;
(13) **else if** multiplicity constraints is "$1 \cdots ^{*}$" **then**
Tbox = Tbox $\cup$ $\{c_1 \sqsubseteq\geq 1r \cdot c_2\} \cup \{c_2 \sqsubseteq\geq 1r^{\sim} \cdot c_1\}$;
(14) **else if** multiplicity constraints is "1" **then**
Tbox = Tbox $\cup$ $\{c_1 \sqsubseteq\leq 1r \cdot c_2,\ c_1 \sqsubseteq\geq 1r \cdot c_2\} \cup \{c_2 \sqsubseteq\leq 1r^{\sim} \cdot c_1,\ c_2 \sqsubseteq\geq 1r^{\sim} \cdot c_1\}$;
(15) **end if**;
(16) **end for**;
(17) **return** Tbox;
(18) **for all** individuals $o$ in application concept model,
(19) **if** *AppFunction(o)* = $c$, **do**
Abox = Abox $\cup$ $\{o\} \cup \{o : c\}$;
(20) **end for**;
(21) **for all** individuals $c_1, c_2$ in application concept model,
(22) **if** *AppFunction(r)* = $R$, **do**
Abox = Abox $\cup$ $\{c_1\} \cup \{c_2\} \cup \{\langle c_1, c_2\rangle : r\}$;
(23) **end for**;
(24) **return** Abox;
**end**

Algorithm 1: Construct *SHOIN*(D) Tbox&Abox.

*3.2. Prototype Implementation.* Algorithms 1 and 2 have been realized and integrated into our requirement analysis tool, the so-called ontology-based requirements elicitation and analysis tool (OBREAT) [25]. The architecture of OBREAT and the major components can be found in Figure 4.

The presentation layer handles the interaction between users and the application. It checks and takes user input, as well as feedbacks on execution information including reasoning results. Currently, there are a variety of modeling notations, such as UML [26–31], Message Sequence Charts (MSCs) [32], and BPMN [33, 34], to visually model the choreography interaction. In our research, the UML class diagrams and collaboration diagrams (or Communication Diagrams in [35]) are preferred for the presentation of service choreography models from structural and behavioral viewpoint, respectively, taking the advantages of rigorous syntax and semantics, easiness to interpret, and various tools such as RSA, EA, and Rose to share some part of models.

As the core component of the architecture, the logic layer processes modeling transactions and submits results to the presentation layer, and it also communicates with the data layer for data persistency. The *DL-based Formal Model Generator* handles the model conversion with the algorithms mentioned before. The output of the generator is sent to the *Model Feature Reasoner* and then is saved in *Formal Model* database. The *Model Feature Reasoner* communicates with DL reasoner Pellet to complete DL-based reasoning and sends the results to the *Execution Control Interface*. At present, the *Model Feature Reasoner* realizes checking the property of consistency, completeness, and state reachability. Specifically, the results are also exported to the *Domain Knowledge* base to enrich domain knowledge for reuse and thus knowledge reusability is continuously enhanced with knowledge accumulation.

The data layer handles data persistency and maintenance. Most of the input/output data are saved as files, either model

**Input**: Deductive reasoning rules for service choreography model
**Output**: Deductive reasoning rules based on SWRL, $R_{\text{SWRL}}$
**Begin**
Antecedent = { }, Consequent = { }, Clause = { };
**for all** concepts $C$ and relations $R$ in prerequisite and conclusion,
**if** $C_1$, $C_2$ has $R$ relation, $c_1$, $c_2$ are individuals of $C_1$, $C_2$, **then**
Clause = Clause $\cup$ $\{R(C_1c_1, C_2c_2)\}$;
**else if** $C$ has a attribute $P$ (type is $t$), **then**
Clause = Clause $\cup$ $\{P(Cc, t)\}$;
**end for**;
**for all** clauses cl do
**if** relation between cl is "and" in prerequisite **then**
Antecedent = Antecedent $\cup$ $\{\text{cl}_1 \wedge \text{cl}_2\}$;
**else** if relation between cl is "or" in prerequisite **then**
Antecedent = Antecedent $\cup$ $\{\text{cl}_1 \vee \text{cl}_2\}$;
**else if** relation between cl is "and" in conclusion **then**
Consequent = Consequent $\cup$ $\{\text{cl}_1 \wedge \text{cl}_2\}$;
**else if** relation between cl is "or" in conclusion **then**
Consequent = Consequent $\cup$ $\{\text{cl}_1 \vee \text{cl}_2\}$;
**end if**;
**end for**;
$R_{\text{SWRL}}$ = Antecedent $\cup$ Consequent;
**return** $R_{\text{SWRL}}$;
**end**

Algorithm 2: Construct $R_{\text{SWRL}}$.

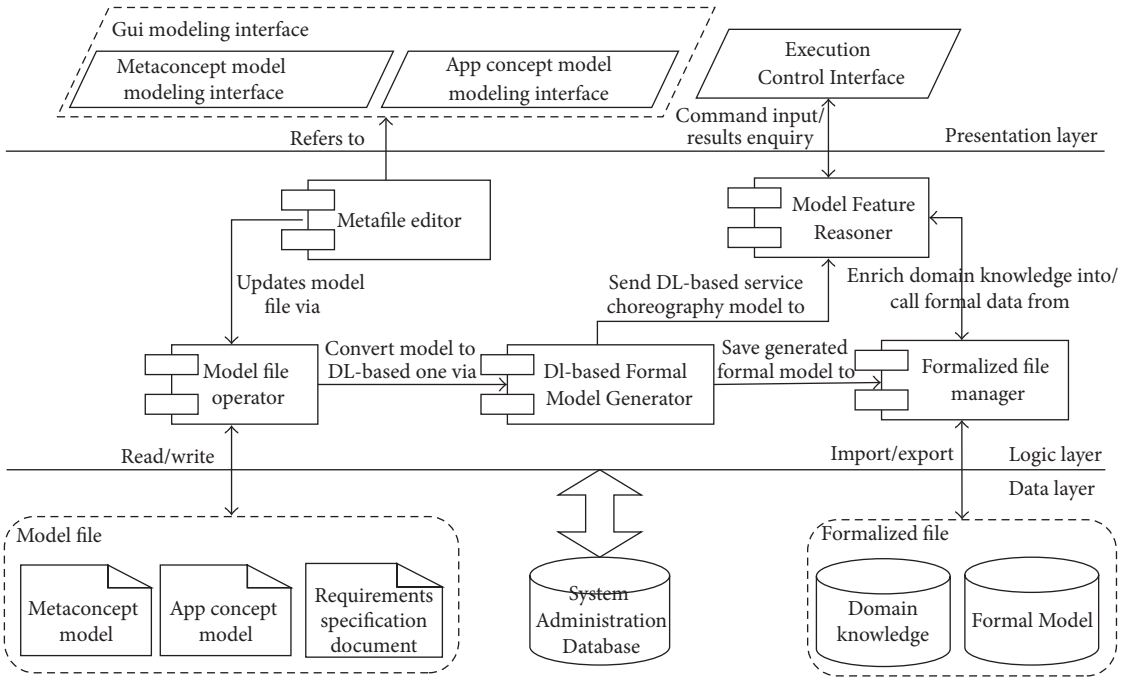



Figure 4: Architecture of service hierarchy of OBREAT.

files or formalized files, while model management data are saved in the database *System Administration Database*.

For the sake of model exchange, we adopt XML as the model data description language. Since the DL-based reasoner accepts data only in OWL which is somewhat different from XML, the conversion between two types of XML documents is needed. To automate the conversion, we choose the eXtensible Stylesheet Language Transformation (XSLT) technology [36, 37] which is widely used for XML document conversion. We designed a set of XSLT transformation templates compliant with the Ontology Definition Metamodel (ODM) and the UML profile. Moreover, we

integrate MagicDraw [38] in OBREAT as the GUI modeling tool in the presentation layer and Pellet 1.5.0 as the DL-based reasoner in the logic layer for verification reasoning.

## 4. Case Study

In this section, a simplified case of purchase order application of the e-commerce system is studied, illustrating how to construct the application model and how to realize the model verification.

*4.1. Construction of Application Model.* Limited by the page size, the choreography of the purchase order application is simplified here, and it covers only a few main activities: the buyer sends an order request to the seller, and then the seller checks the buyer's credit record in the bank and the supplier inventory. If the credit record is good and the inventory is sufficient, then the order will be accepted; otherwise, it will be rejected.

The above activities are encapsulated into five sessions: purchase order request (*S_poReq*), credit check (*S_credChe*), inventory check (*S_invChe*), purchase order response (*S_poResp*), and purchase order reject (*S_poRej*), where there are four participant roles: *Buyer*, *Seller*, *Bank*, and *Supplier*.

The buyer initiates an interaction with the seller by placing a purchase order *po* through the channel *Ch@Seller*, and then the seller acknowledges the buyer by sending him a *poAck*. Meanwhile, the states of the purchase orders of *Buyer* and *Seller* are set to "sent" and "received," respectively. Accordingly, the session *S_poReq* is composed of four atomic sessions which are specified as follows and is modeled as a UML class diagram in Figure 5:

$$\begin{aligned} S\_poReq = {} & S\_poReq\_req\left(Buyer.po \longrightarrow Seller.po, Ch@Seller\right) \\ & \cdot S\_poReq\_resp\left(Buyer.poAck \longleftarrow Seller.poAck, Ch@Buyer\right) \\ & \cdot S\_poReq\_assign\_Buyer\left(Buyer.poState = "sent"\right) \\ & \cdot S\_poReq\_assign\_Seller\left(Seller.poState = "received"\right) \end{aligned} \tag{18}$$

The other four sessions function as follows: *S_credChe* checks the buyer's credit by sending a request to the bank that sends back the state of the credit (good or bad); *S_invChe* checks the seller's inventory by sending a request to the seller who sends back the state of the inventory (sufficient or short); *S_poResp* declares that the order has been accepted if the buyer's credit is good and the inventory is sufficient; and *S_poRej* declares that the order has been rejected if the buyer's credit is bad or the inventory is short. The four sessions are specified as follows, while the related class models are omitted due to space limitation:

$$\begin{aligned} S\_credChe = {} & S\_credChe\_req\text{-}resp(Seller.ccReq \longrightarrow Bank.ccReq, \\ & \qquad Seller.ccResp \longleftarrow Bank.ccResp, Ch@Seller) \\ & \cdot S\_credChe\_assign\_Seller.ccResp\left(Seller.ccResp = "good"/"bad"\right) \\ & \cdot S\_credChe\_assign\_Bank.ccState\left(Bank.ccState = "sent"\right) \\ & \cdot S\_credChe\_assign\_Seller.ccState\left(Seller.ccState = "received"\right) \\ S\_invChe = {} & S\_invChe\_req\text{-}resp(Seller.icReq \longrightarrow Supplier.icReq, \\ & \qquad Seller.icResp \longleftarrow Supplier.icResp, Ch@Seller) \\ & \cdot S\_invChe\_assign\_Seller.icResp\left(Seller.icResp = "sufficient"/"short"\right) \\ & \cdot S\_invChe\_assign\_Supplier.icState\left(Supplier.icState = "sent"\right) \\ & \cdot S\_invChe\_assign\_Seller.icState\left(Seller.icState = "received"\right) \\ S\_poResp = {} & S\_poResp\_resp\left(Buyer.poResp \longleftarrow Seller.poResp, Ch@Buyer\right) \\ & \cdot S\_poResp\_assign\_Buyer.poState\left(Buyer.poState = "completed"\right) \\ & \cdot S\_poResp\_assign\_Seller.poState\left(Seller.poState = "completed"\right) \\ S\_poRej = {} & S\_poRej\_resp\left(Buyer.poResp \longleftarrow Seller.poResp, Ch@Buyer\right) \\ & \cdot S\_poRej\_assign\_Buyer.poState\left(Buyer.poState = "uncompleted"\right) \\ & \cdot S\_poRej\_assign\_Seller.poState\left(Seller.poState = "uncompleted"\right) \end{aligned} \tag{19}$$

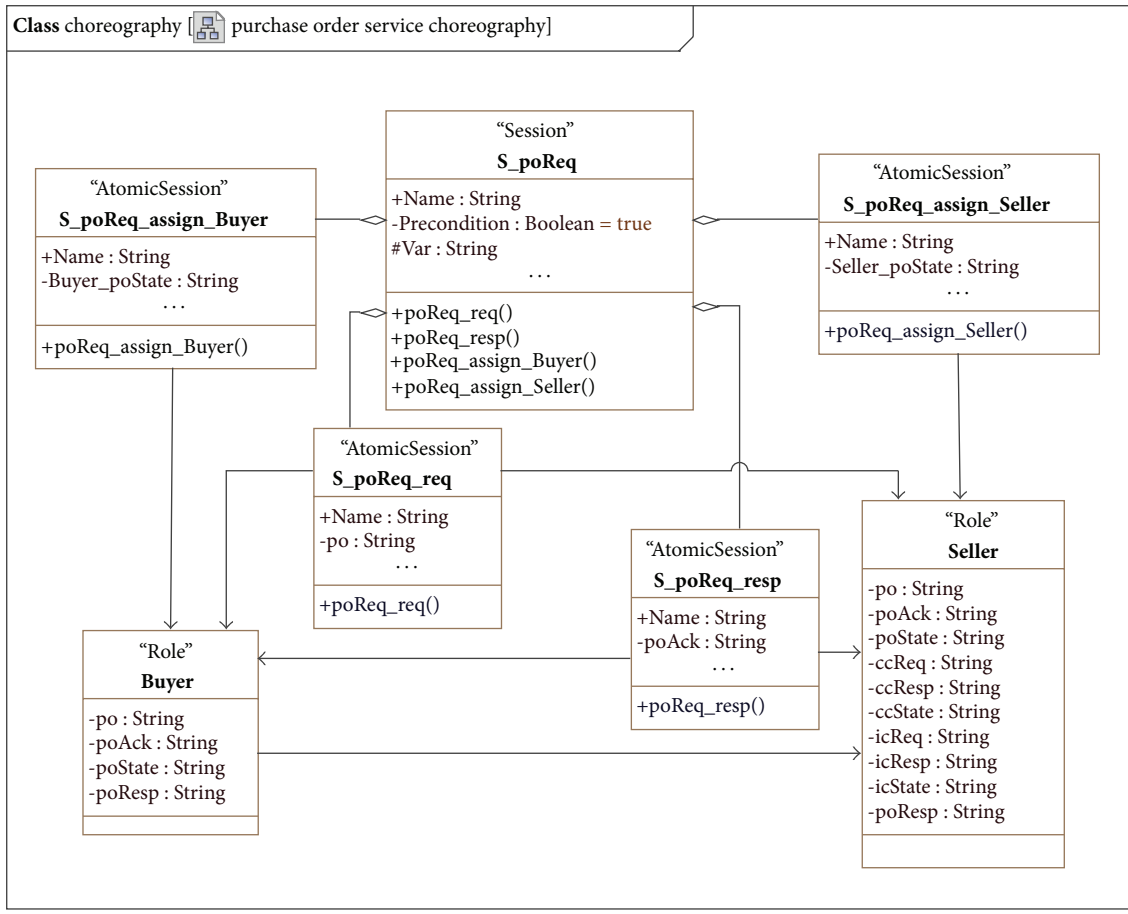



Figure 5: Structure of *S_poReq*.

The interaction sequence companied with the participants and key messages can be modeled as a UML communication diagram, as shown in Figure 6.

Having analyzed the interaction and the functions of the sessions, we can organize the choreography of the purchase order application as follows:

$$Cho\_po = S\_poReq \cdot (S\_credChe \parallel S\_invChe) \cdot (S\_poResp + S\_poRej) \tag{20}$$

and specify the preconditions of the sessions as follows:

$$\begin{aligned} P\_credChe &= p\_invChe \\ &= (Buyer.poState = "sent" \wedge Seller.poState = "received"); \\ P\_poResp &= (Seller.ccResp = "good" \wedge Seller.icResp = "sufficient" \\ &\quad \wedge Seller.ccState = Seller.icState = "received"); \\ P\_poRej &= (Seller.ccResp = "bad" \vee Seller.icResp = "short" \\ &\quad \wedge Seller.ccState = Seller.icState = "received"). \end{aligned} \tag{21}$$

Furthermore, the *SHOIN*(D) ontology can be generated by the algorithm Construct Tbox&Abox, and as a result, the axiom set in Tbox and the instance set in Abox are listed as in Table 3.

*4.2. Verification of Application Model.* Having constructed the application model which is subsequently converted into the *SHOIN*(D) ontology, we can check the correctness properties of consistency, completeness, and state reachability of the specified choreography separately in the following cases.

*Case 1* (consistency checking). Consistency checking is to check whether there is a conceptual conflict against modeling semantics; that is, giving a group of related concepts in the application model, the relationships defined between the concepts should semantically abide by the metarelation declared in the metamodel.

Suppose that the modeler tries to add in the application model a relation of *Executing* between the *Role* Buyer and the *session* S_poReq. But according to the constraints of the *metamodel*, such relation should only appear between two sessions and thus the consistency rule $R_{con1}$ is broken. The error can be easily found through ontology consistency checking using Pellet.

*Case 2* (completeness checking). The completeness checking is to check whether there is a lack of concept or relation in the application model; that is, giving a group of related concepts, the relationships defined between the concepts should quantitatively abide by the multiplicity constraint of the corresponding metarelation in the metamodel.

Suppose that the modeler tries to build a *Belong_to* relation between the *Role* Supplier and the *Participant* ProductTrader, meaning that Supplier belongs to a product trader, but wrongly add at the same time another *Belong_to* relation between Supplier and ServiceProvider. Such relation violates the multiplicity constraint $R_{con1}$ that means each *Role* must belong to only one *Participant* at any time. The checking result is prompted by Pellet as follows: "Consistent: No Reason: The individual Supplier has more than one value for property Belong_to violating the cardinality restriction."

*Case 3* (state reachability checking). The state reachability checking is to examine whether the termination state of a session defined in the application model is reachable; that is, the session can be deduced using the deductive reasoning rules given in Section 2.1.3. If all sessions are deducible, implying that all session states are reachable, the application behavior is verified.

*Session* S_poReq is deduced with the deductive reasoning rules $R_{atom}$ and $R_{seq2}$ and as a result, poState of Buyer will be set to "sent" and the poState of Seller will be set to "received." The reasoning process is shown as follows:

$$
\begin{aligned}
S\_poReq = {} & S\_poReq\_req\left(Buyer.po \longrightarrow Seller.po, Ch@Seller\right) \\
& \cdot S\_poReq\_resp\left(Buyer.poAck \longleftarrow Seller.poAck, Ch@Buyer\right) \\
& \cdot S\_poReq\_assign\_Buyer\left(Buyer.poState = "sent"\right) \\
& \cdot S\_poReq\_assign\_Seller\left(Seller.poState = "received"\right) \\
& \xrightarrow{true,\ a=S\_poReq\_req(Buyer.po \rightarrow Seller.po, Ch@Seller)} \\
& NULL \\
& \cdot S\_poReq\_resp\left(Buyer.poAck \longleftarrow Seller.poAck, Ch@Buyer\right) \\
& \cdot S\_poReq\_assign\_Buyer\left(Buyer.poState = "sent"\right) \\
& \cdot S\_poReq\_assign\_Seller\left(Seller.poState = "received"\right) \\
& \xrightarrow{true,\ a=S\_poReq\_resp(Buyer.poAck \leftarrow Seller.poAck, Ch@Buyer)} \\
& NULL \\
& \cdot S\_poReq\_assign\_Buyer\left(Buyer.poState = "sent"\right) \\
& \cdot S\_poReq\_assign\_Seller\left(Seller.poState = "received"\right) \\
& \xrightarrow{true,\ a=S\_poReq\_assign\_Buyer(Buyer.poState="sent")} \\
& NULL \\
& \cdot S\_poReq\_assign\_Seller\left(Seller.poState = "received"\right) \\
& \xrightarrow{true,\ S\_poReq\_assign\_Seller(Seller.poState="received")} \\
& NULL
\end{aligned}
\tag{22}
$$

The other four sessions credit check (*S_credChe*), inventory check (*S_invChe*), purchase order response (*S_poResp*), and purchase order reject (*S_poRej*) can be deduced in the same way.

The whole reasoning process for the application choreography model *Cho_po* is shown as follows, where the deduction rules applied to each step are bracketed with ⟨ ⟩:

$$
\begin{aligned}
&Cho\_po = S\_poReq \cdot (S\_credChe \parallel S\_invChe) \cdot (S\_poResp + S\_poRej) \\
&\xrightarrow[\langle R_{atom}, R_{seq2}\rangle]{true,\ a^{*}} NULL \cdot (S\_credChe \parallel S\_invChe) \cdot (S\_poResp + S\_poRej) \\
&\xrightarrow[\langle C_6\rangle]{} (S\_credChe \parallel S\_invChe) \cdot (S\_poResp + S\_poRej) \\
&\xrightarrow[\langle R_{atom}, R_{parallel}\rangle]{P\_credChe,\ a^{**}} (NULL \parallel S\_invChe) \cdot (S\_poResp + S\_poRej) \\
&\xrightarrow[\langle C_5\rangle]{} S\_invChe \cdot (S\_poResp + S\_poRej) \\
&\xrightarrow[\langle R_{atom}, R_{seq2}\rangle]{p\_invChe,\ a^{***}} NULL \cdot (S\_poResp + S\_poRej) \\
&\xrightarrow[\langle C_6\rangle]{} (S\_poResp + S\_poRej) \\
&\xrightarrow{P\_poResp,\ a^{****}} NULL
\end{aligned}
\tag{23}
$$

The atomic sessions in each step are as follows:

$$
\begin{aligned}
a^{*} &= S\_poReq\_req \cdot S\_poReq\_resp \\
&\quad \cdot S\_poReq\_assign\_Buyer \\
&\quad \cdot S\_poReq\_assign\_Seller \\
a^{**} &= S\_credChe\_req\text{-}resp \\
&\quad \cdot S\_credChe\_assign\_Seller.ccResp \\
&\quad \cdot S\_credChe\_assign\_Bank.ccState \\
&\quad \cdot S\_credChe\_assign\_Seller.ccState \\
a^{***} &= S\_invChe\_req\text{-}resp \\
&\quad \cdot S\_invChe\_assign\_Seller.icResp \\
&\quad \cdot S\_invChe\_assign\_Supplier.icState \\
&\quad \cdot S\_invChe\_assign\_Seller.icState \\
a^{****} &= S\_poResp\_resp \\
&\quad \cdot S\_poResp\_assign\_Buyer.poState \\
&\quad \cdot S\_poResp\_assign\_Seller.poState
\end{aligned}
\tag{24}
$$

The application behavior is proven correct provided that the reasoning process may eventually end by *NULL*. Otherwise, there must be a session failed to be deduced (i.e., its termination state is not reachable). If the application may reach the final state, the key attribute poState of both Buyer and Seller will be set to either "completed" or "uncompleted" (this may happen when *Byuer.ccResp* = "*bad*" or *Seller.icResp* = "*short*").

To give a negative example, we deliberately assign Seller.poState = "" to make false the precondition *P_credChe*. As a result, the reasoning process rests on the second step, as the session S_credChe cannot be deduced, and therefore all states of the sessions followed cannot be reached. The exception can be found by entering the following DL-based query command in the form of SPARQL [24]:

> SELECT ? a WHERE {? a rdf:type xmlns: Session.? a xmlns: SessionDeduction ? NULL.}.

Figure 7 shows the query result in the human interface panel of the reasoner Pellet, indicating that the session S_poReq and its four atomic sessions are deduced (or can be successfully executed), while all of the subsequent sessions are not shown (or the corresponding states of these sessions may not be reached). The rules $\langle R_{atom}, R_{seq2}\rangle$ are applied to the reasoning process to find the exception where the *SessionDeduction* relation is broken between the sessions S_poReq and S_credChe and all of the followed sessions cannot be deduced.

Beside the above experiment, we have also modeled and verified several other cases, including the online shopping example given in [39], the buyer-seller example given in [40], and the example from the supply chain management [41].

The verification is processed efficiently. The reasoning for each case is finished within a second, tested on a laptop computer with a 2 GHz Intel processor and 1 GB of RAM. That accords with the comments by Haarslev and Möller [42] who pointed out that even for the hardest problems the query time for DL reasoning would be within three seconds. But, is it always true?

In order to evaluate the efficiency of reasoning for CML-DL models, we choose the four popular ontologies, VICODI, LUBM, Semintec, and Wine which are used in previous benchmarks, carry out experiment, and lead a statistical analysis by comparing the average response times against the increasing size of Abox. The detailed descriptions about these ontologies can be found in [43].

Table 3: Axiom set and instances set of purchase order service choreography model (fragment).

| Axiom set in Tbox | Instances set in Abox |
|---|---|
| | $S$_poReq: $S_{com}$ |
| NoAction ⊑ Abas | $S$_poReq_req: $S_{atom}$ |
| SilentAction ⊑ Abas | $S$_poReq_resp: $S_{atom}$ |
| InterAction ⊑ Abas | $S$_poReq_assign_Buyer: $S_{atom}$ |
| Assign ⊑ Abas | $S$_poReq_assign_Seller: $S_{atom}$ |
| Perform ⊑ Abas | $S$_credChe: $S_{com}$ |
| Abas ⊑ Act | $S$_poResp: $S_{com}$ |
| Astr ⊑ Act | ProductTrader: Participant |
| $S_{atom}$ ⊑ Session | ServiceProvider: Participant |
| $S_{com}$ ⊑ Session | $S$_credChe ‖ $S$_invChe $S$_Che: $S_{\text{parallel}}$ |
| Session ⊑ ∀ SessionDeduction.Session | $S$_poResp + $S$_poRej: $S_{\text{choice}}$ |
| Participant ⊑ ∀ Implement.Role | Cho_po: Choreography |
| Choreography ⊑ ∀ Contain.Session | Buyer: Role |
| Interaction ⊑ ∀ Depend.Channel | Supplier: Role |
| Role ⊑ ∀ Belong_to.Participant | $P$_credChe: Precondition |
| Interface ⊑ ∀ Bind.Role | ⟨Cho_po, $S$_poReq⟩: Contain |
| Session ⊑ ∀ Executing. $S_{atom}$ | ⟨$S$_credChe, $P$_credChe⟩: Guard |
| Session ⊑ ∀ Sequence. Session | ⟨$S$_poReq, $S$_poReq_req⟩: Executing |
| Session ⊑ ∀ Choice. Session | ⟨$S$_credChe, $S$_invChe⟩: Parallel |
| ⋮ | ⟨$S$_poReq, $S$_credChe ‖ $S$_invChe $S$_Che⟩: Sequence |
| | ⋮ |

The ontologies represent four standard benchmark datasets with different complexity. The VICODI ontology is relatively small and simple since it does not contain any disjunctions, existential quantification, or number restrictions. The LUBM ontology and the Semintec ontology are also relatively simple like the VICODI ontology, while the Semintec ontology is more complex since it contains functional properties and disjointness constraints and is constructed using OWL DL. The Wine ontology is the most complex one. It contains a classification of wines and is established using advanced DL constructors.

The above ontologies are too small in Abox for our intended performance evaluations. Therefore, we generate different sizes of datasets, ranging from 100 to 1 million, to increase the Aboxes of the ontologies for test.

To evaluate the query times for various sizes of the datasets, we use several query patterns covering all cases from the simplest one that retrieves all individuals of one concept to the most complex one that retrieves all individuals of the concept. The test method can be referenced by [44].

Figure 8 shows the average time for query response for the four ontologies with the generated datasets sizing from 100 to 1 Mio. All the results reported are averaged over 100 queries. The test is led on a laptop computer with one 2 GHz Intel processor and 1 GB of RAM, running Windows XP Service Pack 4.

As expected, there is a significant increase of query time as the size of the ontology increases from the 100 to 1 million. For the example of the Semintec ontology, the average query time for smaller size from 100 to 10,000 is less than a second (<1). But, the time increases sharply as the size exceeds 10,000, and for the maximum size of 1 Mio., the time is about 800 seconds or about 17 minutes.

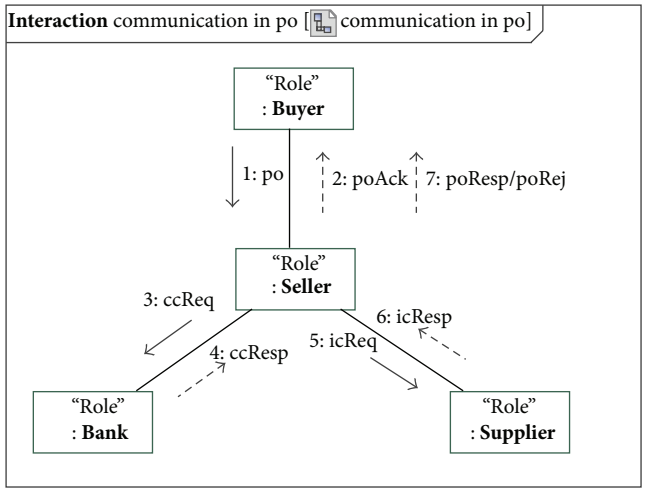



Figure 6: Interaction between roles in the choreography.

Our ontology of CML-DL may match the Semintec ontology since the domain of the CML-DL application is not more complex than that of the financial services, and the model can be constructed using a fragment of OWL DL without advanced DL constructors. Therefore, we expect that the number of individuals of the CML-DL application model, for most projects, would be less than 100000, and thus the reasoning time would be within a few minutes at worst.

## 5. Related Work Discussion

System verification accounts for a larger proportion in the domain of system engineering. There are two categories:

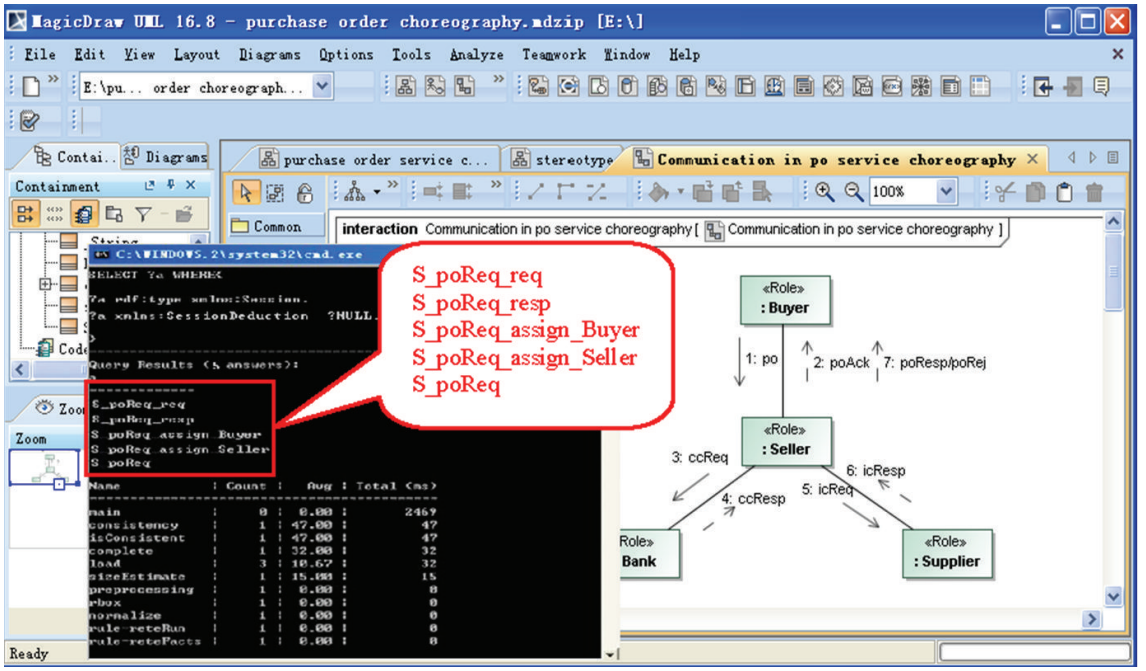
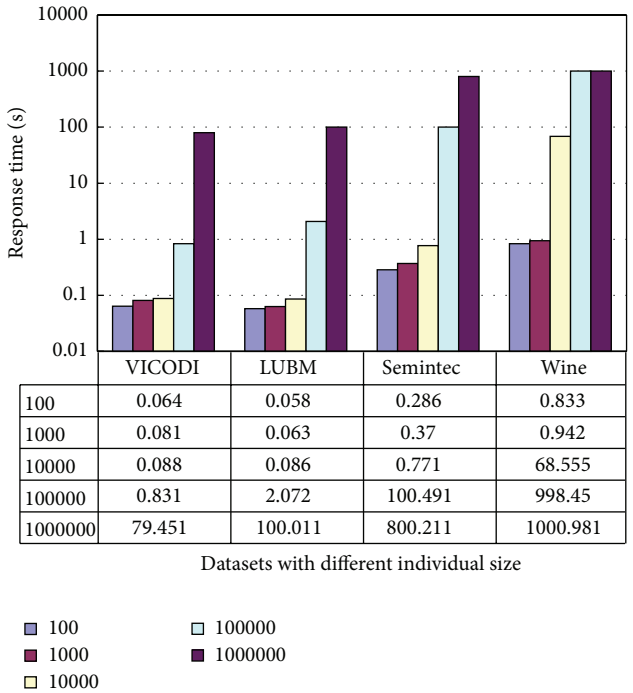



Figure 7: The query command window for checking the state reachability.




| | VICODI | LUBM | Semintec | Wine |
|---|---|---|---|---|
| 100 | 0.064 | 0.058 | 0.286 | 0.833 |
| 1000 | 0.081 | 0.063 | 0.37 | 0.942 |
| 10000 | 0.088 | 0.086 | 0.771 | 68.555 |
| 100000 | 0.831 | 2.072 | 100.491 | 998.45 |
| 1000000 | 79.451 | 100.011 | 800.211 | 1000.981 |

Figure 8: Average query time for the four ontologies (VICODI, LUBM, Semintec, and Wine).

testing and formal verification. Testing is the process in which testers input the use case through access point and then observe the corresponding output to discover potential error. Formal verification, which can be further divided into model checking based on exhausting searching and model verification based on logic reasoning, expresses system specification by using mathematical methods and then proves whether the designed system meets the desired properties according to mathematical theory.

The initial progress of testing static system properties converts static properties into hard-code and calls these codes during program compilation. Because system specification and system implementation are bounded together, the method does not have flexibility and reusability; that is, when the system static properties to be checked change, the hard-code must be updated as well [45].

The emergence of the open-source projects WS-CDL Eclipse Plugin [46] and Pi4SOA [47] makes it possible to separate system specification and implementation. The engines of these projects provide an execution and simulation environment for WS-CDL documents. When the properties to be checked change, the corresponding descriptions can be revised without any modification to the inspection program. Reference [48] notes that although the WS-CDL is just a description language, it is essentially an XML document; therefore, Eclipse Plugin simulators regard the WS-CDL as executable script language for interpretation and implementation. Nevertheless, the verification methods based on the simulation engine need to traverse the entire value space when handling existential and universal quantifiers, leading to a higher complexity of time and space.

Workflow based web service composition and formal verification techniques are broadly applied. They can be divided into three categories: PN- (Petri Net-) based, FSA- (Finite State Automate-) based, and PA- (process algebra-) based techniques.

Xia et al. [49, 50] achieved the complete conversion from a WS-CDL document to stochastic Petri Nets; moreover, they

Table 4: Comparison between several typical verification methods of service choreography model.

| Features | Methods | | | | | |
|---|---|---|---|---|---|---|
| | Hard-coding | Simulation engine | Workflow | Temporal logic | Abstract WS-CDL | CMV-DL |
| Description ability | +++ | +++ | + | + | + | ++ |
| Degree of automation | + | + | ++ | ++ | − | ++ |
| Knowledge reusability | − | − | − | − | − | + |
| Efficiency of verification | + | ++ | +++ | +++ | + | +++ |
| State explosion | − | − | + | + | − | − |

+: support/existence, −: nonsupport/nonexistent.

make assessments of time expectations, probability, and cost expectations of services that normally end due to the index of performance, reliability, and execution cost, respectively. Foster et al. [51] proposed a type of service modeling and verification method called model-based service compositions engineering for the first time. The method maps the service composition model to a finite state process (FSP), aiming at verifying the compatibility between composite services and their environment by using a labeled transition system analyzer (LTSA). Molina-Jimenez and Shrivastava [52] developed the concept of conformance between a contract and a choreography by assuming that they can be modeled by Finite Automaton (FA). The choreography specifications and contracts, specified by the BPMN notation and the event-condition-action rules, describe permissible interactions between partners from different viewpoints. They established a process to automatically check whether all the behaviors permissible in a choreography are also permissible in the corresponding contract and vice versa. Díaz and Llana [53] mapped the service composition model in WS-CDL to a timed automation and then simulated the dynamic behavior of the system using the automatic verification tool UPPAAL. The literature [54] provided the basis for formalizing and reasoning on the mobility characteristics of web services choreography using the process algebra $\pi$-calculus which is, according to Robin Milner, a model of concurrent computation based on the notion of naming. They argued that the process algebras, such as $\pi$-calculus, can be used to formalize web services characteristics to ensure that they satisfy some conditions required in SOA. Salaün et al. [55] presented a method of encoding the collaboration diagrams into the LOTOS process algebra. This encoding allows checking realizability of the collaboration diagrams for both synchronous communication and bounded asynchronous communication.

Gu et al. [8, 9] presented a formal modeling framework Abstract WS-CDL with grammar, congruence relations, and operational semantics. They defined a set of mappings of the Abstract WS-CDL global model to the Pi calculus-based local model and accordingly suggested a set of deductive reasoning rules of state reachability and terminability. Unfortunately, they failed to provide the consistency and completeness verification mechanism as well as the corresponding reasoning engine.

Temporal logic (TL) can be used to assert the behavior change with time evolution. The model verification techniques based on TL model the finite states of systems with Promela and specify the system properties with TL expressions to enable verification by checking whether the intersection between the Promela models and the TL expressions is empty. Zhang and Liu [56] presented a formal verification method for CCML (Cooperative Composition Modeling Language) based web service composition. They build a mapping of CCML description to CCS expression and approach property verification and service compatibility verification with the help of a TL based checking mechanism and an automated tool.

The above verification methods, either based on workflow or based on TL, may lead an exhaustive search for all states of system execution, which may cause the state explosion problem when the number of the states exponentially increases as the system scales up.

The DL-based logic reasoning technique is applied to service modeling and verification. Liu et al. [57] proposed a service modeling and composition method based on DL rules using a uniform way of characterizing the static semantic and dynamic interaction characteristics of web services. The method unifies the service composition model within the framework of DL rules to remedy the defect that DL cannot describe the dynamic characteristics of web services. Chang et al. [58, 59] proposed an Extended Dynamic Description Logic EDDL(X) by extending the traditional Dynamic Description Logic (DDL) to transform the web service composition model into the EDDL(X)-based one. However, the above methods focus more on service discovery, service matching, and service composition while addressing less the property verification of service choreography model.

Over the past decade, many organizations and individuals have conducted in-depth and fruitful research in the field of service choreography model verification. Their achievements can be categorized, according to their underlying methods, into hard-coding, simulation engine, workflow, temporal logic, and Abstract WS-CDL. As shown in Table 4, we have made a primary comparison of our method with those techniques by the important features such as description ability, degree of automation, knowledge reusability, efficiency of verification, and state explosion.

Regarding description ability, the methods of hard-coding and the simulation engine rank highest since they allow compiling the XML scripts of WS-CDL models. CMV-DL ranks second high due to the strong ability of induction and high level of abstraction with DL [60]. The methods based on workflow and TL are weak in description ability, because they cannot ensure the completeness of mapping

one form of model to another, especially when the model is complex in its structure. The current methods based on PA, for example, provide only a few of simple mapping rules and thus do not guarantee a strict conversion, which may cause loss of semantics during conversion [61].

Automation is an important feature for model verification. Each of the methods listed has related supporting tools, except for Abstract WS-CDL. CMV-DL is graded comparatively high in automation, because it takes advantage of available reasoning engines such as Pellet which are based on the mature algorithm Tableau and which have been integrated into the requirement analysis tool OBREAT.

Regarding knowledge reusability, the workflow based, the TL based, and the other three techniques do not provide the mechanism for knowledge accumulation and reuse, while OBREAT allows enriching domain knowledge by adding new reasoning rules to the *Domain Knowledge* base and importing the reasoning results back into the knowledge base for further reasoning. But the process relies on human interference and thus the capability is graded low.

The verification efficiency is determined by both human interference and time consumption in validation process. The hard-code method and CMV-DL depend heavily on human interference and therefore are graded lowest. The simulation engine technique needs to traverse the entire value space when handling existential and universal quantifiers, which leads to a higher complexity and therefore a low efficiency. The other three methods, supported by automated tools, work highly efficiently for verification.

When a system has many concurrent components, the state space of system execution might expand infinitely beyond ordinary computation capability. Based on exhaustive search, workflow and TL may suffer from the problem of state explosion which has long been the bottleneck for their development [62]. In contrast, CMV-DL works on deduction reasoning where the space of deduction reasoning is limited by the size of the generated ontology and the computation complexity is decreased by the high efficient reasoner algorithm Tableau [63, 64].

In general, CMV-DL has an obvious advantage in knowledge reusability and is free of state explosion without the loss of description ability, and it has a modest degree of automation and efficiency compared to the other methods.

## 6. Conclusion and Future Work

This paper focuses on service choreography modeling and verification. It proposes a new approach of choreography model verification based on Description Logic to verify the service choreography model based on *SHOIN*(D) of DL. The main contributions are as follows:

(1) A metaconcept model of service choreography based on WS-CDL is proposed, providing a framework to formally define a service choreography model.

(2) Domain rules of consistency, completeness, and deductive reasoning are defined, enabling formal verification of the service choreography model.

(3) The model transformation algorithms are provided and thus the service choreography model and domain rules can be converted into the DL ontology and model verification can be thereby made automatically with an available reasoner, such as Pellet.

(4) The related work on service choreography verification is investigated. The representative methods are analytically compared with CMV-DL on such features as description ability, degree of automation, knowledge reusability, efficiency of verification, and state explosion.

Currently, CMV-DL focuses only on a few key issues of web service choreography. The goal of the metaconcept model and CMV-DL is to verify several key properties of service choreography models. Therefore, CMV-DL captures only a core set of WS-CDL features while omitting some advanced features, such as exception and finalizing blocks. Future work involves extending the service choreography metaconcept model to cover more properties of WS-CDL, such as the consistency between choreography models and orchestration models. Moreover, the domain rules need to be enriched and improved as the approach is applied to more projects.

## Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

## Acknowledgment

The paper work was supported by National Natural Science Foundation of China under Program no. 61273210.

## References

[1] R. Khadka, B. Sapkota, L. Ferreira Pires, M. Van Sinderen, and S. Jansen, "Model-driven approach to enterprise interoperability at the technical service level," *Computers in Industry*, vol. 64, no. 8, pp. 951–965, 2013.

[2] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: a decade's overview," *Information Sciences*, vol. 280, pp. 218–238, 2014.

[3] N. Kavantzas, D. Burdett, G. Ritzinger et al., Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation, 2005.

[4] F.-X. Xiao, Z.-Q. Huang, Z.-N. Cao, L.-Z. Tu, and Y. Zhu, "Unified formal modeling and analyzing both functionality and qos of web services composition," *Journal of Software*, vol. 22, no. 11, pp. 2698–2715, 2011.

[5] M. E. Cambronero, G. Díaz, V. Valero, and E. Martínez, "Validation and verification of web services choreographies by using timed automata," *Journal of Logic and Algebraic Programming*, vol. 80, no. 1, pp. 25–49, 2011.

[6] L. Zhou, J. Ping, H. Xiao, Z. Wang, G. Pu, and Z. Ding, "Automatically testing web services choreography with assertions," in *Formal Methods and Software Engineering*, vol. 6447 of *Lecture Notes in Computer Science*, pp. 138–154, Springer, Berlin, Germany, 2010.

[7] F. M. Besson, P. M. B. Leal, F. Kon et al., "Towards automated testing of web service choreographies," in *Proceedings of the 6th International Workshop on Automation of Software Testing*, pp. 109–110, 2011.

[8] X. Gu and Z. Lu, "A formal model for BPEL4WS description of Web service composition," *Wuhan University Journal of Natural Sciences*, vol. 11, no. 5, pp. 1311–1319, 2006.

[9] X. Gu, R. Li, and Z. Lu, "Typed formal model for WS-CDL specification of web services composition," *Journal of Southeast University*, vol. 24, no. 3, pp. 300–307, 2008.

[10] S. Hallé and T. Bultan, "Realizability analysis for message-based interactions using shared-state projections," in *Proceedings of the 18th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pp. 27–36, November 2010.

[11] A. McNeile, "Protocol contracts with application to choreographed multiparty collaborations," *Service Oriented Computing & Applications*, vol. 4, no. 2, pp. 109–136, 2010.

[12] S. Basu, T. Bultan, and M. Ouederni, "Deciding choreography realizability," *Acm Sigplan Notices*, vol. 47, no. 1, pp. 191–201, 2012.

[13] W. L. Yeung, "A formal and visual modeling approach to choreography based web services composition and conformance verification," *Expert Systems with Applications*, vol. 38, no. 10, pp. 12772–12785, 2011.

[14] R. J. Brachman and J. G. Schmolze, "An overview of the KL-ONE knowledge representation system," *Cognitive Science*, vol. 9, no. 2, pp. 171–216, 1985.

[15] P. Cimiano, C. Uuger, and J. McCrae, *Ontology-Based Interpretation of Natural Language*, Morgan & Claypool Publishers, 2014.

[16] Web Ontology Language (OWL), World Wide Web Consortium (W3C), 2014, http://www.w3.org/2004/OWL.

[17] J. A. Khan and S. Kumar, "OWL, RDF, RDFS inference derivation using Jena semantic framework & pellet reasoner," in *Proceedings of the International Conference on Advances in Engineering and Technology Research (ICAETR '14)*, pp. 1–8, IEEE, Unnao, India, August 2014.

[18] D. Rodríguez, E. García, S. Sánchez, and C. R.-S. Nuzzi, "Defining software process model constraints with rules using OWL and SWRL," *International Journal of Software Engineering and Knowledge Engineering*, vol. 20, no. 4, pp. 533–548, 2010.

[19] Object Management Group (OMG), *Meta Object Facility (MOF) Core Specification Version 2.0*, 2006, http://www.omg.org/spec/MOF/2.0/PDF.

[20] Q. Dong, Z. Wang, W. Zhu, and H. He, "Capability requirements modeling and verification based on fuzzy ontology," *Journal of Systems Engineering and Electronics*, vol. 23, no. 1, pp. 78–87, 2012.

[21] A. Queralt, G. Rull, E. Teniente, C. Farré, and T. Urpí, "AuRUS: automated reasoning on UML/OCL schemas," in *Conceptual Modeling—ER 2010*, vol. 6412 of *Lecture Notes in Computer Science*, pp. 438–444, Springer, Berlin, Germany, 2010.

[22] Q.-C. Dong, Z.-X. Wang, G.-Y. Chen, J. Xin, and T.-T. Zhang, "Domain-specific modeling and verification for C4ISR capability requirements," *Journal of Central South University of Technology*, vol. 19, no. 5, pp. 1334–1340, 2012.

[23] Q. Dong, *Research on formal analysis framework of effectiveness concept of C4ISR systems [Ph.D. thesis]*, PLA University of Science and Technology, 2013.

[24] H. He, Z. Wang, Q. Dong, W. Zhang, and W. Zhu, "Ontology-based semantic verification for UML behavioral models," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 2, pp. 117–145, 2013.

[25] Y. Zhang, *Research on key techniques for service-oriented C4ISR capability requirements analysis, modeling and design [Ph.D. thesis]*, PLA University of Science and Technology, 2012.

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

[27] J. Su, T. Bultan, X. Fu, and X. Zhao, "Towards a theory of web service choreographies," in *Web Services and Formal Methods: 4th International Workshop, WS-FM 2007, Brisbane, Australia, September 28-29, 2007. Proceedings*, vol. 4937 of *Lecture Notes in Computer Science*, pp. 1–16, Springer, Berlin, Germany, 2008.

[28] T. Bultan and X. Fu, "Choreography modeling and analysis with collaboration diagrams," *Bulletin of the Technical Committee on Data Engineering*, no. 3, 2008.

[29] I. Yahmadi, Y. Baghdadi, and Z. Al-Khanjari, "Graphical description of WS-CDL," in *Proceedings of the 9th International Conference on Innovations in Information Technology (IIT '13)*, vol. 4, pp. 192–197, March 2013.

[30] M. Emilia Cambronero, V. Valero, and E. Martínez, "Design and generation of Web services choreographies with time constraints," *Journal of Universal Computer Science*, vol. 17, no. 13, pp. 1800–1829, 2011.

[31] A. Mellat, N. Nematbakhsh, A. Farahi, and F. Mardukhi, "Suitability of uml state machine for modeling choreography of services," *International Journal of Web & Semantic Technology*, vol. 2, no. 4, pp. 33–53, 2011.

[32] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "An integrated workbench for model-based engineering of service compositions," *IEEE Transactions on Services Computing*, vol. 3, no. 2, pp. 131–144, 2010.

[33] M. V. Rosing, S. White, F. Cummins et al., "Business process model and notation—BPMN," in *Business Process Management Handbook*, pp. 429–453, 2015.

[34] M. Cortes-Cornax, S. Dupuy-Chessa, D. Rieu, and N. Mandran, "Evaluating the appropriateness of the BPMN 2.0 standard for modeling service choreographies: using an extended quality framework," *Software & Systems Modeling*, 2014.

[35] OMG, *OMG Unified Modeling Language (OMG UML), Superstructure, V.2.1.2*, 2007, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF.

[36] D. Li, X. Li, and V. Stolz, "QVT-based model transformation using XSLT," *Acm Sigsoft Software Engineering Notes*, vol. 36, no. 1, pp. 1–8, 2011.

[37] A. Belghiat and M. Bourahla, "Transformation of UML models towards OWL ontologies," in *Proceedings of the 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT '12)*, pp. 840–846, March 2012.

[38] No Magic, MagicDraw Technical Overview, 2015, http://www.nomagic.com/products/magicdraw.html.

[39] J. Su and Y. Sun, "Choreography revisited," in *Web Services and Formal Methods*, vol. 8379 of *Lecture Notes in Computer Science*, pp. 13–25, Springer, 2014.

[40] S. Wieczorek, *Modeling and Model-Based Testing of Service Choreographies*, Akademische Verlagsgemeinschaft München, 2011.

[41] A. Barros, T. Hettel, and C. Flender, "Process choreography modeling," in *Handbook on Business Process Management 1*, International Handbooks on Information Systems, pp. 257–277, Springer, Berlin, Germany, 2010.

[42] V. Haarslev and R. Möller, "Consistency testing: the race experience," in *Automated Reasoning with Analytic Tableaux and Related Methods*, vol. 1847 of *Lecture Notes in Computer Science*, pp. 57–61, Springer, Berlin, Germany, 2000.

[43] B. Motik and U. Sattler, "A comparison of reasoning techniques for querying large description logic Aboxes," in *Logic for Programming, Artificial Intelligence, and Reasoning: 13th International Conference, LPAR 2006, Phnom Penh, Cambodia, November 13–17, 2006. Proceedings*, vol. 4246 of *Lecture Notes in Computer Science*, pp. 227–241, Springer, Berlin, Germany, 2006.

[44] P. Cimiano, P. Haase, Q. Ji et al., "Reasoning with large a-boxes in fuzzy description logics using dl reasoners: an experimental evaluation," in *Proceedings of the ESWC Workshop on Advancing Reasoning on the Web Scalability & Commonsense*, 2008.

[45] Z. Wang, L. Zhou, Y. Zhao et al., "Web services choreography validation," *Service Oriented Computing and Applications*, vol. 4, no. 4, pp. 291–305, 2010.

[46] WS-CDL Eclipse Plugin, 2014, http://wscdl-eclipse.sourceforge.net.

[47] C. Das and P. Bhuyan, "A systematic survey report on various frameworks and models for verification of choreography in SOA," *Journal of Computer Science & Engineering Technology*, vol. 5, no. 4, pp. 411–418, 2014.

[48] F. Besson, *A framework for automated testing of web service choreographies [Ph.D. thesis]*, University of São Paulo, São Paulo, Brazil, 2011.

[49] Y. Xia, H. Xue, and X. Wang, "Performance prediction of WS-CDL based service composition," in *Proceedings of the 10th International Conference on Quality Software (QSIC '10)*, pp. 294–299, July 2010.

[50] Y. Dong, Y. Xia, T. Sun, and Q. Zhu, "Modeling and performance evaluation of service choreography based on stochastic Petri net," *Journal of Computers*, vol. 5, no. 4, pp. 516–523, 2010.

[51] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "An integrated workbench for model-based engineering of service compositions," *IEEE Transactions on Services Computing*, vol. 3, no. 2, pp. 131–144, 2010.

[52] C. Molina-Jimenez and S. Shrivastava, "Establishing conformance between contracts and choreographies," in *Proceedings of the 15th IEEE Conference on Business Informatics (CBI '13)*, pp. 69–78, IEEE, Vienna, Austria, July 2013.

[53] G. Díaz and L. Llana, "Contract compliance monitoring of web services," in *Service-Oriented and Cloud Computing*, vol. 8135 of *Lecture Notes in Computer Science*, pp. 119–133, Springer, Berlin, Germany, 2013.

[54] P. Nduwimfura, D. Xu, H. Miao, Z. Lei, and B. Chen, "Reasoning on formalizing WS-CDL mobility using process algebra," in *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC '10)*, pp. 676–682, December 2010.

[55] G. Salaün, T. Bultan, and N. Roohi, "Realizability of choreographies using process algebra encodings," *IEEE Transactions on Services Computing*, vol. 5, no. 3, pp. 290–304, 2012.

[56] X. Zhang and H. Liu, "Formal verification for CCML based web service composition," *Information Technology Journal*, vol. 10, no. 9, pp. 1692–1700, 2011.

[57] S. Liu, D. Liu, H. Qi, and J. Guan, "Composing semantic web service with description logic rules," *Journal of Computer Research and Development*, vol. 48, no. 5, pp. 831–840, 2011.

[58] L. Chang, Z. Z. Shi, L. M. Chen, and W. Niu, "Family of extended dynamic description logics," *Journal of Software*, vol. 21, no. 1, pp. 1–13, 2010.

[59] L. Chang, Z. Shi, T. Gu, and L. Zhao, "A family of dynamic description logics for representing and reasoning about actions," *Journal of Automated Reasoning*, vol. 49, no. 1, pp. 1–52, 2012.

[60] B. B. Hariri, D. Calvanese, M. Montali, G. de Giacomo, R. De Masellis, and P. Felli, "Description logic knowledge and action bases," *Journal of Artificial Intelligence Research*, vol. 46, pp. 651–686, 2013.

[61] Z. Huang, F. Xiao, and L. Tu, "Modeling service composition using priced probabilistic process algebra," in *Proceedings of the 5th IEEE International Symposium on Service-Oriented System Engineering (SOSE '10)*, pp. 35–38, June 2010.

[62] P. Drobintsev, V. P. Kotlyarov, and I. V. Nikiforov, "Technology aspects of state explosion problem resolving for industrial software design," in *Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering*, pp. 46–51, Kazan, Russia, May 2013.

[63] S. Ben-David, R. Trefler, and G. Weddell, "Model checking using description logic," *Journal of Logic & Computation*, vol. 20, no. 1, pp. 111–131, 2010.

[64] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *Tools for Practical Software Verification*, vol. 7682 of *Lecture Notes in Computer Science*, pp. 1–30, Springer, Berlin, Germany, 2012.