Computers, Materials & Continua CMC, vol.62, no.2, pp.893-904, 2020

A Formal Method for Service Choreography Verification Based on Description Logic

Tingting Zhang[1, 2, 3, *], Yushi Lan[2], Minggang Yu[1], Changyou Zheng[1] and Kun Liu[1]

Abstract: 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. This paper presents a new approach of choreography model verification based on Description Logic. A meta model 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 reasoned. A case study is given to demonstrate applicability of the method. Furthermore, the work will be compared with other related research.

Keywords: Service choreography, WS-CDL, meta-concept model, description logic, formal verification.

1 Introduction

In recent years, the industry and researchers have proposed a new service composition construction based on Choreography. On this basis, the implementation of each part based on service orchestration is automatically generated [Khadka, Bramhananda and Pires (2013); Mendling and Hafner (2009); Cui, Zhang, Cai et al. (2018)]. In order to realize the automatic implementation mechanism of service choreography, the WS-CDL (Web Services Choreography Description Language) [Kavantzas, Burdett and Ritzinger (2015); Sheng (2015)] language is designed. It is a service-oriented description language based on global perspective, which defines a set of services from a global perspective. The rules of collaboration and interaction must be followed. However, WS-CDL, as an XML-based descriptive language, lacks formalized models and verification mechanisms, and it is difficult to ensure the correctness of collaboration and interaction. The correctness of the service orchestration model directly affects all participants involved in the interaction, which has a great impact on the implementation of the later system

[1] PLA Army Engineering University, Nanjing, China.

[2] The 28th Research Institute of China Electronics Technology Group Corporation, Nanjing, China.

[3] Southeast University, Nanjing, China.

[*] Corresponding Author: Tingting Zhang. Email: zhangtings@sohu.com.

[Boehm (1981)]. Web services composition and verification methods based on deductive reasoning [Wang, Li and Li (2008); Madani and Nematbakhsh (2009); Shi and Chang (2008)] use logic formulas to describe systems and their properties, and use some axioms and inference rules to verify that the system has certain properties. The previous research lacking verifications on the consistency, integrity and state accessibility of the service choreography model.

This paper proposes a service choreography model verification method: DLV-CM based on description logic. Firstly, the conceptual model of service choreography is built by extending the WS-CDL normatively, then the consistency, integrity and reasoning deduction rules of the model are proposed. On this basis, the verification problem of the service choreography model is transformed into SHION (D) [Horrocks and Sattler (2007)]. DL [Brachman and Schmolze (1985); Schmidt and Smolka (1991)] reasoning problem, by means of the inference engine such as Pellet to realize the consistency, integrity and state accessibility verification of the service choreography model.

This paper is organized as follows: Section 1 builds a WS-CDL-based service choreography model, and gives the model integrity, consistency, and state reachability reasoning rules; Section 2 introduces the SHOIN (D)-based service choreography model. Then illustrates the conversion algorithm from the service choreography model to SHOIN (D), as well as the conversion algorithm from the reasoning deduction rules to the Semantic Web Rule Language (SWRL) [Parsia (2005); Dong, Wang and Chen (2010)]. And defines the query verification statement based on the Simple Protocol and RDF Query Language, SPARQL according to the type of the conflict problem; Section 3 demonstrates the modeling and verification process of WS-CDL service choreography model based on description logic in a case; The last section summarizes the full text.

2 WS-CDL-based service choreography formal modeling method

This section summarizes the main ideas of service choreography, builds a conceptual model of service choreography based on WS-CDL, and demonstrates the model with consistency, integrity and reasoning deductive rules, all of which provide a formal basis for model verification.

2.1 WS-CDL-based service choreography meta-concept model

Definition 1 Choreography (Cho). It describes the contract between the participants on the cooperation between services, which is the consensus on the global interaction process and constraint rules. Also, it describes the interoperability process between cross-system and cross-organizational services.

The service choreography is represented as a triad: $Cho =< N, Ro, S >$.

Where: N represents the name of the choreography; Ro represents all the roles involved in the choreographer; S is the session included in the choreography.

Definition 2 Session (S). A group of basic interaction activities that perform specific functions is the basic unit that constitutes service choreography.

The session is represented as a four-tuple: $S =< N, Act, Ro, Pr >$.

Where: *N* is the session name; *Act* is all activities that complete the session function; *Ro* is the role contained in the session; and *Pr* is the condition that makes sure the session to be carried out.

The concept in the meta-concept model comes from WS-CDL, and extends the concepts of session, atomic session, compound session, and port based on the purpose of formal modeling and verification of service choreography. Tab. 1 gives a detailed definition and description of each major concept.

Table 1: Service choreography concept

Name	Definition	Description
Participant	Par=<N,Ro>	Describe the entities that participate in service collaboration. A participant can play multiple roles. 'N' is the name (the same below), and 'Ro' is the included role.
Role	Ro=<N,Op,Ch,Va>	Enumerate the roles that participants may play to participate in the interaction. 'Op' provides the action for the character, 'Va' is a local variable, and channel 'Ch' is also associated with the character.
Activity	Act=<N,Ca >	Describe the actions performed in the orchestration. 'Ca' is a classification of activities divided into basic activities and structural activities.
Precondition	P=<N, Boolean>	Describe the preconditions for session execution. The condition is of the Boolean type.
Guard	G=< P, S >	Describe the binding of the guard condition to the session. Contains 'P' and corresponding session 'S'
Variable	Va=<N,Ro>	Describe the variables contained in a role. Contains the variable name 'N' and the role it belongs to.
Operation	Op=< N,Ro>	Describe the actions that the role has. Including the operation name 'N' and the role.
Channel	Ch=<N,Ro,Loc,Int>	Define where and how the interaction takes place. Contains the connected role 'Ro', the channel location 'Loc', and the basic interaction 'Int' that occurs on the channel.
Interface	In=<N,Par>	An interface corresponds to a role and contains multiple ports. 'Par' is the participant to which the interface belongs.
Port	Po=<N,In>	Describe the path of interaction between roles. 'In' is the interface

Among them, activities are divided into basic activities (Abas) and structural activities (Astr). Abas is the basic unit of activity for participating in service interactions, including NoAction, SilentAction, Interaction, Assign and Perform. Astr combines the basic activities it contains in a structured way. Astr is defined as follows:

$$
\begin{aligned}
A_{str} ::= \; & A \| B && (Parallel) \\
| \; & A.B && (Sequence) \\
| \; & [p_1]A + [p_2]B && (Choice) \\
| \; & [p_g][p_{rep}]^* A && (WorkUnit)
\end{aligned}
$$

The structure represents parallel activities, sequential activities, selection activities, and iteration activities, respectively, where the guardian condition p is enforced for the activity. If the true activity p_g is executed in the iterative activity by A, if p_{rep} is true after the execution of the activity A, the activity A will be executed iteratively until it is false.

DL is a family of knowledge-representation-languages with strict formal semantics. It is a decidable subset of the first-order predicate logic. It contains a set of basic concept constructors, such as and ($\sqcup$), implied ($\sqsubseteq$), full-name constraints ($\forall$). Sessions are divided into Atomic Session (S_{atom}) and Compound Session (S_{com}). Atomic Sessions are inseparable sessions, and Composite Sessions are composed of Atomic Sessions. Firstly, we give the definition of the S_{atom}:

$$
\begin{aligned}
S_{atom} ::= \; & Ro(\varnothing) && (S_{no}) \\
| \; & Ro(\tau) && (S_{silent}) \\
| \; & assign(Ro.x = e) && (S_{assign}) \\
| \; & request(Ro_1.x \rightarrow Ro_2.y, Ch@Ro_2) && (S_{req}) \\
| \; & respond(Ro_1.x \leftarrow Ro_2.y, Ch@Ro_1) && (S_{resp}) \\
| \; & req-resp(Ro_1.x \rightarrow Ro_2.y, \; Ro_1.v \leftarrow Ro_2.u, Ch@Ro_2) && (S_{req-resp}) \\
| \; & Cho(x_1, x_2, x_3, ...) && (S_{perform}) \\
| \; & \psi && (NULL)
\end{aligned}
$$

S_{no} is an empty behavior on the character Ro; S_{silent} is a dummy behavior performed on the character Ro that does not have any effect on the current session; S_{assign} is an assignment behavior on the role Ro, the value e is assigned to the variable x; S_{req} is the request interaction behavior between role Ro_1 and Ro_2, Ro_1 that sends request x to Ro_2 through the channel $Ch@Ro_2$, Ro_2 receives the request and saved in x; $S_{req-resp}$ stands for the request-response interaction behavior between role Ro_1 and Ro_2, Ro_1 sending a response x through the channel $Ch@Ro_2$, Ro_2 receives the request and saves it in y; Ro_2 sends the answer u to Ro_1 through the channel $Ch@Ro_1$, Ro_1 receives the request and saves it in v; $S_{perform}$ stands for the call to the session Cho, $x_1, x_2, x_3, ...$ stands for the actual parameter when the call is being executed; $NULL$ stands for a null session, or the termination of the session.

A Compound Session is composed of Atomic Sessions through a connection relationship, and a Composite Session can form a larger Composite Session. The Composite Session is defined as follows:

$$
\begin{aligned}
S_{com} ::= \; & S_1 \| S_2 && (S_{parallel}) \\
& | \, S_1 . S_2 && (S_{sequence}) \\
& | \, [p_1] S_1 + [p_2] S_2 && (S_{choice}) \\
& | \, [p_g][p_{rep}]^* S && (S_{workunit})
\end{aligned}
$$

$S_{parallel}$ represents two concurrently executed sessions; $S_{sequence}$ stands for 2 consecutively executed sessions; S_{choice} indicates the session selected for execution. The session S_1 is executed when the guardian condition is true, and the session S_2 is executed when guardian condition p_2 is true; $S_{workunit}$ indicates the iterative execution session. If p_g is a true session, then S will be executed. If the session S is completed, if it is true, then the session S will be iteratively executed until p_{rep} is false. Inter-session relationship constraints satisfy the following DL expressions:

$S_{no} \sqcup S_{silent} \sqcup S_{assign} \sqcup S_{req} \sqcup S_{resp} \sqcup S_{req-resp} \sqcup S_{perform} \sqcup NULL \sqsubseteq S_{atom}$;

$S_{parallel} \sqcup S_{sequence} \sqcup S_{choice} \sqcup S_{workunit} \sqsubseteq S_{com}$;

$S_{atom} \sqcup S_{com} \sqsubseteq S$.

2.2 Field rule description

A domain rule is a description of constraints in a particular domain that is used to maintain business structure or to control and influence business behavior, providing validation criteria for the model validation process. This paper divides the domain rules of service collaboration into three categories: consistency rules, integrity rules and deductive inference rules. The domain rules are not fixed and will be continuously enriched and refined by domain experts in practical applications.

The consistency of the model includes multiple aspects, such as syntax consistency, semantic consistency, etc. Consistent reasoning based on Tableaux algorithm can slove the problem of consistency verification such as Subsumption, Equivalent, Satisfiability and so on [Wang, Dong and Zhu (2013)]. In this paper, the consistency of the model is defined as two aspects: whether there is behavioral semantic conflict inside the applied conceptual model; whether there exists semantic contradiction between the applied conceptual model and the meta-conceptual model.

Based on the characteristics of service orchestration model integrity analysis, this paper focuses on the following service orchestration model integrity constraint rules based on the WS-CDL statute:

$R_{com1} : Choreography \sqsubseteq \geq 1 \, Contain.Session$.

$R_{com2} : Session \sqsubseteq \geq 1 \, Support.Activity$.

$R_{com3} : Channel \sqsubseteq \geq 2 \, Link.Role \sqcap Channel \sqsubseteq \leq 2 \, Link.Role.$

$R_{com4} : Interaction \sqsubseteq \geq 1 \, Depand.Channel$.

$R_{com5} : Participant \sqsubseteq \geq 1 \, Implement.Role.$

$R_{com6} : Role \sqsubseteq \geq 1 \, Belong\_to.Participant \sqcap Role \sqsubseteq \leq 1 \, Belong\_to.Participant$.

$R_{com7} : Interface \sqsubseteq \geq 1 \, Bind.Role \sqcap Interface \sqsubseteq \leq 1 \, Bind.Role$.

R_{com8} : $Role \sqsubseteq \geq 1 Own.Port$.

R_{com9} : $Role \sqsubseteq \geq 1 Provide.Operation$.

The deductive inference rules defined in this paper are mainly for the state reachability verification of the service choreographer model. Deductive inference rules are given as shown follows: $S \xrightarrow{p,a} S'$

The deductive inference rules between sessions are as follows:

$$R_{atom}: \frac{}{a \xrightarrow{true,a} NULL} \quad R_{struct}: \frac{S \equiv S' \; S \xrightarrow{p,a} T \; T \equiv T'}{S' \xrightarrow{p,a} T'}$$

$$R_{seq1}: \frac{}{a.S \xrightarrow{true,a} S} \qquad R_{seq2}: \frac{S \xrightarrow{p,a} S'}{S.T \xrightarrow{p,a} S'.T}$$

$$R_{parallel}: \frac{S \xrightarrow{p,a} S'}{S \| T \xrightarrow{p,a} S' \| T} \quad R_{choice}: \frac{S_1 \xrightarrow{p_1,a} S_1'}{[p_1]S_1 + [p_2]S_2 \xrightarrow{p_1,a} S_1'}$$

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

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

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

The ‘atom’ rule specifies the indivisibility of the atomic session; the ‘struct’ rule specifies the application of the isomorphism in the session deduction; the ‘seq’ rule refers to the final representation of the atomic session being executed sequentially; the ‘par’ rule refers to the final expression of the parallel execution of the basic session; The ‘choice’ rule means that if the session S_1 is executed by the execution of the basic session a, then the ‘choice’ session $[p_1]S_1 + [p_2]S_2$ can be expressed as S_1'; the ‘no-block’ rule means that if p_g is false, and the $[p_g][p_{rep}]^* S$ work is in the no-blocking mode, then the session executed iteratively will be shipped directly; the ‘no-repeat’ rule means it p_g is true and p_{rep} is false, and the execution of S via the basic session a is performed as S', then the session executed iteratively will be expressed as S' via the execution of a; the ‘repeat’ rule means that if p_g and p_{rep} are true simultaneously, and The execution of S is expressed as S' via the basic session a, then the session executed iteratively is performed as $S'.[p_{rep}]^* S$ via a, where $[p_{rep}]^* S$ means it p_{rep} is true then S will be executed iteratively.

3 Service choreographer model and deductive inference rule conversion algorithm

SWRL combines the sublanguage Datalog of OWL DL and RuleML. It is a semantic rule description language that combines DL and Horn clauses. Not only does it realize the representation of the rule knowledge, but also it maintains efficient reasoning of DL language. Moreover, due to Tableau algorithm supports ontology reasoning of additional

SWRL rules, this paper uses SWRL as the rule language of SHOIN(D) deductive inference rules. The rule of conversion algorithm Construct RSWRL can be constructed as follows:

Algorithm Construct RSWRL

Input: Deductive reasoning rules of service choreography model

Output: Deductive reasoning rules based on SWRL, R_{SWRL}

begin

Antecedent = {}, Consequent = {}, Clause= {};

for all concepts C and relations R in prerequisite and conclution,

if C1, C2 has R relation, c1, c2 are individuals of C1, C2, **then**

Clause = Clause $\cup$ { $R(C_1 c_1, C_2 c_2)$ };

else if C has a attribute P (type is t), **then**

Clause = Clause $\cup$ { $P(C c, t)$ };

end for;

for all clauses cl do

if relation between cl is "and" in prerequisite **then**

Antecedent = Antecedent $\cup$ { $cl_1 \wedge cl_2$ };

else if relation between cl is "or" in prerequisite **then**

Antecedent = Antecedent $\cup$ { $cl_1 \vee cl_2$ };

else if relation between cl is "and" in conclution **then**

Consequent = Consequent $\cup$ { $cl_1 \wedge cl_2$ };

else if relation between cl is "or" in conclution **then**

Consequent = Consequent $\cup$ { $cl_1 \vee cl_2$ };

end if;

end for;

RSWRL= Antecedent $\cup$ Consequent;

return R_{SWRL};

end

The concepts, concept attributes, and concepts in the premise and conclusions are all converted into the pre-questions and results of the Horn clause. The pre-requisites and results of the Horn clause are combined into the conditions and inferences of the SWRL by logical connectors. For example, for the rule R_{atom}, the converted SWRL expression is:

$$Guard(Session\ S, Precondition\ p)$$
$$\wedge Excuting(Session\ S, S_{atom}\ a)$$
$$\wedge Boolean(Precondition\ p, true)$$
$$\rightarrow SessionDeduction(S_{atom}\ a, NULL)$$

4 Case analyses

This section takes the shopping order choreographer model in e-commerce as an example. Under the guidance of the meta-concept model, the application conceptual model is constructed. Based on SHOIN(D), the consistency, integrity and state reachability of the model are verified by using Pellet.

4.1 The construction of service choreographer application concept model

The customer sends a purchase order request to the seller. After receiving the request, the seller will review the customer's credit record with the bank and check the supplier' inventory at the same time. If the credit is good and the inventory is sufficient, the order will be accepted, otherwise the order will be rejected.

The service director involves four roles: *Buyer, Seller, Bank* and *Supplier*. According to Definition 2, the interaction activity between roles can be divided into five sessions by function: $S_{poRequest}$, $S_{creditCheck}$, $S_{invCheck}$, $S_{poResponse}$, $S_{poReject}$.

Firstly, the *Buyer* sends an order request *'po'* to *Seller* via the channel *Ch@Seller*. After receiving the request, *Seller* sends a confirmation message *'poAck'* to the *Buyer* via the channel *Ch@Buyer*. The *Buye*r sets its order status variable to *"sent"* and the *Seller* will place his order status, the variable of which is set to *"received"*.

$$\begin{aligned} S_{poRequest} = & S_{req}(Buyer.po \rightarrow Seller.po, Ch@Seller). \\ & S_{resp}(Buyer.poAck \leftarrow Seller.poAck, Ch@Buyer). \\ & S_{assign}(Buyer.poState = "sent"). \\ & S_{assign}(Seller.poState = "received"). \end{aligned}$$

The session $S_{creditCheck}$ completes the credit inquiry function, *Seller* sends a credit inquiry request *ccReq* to the *Bank*, and the *Bank* gives a reply *ccResp="good"/"bad"* to the request. Finally, the *Bank* and the *Seller* set the relevant state variables to *"sent"* and *"received"* respectively.

$$\begin{aligned} S_{creditCheck} = & S_{req-resp}(Seller.ccReq \rightarrow Bank.ccReq,\ Seller.ccResp \leftarrow Bank.ccResp, Ch@Seller). \\ & S_{assign}(Seller.ccResp = "good"/"bad"). \\ & S_{assign}(Bank.ccState = "sent"). \\ & S_{assign}(Seller.ccState = "received"). \end{aligned}$$

The session $S_{invCheck}$ completes the inventory query function, the *Seller* sends an inventory inquiry request to the *Supplier*, and the *supplier* returns the inventory status *icResp="sufficient"/"short"* to *Seller*, and assigns values to the relevant state variables respectively.

$$\begin{aligned} S_{invCheck} = & S_{req-resp}(Seller.icReq \rightarrow Supplier.icReq,\ Seller.icResp \leftarrow Supplier.icResp, Ch@Seller). \\ & S_{assign}(Seller.icResp = "sufficient"/"short"). \\ & S_{assign}(Supplier.icState = "sent"). \\ & S_{assign}(Seller.icState = "received"). \end{aligned}$$

If the customer has good credit and sufficient inventory, the *Seller* will respond to the *Buyer* to accept the order, and the status of the order is set to *"completed"*.

$$S_{poResponse} = S_{resp}(Buyer.poResp \leftarrow Seller.poResp, Ch@Buyer).$$
$$S_{assign}(Buyer.poState = "completed").$$
$$S_{assign}(Seller.poState = "completed").$$

If any of the customer's credit history and inventory status cannot be met, then the *Seller* will respond to the *Buyer* to reply a rejection of the order, and the status of the order is set to *"uncompleted"*.

$$S_{poReject} = S_{resp}(Buyer.poResp \leftarrow Seller.poResp, Ch@Buyer).$$
$$S_{assign}(Buyer.poState = "uncompleted").$$
$$S_{assign}(Seller.poState = "uncompleted").$$

Based on the business process, the service choreography combining all sort of sessions are built as follows:

$$Cho = S_{poRequest}.(S_{creditCheck} \| S_{invCheck}).(S_{poResponse} + S_{poReject})$$

The guardian conditions for each session are:

$$p_{creditCheck} = p_{invCheck}$$
$$= (Buyer.poState = "sent" \wedge Seller.poState = "received");$$
$$p_{poResponse} = (Seller.ccResp = "good" \wedge Seller.icResp = "sufficient" \wedge$$
$$Seller.ccState = Seller.icState = "received");$$
$$p_{poReject} = (Seller.ccResp = "bad" \vee Seller.icResp = "short" \wedge$$
$$Seller.ccState = Seller.icState = "received").$$

4.2 Model reasoning and verification

In the case of consistency and integrity verification of the service choreography model, taking R_{con1}, R_{com1} as an example, enter the SPARQL query command under the Pellet console:

SELECT ? a ?b WHERE{? a xmlns: Sequence? b. ? b xmlns: Sequence? a.} and

SELECT ? a WHERE{? a rdf:type xmlns: Choreography. UNSAID{? a xmlns: Contain ? b.}.}

The results show that there is no case of violation of R_{con1}, R_{com1} (Due to space limitations, the corresponding reasoning interface is no longer given here). Both types of rules are verified one by one until all possible inconsistencies and incompleteness are excluded.

In the service accessibility model state reachability verification, the reasoning process of Cho is first given (where the rule used in the step inference is given in < >, in which a^* represents several consecutive basic sessions).

Step1:

$$Cho = S_{poQuest}.(S_{creditCheck} \| S_{invCheck}).(S_{poResponse} + S_{poReject})$$

$$\xrightarrow{true, a^*} NULL.(S_{creditCheck} \| S_{invCheck}).(S_{poResponse} + S_{poReject}) < R_{atom}、 R_{seq2} >$$

$$\xrightarrow{true, a^*} (S_{creditCheck} \| S_{invCheck}).(S_{poResponse} + S_{poReject}) < C_6 >$$

$$a^* = req - resp(Buyer.po \rightarrow Seller.po,\ Buyer.poAck \leftarrow Seller.poAck, Ch@Buyer).$$
$$assign(Buyer.poState = "sent").$$
$$assign(Seller.poState = "received").$$

Step2:

$$(S_{creditCheck} \| S_{invCheck}).(S_{poResponse} + S_{poReject})$$

$$\xrightarrow{P_{creditCheck}, a^*} (NULL \| S_{invCheck}).(S_{poResponse} + S_{poReject}) < R_{atom}、 R_{parallel} >$$

$$\xrightarrow{P_{creditCheck}, a^*} S_{invCheck}.(S_{poResponse} + S_{poReject}) < C_5 >$$

$$a^* = req - resp(Seller.ccReq \rightarrow Bank.ccReq,\ Seller.ccResp \leftarrow Bank.ccResp, Ch@Seller).$$
$$assign(Bank.ccState = "sent").$$
$$assign(Seller.ccState = "received").$$

Step...

$$Cho = S_{poRequest}.(S_{creditCheck} \| S_{invCheck}).(S_{poResponse} + S_{poReject})$$

$$\longrightarrow ... \longrightarrow NULL$$

If the guardian condition of the session is met and the basic session of each session is successfully executed, then the state represented by the session is reachable. If each step of the *Cho* is successfully executed, the final state is terminated after a number of intermediate states.

The reasoning results are shown in Fig. 1

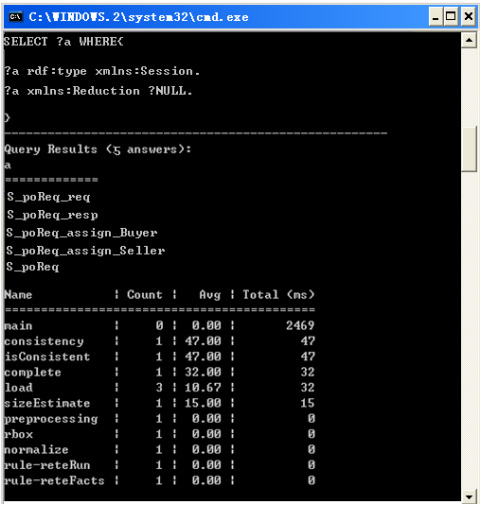

Figure 1: Pellet engine-based modeling model reachability verification results

5 Conclusions and future works to do

This paper builds a conceptual model of service choreography based on WS-CDL specification, and proposes the consistency, integrity and state reachability deductive inference rules. On this basis, the method of transforming the verification problem of service director model into DL inference problem based on SHION(D) is proposed. The automatic inference engine is used to realize data consistency, integrity verification and rule-based logical reasoning. The main innovations of this paper are shown below:

(1) The extended WS-CDL builds a WS-CDL-based service choreography meta-concept model, and gives related concepts and relationships between concepts.

(2) The consistency, integrity and state reachability verification methods for the service choreography model are proposed and the corresponding verification rules are given.

(3) The service choreography model verification method DLV-CM based on SHOIN(D) is proposed. The DLV-CM has the description ability, automation degree and verification efficiency of the traditional system verification method, and supports the decidability of reasoning and the reuse of knowledge.

Compared with the complete semantic description and verification of WS-CDL, DLV-CM only focuses on some key issues of service coding. The service director meta-concept model and the DLV-CM verification method attempt to model and verify the consistency, integrity and state reachability of the service director model as simple as possible on the basis of absorbing the core idea of WS-CDL, and therefore ignore Some of the more advanced concepts and mechanisms in WS-CDL, such as exception handling and finalize blocks. Extending the service choreography conceptual model and incorporating more concepts of WS-CDL will be the next step of our working group. At the same time, we are also actively exploring the possibility of verifying more model attributes under the DLV-CM framework.

Acknowledgement: This work is supported by the National Natural Science Fund number 61802428.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

Boehm, B. (1981): *Software Engineering Economics*. Prentice-Hall, USA.

Brachman R. J.; Schmolze, J. G. (1985): An overview of the KL-ONE knowledge representation system. *Cognitive Science*, vol. 9, no. 2, pp. 171-216.

Cui, J. H.; Zhang, Y. Y.; Cai, Z. P.; Liu, A. F.; Li, Y. Y. (2018): Securing display path for security-sensitive applications on mobile devices, *Computers, Materials & Continua*, vol. 55, no. 1, pp. 17-35.

Dong, Q. C.; Wang, Z. X.; Chen, J. (2010): Method of checking capability model based on description logic. *Systems Engineering and Electronics*, vol. 32, no. 2, pp. 533-539.

Horrocks, I.; Sattler, U. (2007**)**: A tableau decision procedure for SHOIQ. *Journal of Autimated Reasoning*, vol. 39, no. 3, pp. 249-276.

Kavantzas, N.; Burdett, D.; Ritzinger, G.; Fletcher, T.; Lafon, Y. et al. (2005): Web services choreography description language version 1.0. http://www.w3.org/TR/2005/CR-ws-cdl-10-20051109/.

Khadka, R.; Sapkota, B.; Pires, L. F. (2013): Model-driven approach to enterprise interoperability at the technical service level. *Computers in Industry*, vol. 64, no. 5, pp. 951-965.

Mendling, J.; Hafner, M. (2009): From WS-CDL choreography to BPEL process orchestration. *Journal of Enterprise Information Management*, vol. 21, no. 5, pp. 525-542.

Madani, Z.; Nematbakhsh, N. (2009): A logical formal model for verification of web service choreography. *Proceedings of 12th International Conference on Computer and Information Technology*, pp. 448-453.

Parsia, B. (2005): Cautiously approaching SWRL.

https://pdfs.semanticscholar.org/aebd/132b9b7707aa9560744294af1c7148007624.pdf.

Sheng, Q. Z. (2015): Web services composition: a decade's overview. *Information Sciences*, vol. 280, no. 5, pp. 218-238.

Shi, Z. Z.; Chang, L. (2008): Reasoning about semantic web service with an approach based on dynamic description logics. *Chinese Journal of Computers*, vol. 31, no. 9, pp. 1599-1611.

Schmidt, S. M.; Smolka, G. (1991**)**: Attributive concept descriptions with complements. *Artificial Intelligence*, vol. 48, no. 1, pp. 1-26.

Wang, J. H.; Li, Z. J.; Li, M. J. (2008): Composing semantic web services with description logics. *Chinese Journal of Computers*, vol. 19, no. 4, pp. 967-980.

Wang, Z. X.; Dong, Q. C.; Zhu, W. X. (2013): An approach for conceptual analysis on capability requirements consistency and reasonability. *Chinese Journal of Computers*, vol. 36, no. 1, pp. 10-21.