International Journal of Engineering &Technology, 7 (1.7) (2018) 5-10

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Review paper

An approach for verifying correctness of web service compositions

C. Saranya Jothi [1] *, Ravikumar S [1], Antony Kumar K [1], A. Suresh [2]

[1] *Assistant Professor, Department of Computer Science and Engineering, School of Computing, Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology, Avadi, Chennai-62, TamilNadu, India*

[2] *Professor & Head, Department of Computer Science and Engineering, Nehru Institute of Engineering and Technology, T. M. Palayam, Coimbatore-641105, TamilNadu, India*

**Corresponding author E-mail: saranyajothi22@gmail.com*

Abstract

Web services are utilized to illuminate some particular assignment. When a single web service cannot solve a given task, several web services are composed. Composition can be done either statically at design time or dynamically at runtime. Dynamic composition is more suitable for business applications where in business policies and user requirements frequently changes. Interleaved dynamic composition and execution of services is beneficial for adapting to changing user preferences. One of the main issues in such a scenario is that whether the component services that are composed operate according to the business rules specified. Safety, liveness and deadlock freedom properties of a composition depend on the behavior of individual services. Existing modeling techniques capture these properties and perform model checking only statically. Hence in this work, a two level model verification approach has been proposed to verify the correctness of dynamically composed services.

Keywords: *Web Service Composition; Service Oriented; CTL; LTL; Verifying Correctness.*

1. Introduction

Administration Oriented Architecture is a structural style for building venture arrangements in light of administrations. Many companies and individuals rely on the service oriented architecture for their business activity. Consumers depend on the service producers to fulfill their requirements. Their requirement may not be satisfied in a single service. To fulfill their requirement, dynamic composition of services is adopted in service oriented architecture. The correctness of the composition is verified using temporal properties like safety, liveness and deadlock freedom.

1.1. Web services

Web Services are well-defined business functionalities used to solve a specific task, executed in heterogeneous environment that can be reused for different purposes. Web administrations are programming frameworks intended to help interoperable machine-to-machine connection over a system. Web administrations enable organizations to uncover their business usefulness through the Internet which can then be accessed by anyone wishing to use the business function.

1.2. Web services composition

Web benefit arrangement is a procedure of joining the functionalities of a few web benefits with a specific end goal to play out an intricate assignment. Piece of web administrations should be possible in a static or dynamic way. In static organization, the solid work process is worked amid configuration time. Static arrangement can be performed just if the accomplices associated with structure are foreordained and are not prone to change amid execution [13]. In unique piece, the administrations are found at runtime and in view of the client prerequisites the service selection, binding, invocation are performed for composition at runtime.

1.3. Verification of service composition

Web services are passive component which react only upon request. Web services deployed on the web should be sound and complete. The correctness of a single web service and compositions are verified by modeling the web service [14]. Commonly used formal models for web service modeling includes petrinet model, finite state automata and UML diagrams.

1.3.1. Modeling the behavior of web service

The web services and their compositions are modeled as kripke structure. A Kripke structure is a sort of nondeterministic limited state machine proposed by Saul Kripke in 1963, utilized as a part of model checking to speak to the conduct of a framework [16]. It is essentially a diagram whose hubs speak to the reachable conditions of the framework and whose edges speak to state advances

1.3.2. Modeling the specifications

The determinations are displayed as fleeting rationales. Worldly rationales are utilized to predicate over the conduct characterized by Kripke structures. Two profitable transient methods of thinking are Computation Tree Logic (CTL) and Linear Temporal Logic (LTL). Direct Temporal Logic Temporal rationale expands the propositional rationale with an arrangement of worldly administrators.

1) Ff: in some cases in the Future, f is valid in some future minute.
2) Gf: All inclusive later on, f is valid in all future minute.
3) Xf: neXt time, f is valid in the following minute in time.

Computation Tree Logic
CTL allows just fanning time administrators, which implies each of the
Linear time administrators G, F, X, and U must be rapidly gone before by a way quantifier. For example: AG (EFp).
CTL Path quantifier
There are two way quantifiers An, E. A: for All ways. E: there Exists a Way

1.3.3. Emblematic model checking

Emblematic model checking is a programmed, show based, property confirmation approach. Given a model of a framework, test naturally whether this model meets the given particular. Commonly, the framework can be an equipment or programming framework, and the particular contains wellbeing prerequisites, for example, the nonappearance of stops and comparative basic expresses that can make the framework crash. So as to tackle such an issue algorithmically, both the model of the framework and the determination are defined in some exact mathematical language. Model checker is a tool used for performing model checking, which has the following functions:

1) Given the Kripke structure model M
2) Given the properties f
3) Decides whether M j = f
4) Returns yes or true, if the properties are satisfied
5) Otherwise returns no or false, and provides a counter example.

2. Literature survey

Existing modeling techniques and their approach for verifying the correctness for single web service and statically composed services have been discussed by a few authors. Model checking is a programmed, display based, property confirmation approach. Given a model of a system, test automatically whether this model meets a given specification.

2.1. Auto modeling of composite web services

Manual endeavors are required to display the conduct of an administration, which regularly require casual documentation from benefit merchants. Syed Adeel Ali proposed (Warren et al.) a solution for the above problem in which service behavior are automatically extracted from its WSDL document [6]. A Web Service Description Language is a document which characterizes what an administration gives, without uncovering how it is actualized. It doesn't characterize any kind of sequencing imperatives among the operations. The WSDL document was converted to Finite State Machine by using an algorithm. Behavior models could be generated only for top-down pattern, where WSDL was made first and the relating code for web benefit usage was produced later. The drawback of this approach is model could be generated only for statically composed services.

2.2. Web benefit composition

Kil, H. Nam et al. (Nam et al.) proposed a calculation for tackling the web benefit creation issue [7]. The web benefit structure issue was characterized as to discover a co-ordinator web benefit which controls the segment web administrations required to fulfill the objective indicated. Approach: Mark Preserving Abstraction or Signature Subsuming Abstraction was utilized to decrease the web administration to digest one. A co-ordinator web service was identified which was used to control the abstract web services [15]. If the co-ordinator web service could not be identified then web services were refined. Drawback: The correctness of composition was ensured only for statically composed services. Web services were reduced only to variables, not to models.

2.3. Verifying correctness of web service choreograph

Melliti Tarek et al. (Tarek et al.) proposed a technique for checking accuracy in view of a solitary accomplices discernible conduct [9]. A movement was right if each accomplice in the movement was good with its accomplices. Choreography could be deadlock due to incompatibility. Hence to find choreography's correctness web services were modeled as Timed Input Output Transition System. An algorithm was designed to check the similarity between two TIOTS. This strategy was additionally utilized for movement repair, when one accomplice fizzles, another partner with same behavior will replace it. Drawback: This method did not address the whether the composed web services conforms to the expected behavior.

2.4. Analyzing behavioral compatibility using petri net

Xitong Li et al. proposed a strategy to confirm the behavioral similarity of web administrations [8]. Behavioral similarity was checked among the part web administrations required to guarantee the correctness of the whole composition. Different formalism reflect different ways of addressing the compatibility. Here, petri net method was used for verifying the behavioral compatibility. In petri net it should be ensured that no internal or external messages should be buffered i.e after the execution of web service it should not contain any unprocessed internal messages. The approach used here verify the behavioral similarity of web benefits by checking appropriate end and reachable end. Drawback: In case of complex web service where there exist more number of states, petri net method may lead to state space explosion problem. Because all states could not be captured using petri net method. Only the behavioral compatibility was verified, whether the component web services involved are compatible with its partners. Correctness of the composition was not verified. In (Simmonds et al.), the makers proposed a framework to check behavioral exactness of the Web benefits by watching runtime talks between assistants [4]. Drawback: The safety and liveness properties were captured only for statically composed services. Web service's behavior was not separated.

2.5. Separating operational and control behaviors

Sheng et al. (Sheng et al.) proposed a novel approach for displaying Web administration's conduct [1]. The web administration's conduct was characterized into operational conduct and control conduct. Operational conduct characterizes the business rationale though control conduct directs the path in which benefit must advance. After separating the web service behavior a conversation session was established in which a sequence of messages was exchanged between both the behaviors. A web service's soundness and completeness could be verified if there was a well formed conversation between both the behaviors.

2.6. Behavior modeling of web services

Zakaria maamar et al. (Maamar et al.) proposed an approach for modeling the web service's behavior [2]. The web service's behavior was separated into operational behavior and control behavior. Both the behaviors are formally modeled for performing model checking. Show checking was done keeping in mind the end goal to confirm that operational conduct was all around composed and fits well with control conduct. Display checking was finished utilizing a model checking instrument. Approach: The operational behavior was transformed into kripke structure [12]. The kripke structure was converted into SMV code manually. Then model checking was done in order to check whether the model conforms the specification. The coherent properties was separated from control conduct.

2.7. Representative model checking

Jamal Bentahar et al. (Bentahar et al.) proposed an answer for confirm the conduct of composite Web benefits as far as stop opportunity, wellbeing, and reachability [3]. The composite Web benefit was demonstrated in view of control and operational conduct. These two practices were formally characterized utilizing automata based methods. A model checking approach was used to confirm the rightness of the creation. This method was only applied for static composition. Due to dynamic nature of web service in dynamic composition this method was not applied for dynamic composition by the authors.

3. Architecture

The domain service developed are deployed in glassfish server. The middleware is used to perform dynamic composition and model checking. The cookout management application act as web portal which is developed using JSP. The architecture as shown in Figure 1 designed to support interleaved dynamic composition. Interleaved dynamic composition is a composition in which yield of one web benefit is sustained as contribution to another web benefit.

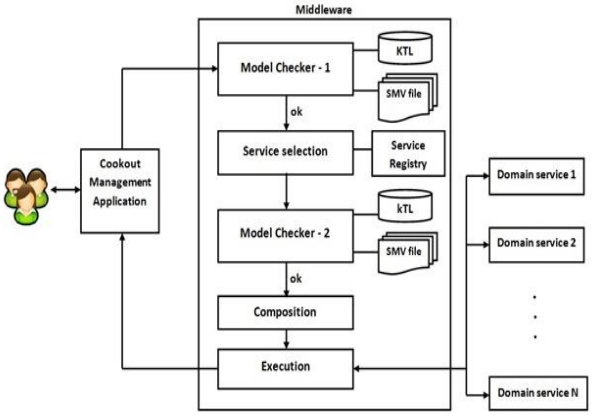

Fig .1: Proposed Approach.

The user request for the available web services through cookout management application. When a request for a domain service is made, composition is modeled as kripke. The kripke contains previously composed services and the service which is to be composed. Each service is represented as a single state in the kripke structure. Kripke structures are generated manually. The kripke structure must be verified against temporal properties which is specified in term of LTL and CTL [3]. The temporal properties include the safety, liveness and deadlock freedom properties. The kripke and the corresponding SMV file for the possible combinations are already modeled. So when a request is made by the user, the corresponding SMV file is fetched from the database and model checking is done to check for the correctness of the composition. If the model is valid, then composition takes place. Composition involves selecting an instance of web service from the service registry, composing and executing it. When an instance of a service is selected it is modeled as kripke. The kripke states corresponds to the control behavior of the service instance. Level-II model checking is done to ensure whether the web service satisfies the temporal properties. If there is no violation in the temporal properties specified, the selected instance of web service is composed and executed by the composer which is developed using Apache axis2 technology. After executing a service the output of the service is passed to the web portal.

3.1. Model checker-i

Model checker-I contains a kripke template, which has the order in which composition must occur in order to reach the final state. In order to check whether the composition reaches the final state or not, it is shown as restricted state machine which address the direct of the structure. The model contains the composed services and the services to be composed. when an input is given, transitions are made to the corresponding states depending upon the states in the kripke template shown in Figure 2. The model and the kripke template are given as contribution to the model checker-I to check whether the displayed limited state machine achieves the last state. If the model reaches the final state, composition will proceed otherwise it halts so that it may not lead to any execution errors.

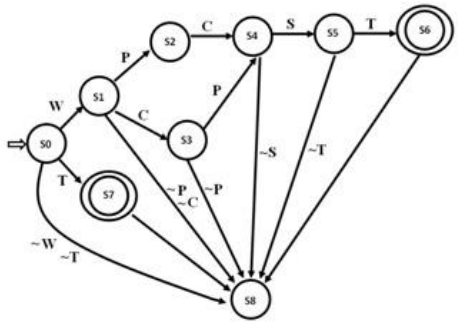

Fig. 2: Kripke Template.

3.2. Model checker-II

Deadlock can be detected at design time using model checking, when detailed internal behavior is available. The internal behavior of the domain services are exposed in WSDL as an abstraction. Which is accessed, preprocessed to generate SMV file for each web service. All kripke files of domain service is assumed to be available to model checker-II. Model checking-II is done after selecting a service from a service registry. The internal behavior of the domains services are already modeled and kept in a database. Depending upon the service selected the corresponding symbolic model verifier file is fetched to find the kripke product. The kripke product is given as input to the model checker-II to find whether the product may reach the final state. If the final state is reached, the selected service will be composed and executed.

4. CASE study - cookout party scheduler

Cookout party is a party where meal is cooked and eaten out of doors with friends. Some basic conditions must be satisfied in order to organize the cookout party successfully. If the party happens at out of doors then there are some basic criteria like climate should be in good condition, need a place for having lunch & for parking. Hence Cookout party application involves the composition of Weather web service, Place Booking web benefit, Catering web benefit, Photography web benefit, Discussion Group web service. This application is chosen because it is suitable for interleaved dynamic composition. Operations of Web Services Weather web service: is utilized to check the climate for the day of the picnic party. Place Booking web benefit: is utilized to book a place to have the picnic gathering. Catering web service: is used to place orders with catering companies. Photography web service: is used to book orders with photo studios, to take photos during the party Discussion group web service: is used to invite friends, send thank note to friends, post the date for the cookout.

4.1. Business rules for cookout party scheduler

Weather clearance must be obtained before booking a place for cookout. Capacity of the place booked should be greater than the number of guest invited. The date of sending thank note must occur after the date of the cookout. Catering web service and car Booking web services must be invoked only after booking a place. There should not be any date conflicts across the services. The catering order placed must be at least ten greater the number of

guest invited. The business rules which is specified in terms of words cannot be fed in to the model checker directly. Model checker supports property specification in terms of LTL/CLT. Hence the above business rules are converted into LTL/CTL which is defined below.

4.2. LTL specification

The business rules extracted from the application are classified into safety, liveness and deadlock freedom properties. Safety property ensures that nothing bad happens, liveness properties are used to check that good thing eventually happens & deadlock freedom property is used to check whether the execution is deadlock free. The safety properties for the application considered and the corresponding LTL specifications are given below.

1) Safety property

Security property guarantees that nothing terrible occurs for instance climate must be checked before booking a place for cookout because if it has been found that the day for which the place was booked has bad climate, the booked place could not be canceled. In order to minimize the execution cost, safety properties must be clearly defined.

a) Weather clearance must be obtained before booking a place for cookout.

LTLSPECG ((state = start)! ((X (state =weather)) &(X(X (state = place)))))

b) In all possible computation of instances, idle state must be followed by either an idle state or invoke state.

LTLSPECG (ws:state = idle ! X (ws:state = idlejws:state = invoke))

c) All web services must be able to reach the final state after processing.

LTLSPECF (ws:state = proc!X(ws:state = exit))

d) Mutual exclusion

LTLSPECG! (state = weather&state = place)

e) Order of composition must be

Weather! place!catering! Photography

LTLSPECG ((state = Idle)! ((X (state = weather)) &(X(X (state = place))) &(X(X(X (state =catering)))) &(X(X(X(X (state = photography)))))))

4.3. CTL specification

All properties specified can be defined in terms of LTL. But CTL specification are considered to be more stronger than LTL.

1) Safety property

The safety property for the cookout party scheduler defined in terms of CTL

a) The capacity of the place booked should be greater than the number of guest invited.
b) SPECAG(cookout place capacity > cookout catering order)
c) Always the Order of composition must be weather! Place! Catering! photography SPECAG((state=Idle)!((AX(state=weather))&(AX(AX(state=place)))&(AX(AX(AX(state=catering))))&(AX(AX(AX(AX(state = photography)))))))
d) Reachability

Reachability is one of the safety property which defines a terminal state. All web service in the composition must eventually reach the end state to indicate that it has been completed.

SPECAG (EF (state = end))

4.4. Model for cookout party scheduler

The workflow is converted to kripke manually. Each web service is modeled as a state in the kripke. The order in which the composition takes place are modeled as transition relation between the kripke states. Even though the kripke models are created statically they will be executed only at runtime. The composition takes place in the order as shown in Figure 3

Weather! Place! Catering! Photography! Discussion group

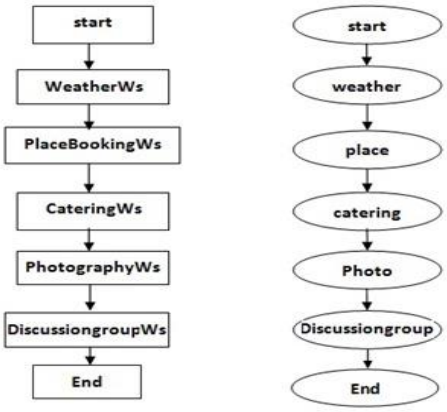

Fig. 3: Workflow and the Corresponding Kripke Structure.

The states in the kripke are hand coded. Any names can be given for states of the kripke which should corresponds to the workflow considered. Since kripke is done manually any mistakes made in the construction of kripke will be considered as a error in the workflow.

5. Implementation

5.1. Technologies used

5.1.1. Java

Web administrations are created utilizing Java Technology APIs and apparatuses gave by an incorporated web administrations Stack. Java turned into an intense improvement stage for Service-Oriented Architecture (SOA). Since vigorous web administrations innovation is the establishment for actualizing SOA, Java now gives the devices that cutting edge undertakings require to coordinate their Java applications into SOA foundations.

5.1.2. J2EE

Java Platform 2 Enterprise Edition (J2EE) is a broadly utilized stage for server programming in the Java programming dialect. J2EE incorporates a few API particulars, for example, web administrations, XML, Java Server Pages, JDBC, RMI and so forth., and is the business standard for actualizing venture class SOA. Java Server Pages (JSP) is a Java innovation that enables programming engineers to serve powerfully produced website pages in view of HTML, XML or other report sorts. JSP innovation isolates the UI from content age, empowering change of the general page format without adjusting the basic dynamic substance i.e. business rationale is isolated from introduction rationale.

5.3. Domain web service creation and dynamic composition

The domain web services are provided by multiple service providers. For the Cookout management application the services are developed using Apache axis2 technology. Netbeans IDE is a development environment for the web services and Derby database is used for database design to the application. The developed domain web services are deployed in Glassfish server. The composer which perform dynamic web service composition is developed using Apache axis2 technology and deployed in Glassfish server [10]. In dynamic web benefit organization the administration choice depends on the sources of info from the client at run time. The service selection, binding and invocation for the composition are performed only at run time. The client gives the input through the web portal which is developed using JSP. The input is sent through the HTTP request to the middleware servlet [11]. The servlet sends a SOAP ask for to the author. In light of the activity

determined in the SOAP ask for the suitable administration was chosen for sythesis.

6. Testing and results

6.1. Service oriented test bed for web service composition

The services involved in Cookout management application are created using Apache axis2. The development environment consists of Netbeans IDE, OpenESB.

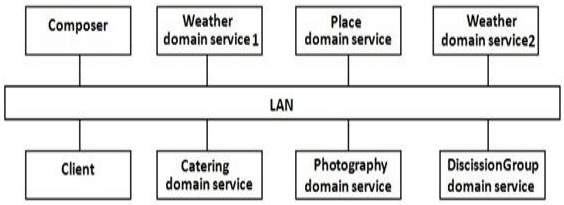

Fig. 4: Service Oriented Test bed for Web Service Composition.

The Web Services, composer are developed and deployed on different machines as shown in Figure 4. All machines are connected through LAN. The input from the client passes through the network and reaches the composer. The composer invoke the services on different machine based on user input. After the service invocation the service executes its own functionalities. The response from the service is send to client through the composer.

6.2. Model checker -i

The temporal property considered here is safety property. Safety property ensures that nothing bad happens. Cookout party application is executed and correctness is checked using safety property. The safety property considered is, weather clearance must be obtained before booking a place for cookout. Two test cases has been considered, case 1 obeys the property and case 2 violates the property.

TEST CASE I

Composition takes place by composing weather web service followed by place booking service. User makes a request for the weather web service in the cookout management application. Level I model checking was done to ensure whether there is any violation in the safety property considered. The output of the model checker 1 is ok, hence a weather web service was selected from the service registry. when the user selects the weather service type model of the system was developed which is given below. The generated finite state machine is verified against the kripke template shown in Figure 5.

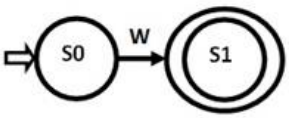

Fig. 5: Finite State Machine of The Model.

Which is given to the model checker as specification. Depending upon the start state of the finite state machine the kripke template is accessed in Figure 6.

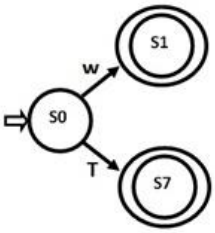

Fig. 6: Finite State Machine of the Model.

Test Case II

Composition takes place by composing the place web service first. User makes a request for the Place Booking web service in the cookout management application. The kripke for the composition was modeled and the corresponding SMV file was executed in the model checker. The output is shown in Figure 7.

Fig. 7: NuSMV Output.

Of the model checker 1 was not ok, because the LTL and CTL formulas specified indicates that weather clearance must be obtained before booking a place for cookout. Hence an error message was displayed which shows what was the error and some suggestions for the user shown in Figure 8.

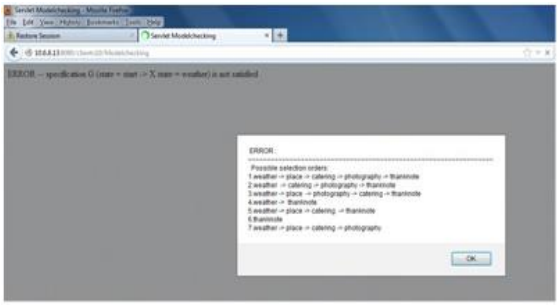

Fig .8: Error Message.

7. Conclusion

In this work, correctness of Web Services composition can be verified. In phase 1 of this work, a business application that dynamically compose the domain web services was designed and model for the composition and temporal properties for the composition were developed. Correctness of composition was verified at level-I. The phase 2 of the work involves the complete implementation of the cookout party application and the verification of correctness of the instances of the web service used in composition along with the verification of correctness of whole composition.

References

[1]. Sheng, M., Maamar, Z., Yahyaoui, H., Bentahar, J., Boukadi, K. (2010). Separating operational and control behaviors: A new approach to web services modeling. IEEE Internet Computing, 14 (3), 3039. https://doi.org/10.1109/MIC.2010.78.

[2]. Quan Z. Sheng, Zakaria Maamar, Lina Yao, Claudia Szabo, Scott Bourne (2014): Behavior modeling and automated verification of Web services. Information Science 258: 416-433. https://doi.org/10.1016/j.ins.2012.09.016.
[3]. Jamal Bentahar, Hamdi Yahyaoui, Melissa Kova, Zakaria Maamar,(2013) Symbolic model checking composite Web services usingoperational and control behaviors. Information Science 508522.
[4]. Jocelyn Simmonds, Yuan Gan, Marsha Chechik,(2009) Runtime monitoring of web service conversations, IEEE transactions on services computing, VOL. 2, NO. 3.
[5]. Cimatti, A et al, (2002). NuSMV 2: An OpenSource tool for symbolic model checking. Proceeding of the international conference on computer-aided verification (CAV) (Vol. 2404, pp. 241268). LNCS. https://doi.org/10.1007/3-540-45657-0_29.
[6]. Syed Adeel Ali, Partha S. Roop, Ian Warren: Stateful Web Services - Auto Modeling and Composition. IEEE International Conference on Web Services (ICWS) 2013, pp. 284-291.
[7]. Kil, H., Nam,W. and Lee, D.(2013) Behavioural description based web service composition using abstraction and refinement, Int. J. Web and Grid Services, Vol. 9, No. 1, pp. 5481. https://doi.org/10.1504/IJWGS.2013.052849.
[8]. Xitong Li, Yushun Fan, Quan Z. Sheng, Zakaria Maamar, and Hongwei ZhuA, (2011) Petri Net Approach to Analyzing Behavioral Compatibility and Similarity of Web Services, Algeria (2011), pp. 8793.
[9]. Melliti Tarek,Celine Boutrous-Saab, Verifying correctness of Web services choreography (2006). ECOWS'06 pp. 306-318. https://doi.org/10.1109/ECOWS.2006.38.
[10]. Xu Guoyan, Yang Li, Kang Jiehua, Ping Ping, Lv Xin, (2016). Trustworthiness calculation of composite Web service output data based on how provenance, Computer Science & Education (ICCSE).
[11]. Li Bao, Yi Deng (2017). A Pi-Calculus Based Context-Aware Model for Web Service Composition, Information Science and Control Engineering (ICISCE).
[12]. Xu Wenjun, Yin Zhenyu, Gu Ai, Yao Kaifeng (2017). Design and Implementation of Web Services Client Based on ARM Linux Embedded Platform, 10th International Conference on Intelligent Computation Technology and Automation (ICICTA).
[13]. Ying Wu, Rui Zhang, Rui Xue, Ling Liu (2017). Multi-Client Verifiable Computation Service for Outsourced Data, IEEE International Conference on Web Services (ICWS). https://doi.org/10.1109/ICWS.2017.65.