DATA SCIENCE JOURNAL

Xiang, D et al 2015 The Executable Invocation Policy of Web Services Composition With Petri Net. *Data Science Journal,* 14: 5, pp.1-15, DOI: http://dx.doi.org/10.5334/dsj-2015-005

PROCEEDINGS PAPER

The Executable Invocation Policy of Web Services Composition With Petri Net

Dongming Xiang[2], Nengfu Xie[3], Bingxian Ma[1] and Kai Xu[1]

[1] School of Information Science and Engineering, University of Jinan, Jinan, 250022, China
ise_mabx@ujn.edu.cn

[2] Department of Computer Science and Technology, Tongji University, Shanghai, 201804, China
flysky_xdm@163.com

[3] Agricultural Information Institute, the Chinese Academy of Agricultural Science, Beijing, 100081, China
xienengfu@caas.cn

The web service composition execution engine is a critical problem for web service composition. Based on Petri net and the analysis of structural relationships among web services, the invocation sequence of web service composition and its related invocation policies are fully studied in this paper. Also, the executable invocation policies of web service composition are successfully constructed based on Petri net. Finally, an example of a scientific computation service is given to validate the effectiveness of this method.

Keywords: Petri net; Web service; Service composition; Invocation sequence; Scientific computation service

1 Introduction

Following the pace of big data, more and more data will realize sharing and application in the form of DaaS (Data-as-a-Service). Therefore, more data and information services will emerge on the Internet. What is more, the collaboration (composition) of multiple web services is required to satisfy users' complex demands. In addition, "big data, big service" also brings new technology requirements and challenges for service computing. In particular, how to achieve an efficient and real-time collaboration of web services has become a key question in web service composition. Thus, a fast and efficient web service composition execution engine is the most important technology for meeting the new requirements.

A considerable amount of effort has been carried out on the researching of a web service composition execution engine. The WS-CDL$^+$ execution engine (Ai, Tang, & Fidge, 2011) achieved completely the specification standards of WS-CDL. It directly executes WS-CDL+ documents with corresponding configuration files and realizes scheduling coordination for web services. Kang et al., (2007) proposed a numeric segmentation algorithm for composite services developed using BPEL (Business Process Execution Language) (Jordan & Evdemon, 2007). In this algorithm, the sub-process was executed in a dispersed form, in which the concurrency and throughput were all improved. Composite services were divided into different component services, and each sub-service was arranged according to its own execution engine (Kang et al., 2011). Yu (2007) proposed a BPEL execution engine based on P2P. However, this engine did not deal with static instances. Business processes were arranged and executed based on web services according to domain ontology, and the WebFlowAH platform was constructed (Mendes & Paulo, 2009). Narendra and Orriens (2007) presented a conceptual model that could track the demand changes during web service execution. Park and Park (2008) adopted the intentional XML data and invoked external services on related nodes. Meanwhile, they employed an A* heuristic search algorithm to find the optimal trace and greedy algorithm to generate an efficient solution in a short time. Tsamoura et al. (2011) and Darmstadt et al. (2009) studied the execution of distributed workflows. The former reduced the response time of multi-pipeline invocations of remote web services. The latter guaranteed the correctness of control flows from the point of view of security and realized the communication and data transmission between web services based on "process slip". The fault web services were replaced to realize the forward recovery, and Colored Petri Nets (CPNs) were utilized and

represented compensation flows to achieve backward recovery. Finally, an effective algorithm was proposed to deal with transactional composite service invocation and strategy recovery (Cardinale & Rukoz, 2011).

The above research examples have proposed different execution frameworks of web service composition or optimized their executive processes from different perspectives. However, most of these are based on the service process orchestration to schedule services, such as BPEL, rather than considering the real-time collaborative invocation and problem analysis in web service composition. The main reason for this is that the research lacks a relatively flexible, describable, and simple formal model of automatic web service composition. Correspondingly, it is hard to construct an execution engine (Suzumura, Trent, Tatsubori, Tozawa, & Onodera, 2008). In addition, because many complicated data associations and structural associations are presented within web service composition, if the complex semantic associations are taken into account, it will become more difficult to construct and analyze the logical flow of web services.

In contrast, the complex process logic and structural relationships among web services can be vividly described by the Petri net of web service composition (Xiang et al., 2012). Many works (Cardinale & Rukoz, 2011; Xiong et al., 2010; Tang et al., 2007; Ding et al., 2008; Tang et al., 2011; Tan et al., 2010) have studied the modeling and analysis of web service composition based on the Petri net. However, because of the time complexity of Petri net's reachability analysis, many works only focused on static modeling, off-line property analysis, or quantitative evaluation, rather than using the Petri net for dynamic execution and real-time analysis of web service composition.

Therefore, based on the Petri net description of web service composition, dynamic execution and real-time scheduling policies are proposed in this paper. First, we discuss description methods of web service composition and composite web services with Petri net. Then, we concretely analyze structural relationships within the web service composition and design executable invocation of web service policies based on Petri net. Finally, an execution engine based on Petri net is constructed for web service composition and composite web services. In a practical application of web service composition, this paper will provide an effective technical solution in applying Petri net theory and its relevant analysis method to a real-time execution and analysis.

The rest of this paper is organized as follows. Related concepts and knowledge are introduced in Section 2. Section 3 lists some concepts related to web service. In Section 4, we concretely analyze the possible structural relationships within web service composition. Section 5 and Section 6 put forward a web service composition execution algorithm using Petri net and apply this method to a practical instance. Finally in Section 7 we summarize the work presented in this paper and point out further work.

2 Related Concepts and Knowledge

2.1 PNML+OWL

PNML (Petri Net Markup Language) (Jüngel et al., 2000) stores and describes a Petri net. It is used mainly to solve the problem of sharing Petri nets among different tools. However, due to the dependence on syntaxes, it cannot realize the interoperability of Petri nets.

OWL (Web Ontology Language) (McGuinness & van Harmelen, 2011) is a language system, and its theoretical basis is description logic. It can create ontology by using different attributes, such as an object attribute, data attribute, and domain attribute. Web information with semantic information in OWL is easy for machines to understand. OWL inherits the basic way of stating the fact of RDF (resource description frame) and the hierarchical structure of RDFS (RDF Schema) with classes and properties. By expanding upon these, OWL adds many new words and overcomes the problem of RDF/RDFS not describing concepts and attributes well.

In our related research (Ma & Xu, 2009; Ma & Xie, 2010), web services and their services composition have been modeled with Petri net. To be specific, the transition label in PNML represents the information about web services, such as service name, input and output, etc.; the place label in PNML represents inputs and outputs of web services; the flow label defines the relations between web services and their inputs or outputs. In addition, based on the semantic database, we have added some semantic information into the inputs and outputs of the web services in PNML documents so that the PNML+OWL description of the web services can be acquired.

2.2 XFire

XFire (Codehaus) is the next framework for Java SOAP (Simple Object Access Protocol). It bridges the gap between POJO (Plain Old Java Objects) and SOA (service-oriented architecture). Due to the use of a programming interface and its support for web service standards, XFire has a relatively simple service-oriented development. It simplifies the process of converting a Java application into web services and provides a simple

and feasible way for enterprises to build SOA architectures. By building on a low memory model (STAX), it has high performance characteristics.

Based on XFire, web services can be invoked instantly. First, using XFire, we generate the web service clients, and then we invoked the web services by using their methods' names and input parameters. If the call result from the web services does not belong to the array type, it should be further analyzed; otherwise, it is the desired result.

3 Atomic Web Service and Composite Web Service

Web service composition can be categorized as orchestration (Lapadula, Pugliese, & Tiezzi, 2007; Wang, Dai, Hou, Fang, & Ren, 2009) and choreography (Valero et al., 2009). The general process of an "Orchestrated" web service composition is that this service composition can be considered as a single atomic web service. A "Choreographed" web service composition collaboratively invokes each sub-service during its execution. Sub-services in web service composition can be atomic web services or composite web services.

3.1 Atomic web services

An atomic web service usually refers to a service that has a relatively simple or independent function and provides single interfaces that meet specific requirements. In addition, an atomic web service can be designed and deployed based on general industrial standards or techniques, such as XFire (Codehaus).

3.2 Composite web services

Atomic web services can be "orchestrated" or "choreographed" into a composite web service to meet complex user demands. Composite web services are divided into two types according to the mode of the web service composition, either orchestration or choreography. An orchestration composite web service is considered to be an atomic web service while a choreography composite web service process structure is a part of the web service composition. By default, in the rest of this paper, composite web service refers to a choreography composite service.

During the choreography of composite web services, it is not easy to construct a complex service process depending only on the data association among services. It is also necessary to use control structures such as loop structure and choice. A loop control structure is used to control the execution times of sub-services while a choice control structure affects service selection and execution paths within a chosen structure. The control structure can be designed as a web service.

In our related work (Ma & Xu, 2009; Ma & Xie, 2010), we placed elements of the Petri net corresponding to the input and output of atomic web services or orchestration composite services; the transitional elements correspond to atomic web or orchestration composite services, and the Petri net of a composite web service can be generated with this method.

Definition 1 (Petri net of composite web services) The Petri net of a composite web service is an 8-tuple $\sum = (S, T, CS, CT, F, CF, M, L)$, where

(1) S is the set of places. A place represents the input, output, precondition, or execution effects of sub-services in the composite web services;
(2) T is the set of transitions. A transition corresponds to an atomic service or an orchestration composite service;
(3) $F \subseteq (S \times T) \cup (T \times S)$ is the set of flows;
(4) CS is the set of control places. A control place is one input or output of a control condition;
(5) CT is the set of control transitions. A control transition represents a loop control condition or a choice control condition;
(6) $CF \subseteq (CS \times CT) \cup (CT \times CS)$ is the set of control flows;
(7) $M: S \cup CS \rightarrow \{0, 1, 2, \ldots\}$ is the number of tokens in places;
(8) $L: S \rightarrow D \cup \{ \tau \}$ is the semantic markup function, where D is the set of classes and instances from one domain ontology. τ means an empty semantic.

The foundation of a Petri net of a composite web service is the data associations among sub-services and related structural relationships that determine the basic data flow (*F*) among services. On the other hand, control structures in composite web services include control places, control transitions, and control flows. The control structures influence the services' flow direction and co-scheduling in the form of a control flow (*CF*).

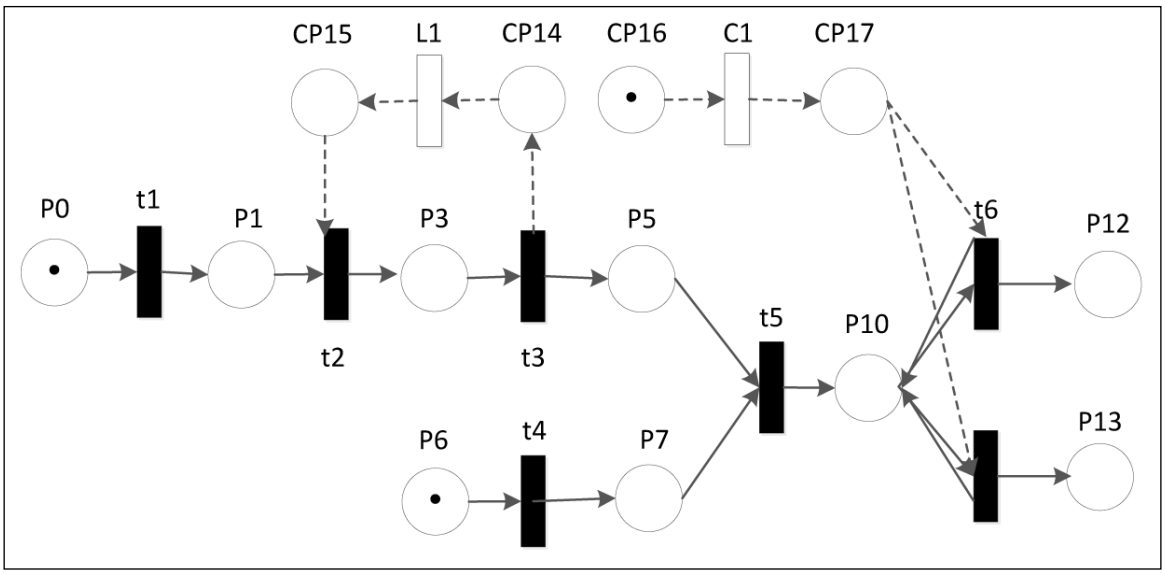

Figure 1: Petri net of a composite web service.

Figure 1 shows the Petri net of a composite web service, where t1-t7 are transitions that represent atomic services. L1 and C1 are control transitions. L1 represents the loop condition, which influences the invocation of t2 and t3. C1 represents the choice condition that determines which web service is to be invoked next.

3.3 Petri net of a web service composition

If we ignore the complex business processes within a composite web service and only regard it as a web service that meets specific functional requirements and has multi-inputs and multi-outputs, then we further abstract and obtain the Petri net of a composite web service. Specifically, the initial input of the web service forms the input place elements in the Petri net, and the users' end needs become the output place elements while the whole web service body is represented as a transition element. Afterwards, the abstracted Petri net model of the composite web service can be obtained.

Based on an abstraction of the Petri net of a composite web service and the data association among web services, we can get the Petri net and its PNM+OWL description of the web service composition (Xiang et al., 2012; Ma et al., 2013).

Definition 2 (Petri net of web service composition) The Petri net of a web service composition is a 5-tuple $\sum = (S, T, F, M, L)$, where

(1) S is the set of places. A place corresponds to an input, output, precondition, or execution effect of some atomic service or the abstracted composite web services;
(2) T is the set of transitions. A transition refers to an atomic service or the abstracted composite web services;
(3) $F \subseteq (S \times T) \cup (T \times S)$ is the set of flows;
(4) $M: S \cup CS \rightarrow \{0, 1, 2, \ldots\}$ is the number of tokens within places;
(5) $L: S \rightarrow D \cup \{ \tau \}$ is a semantic markup function, where D is a set of classes and instances from one domain ontology. τ means empty semantic.

3.4 Layered service composition architecture

From the basic components perspective, based on the abstraction of composite web services, the web service composition system can be treated as 7-layer architecture (**Figure 2**).

(1) Base support layer: this layer supports the deployment, invocation, and registration of the web services;
(2) Atomic service layer: this layer contains atomic web services and is the foundation of the design, orchestration, and invocation services;
(3) Composite service layer: this layer contains composite web services;
(4) Service abstraction layer: composite web services are abstracted in this layer, which is the foundation of the service composition;

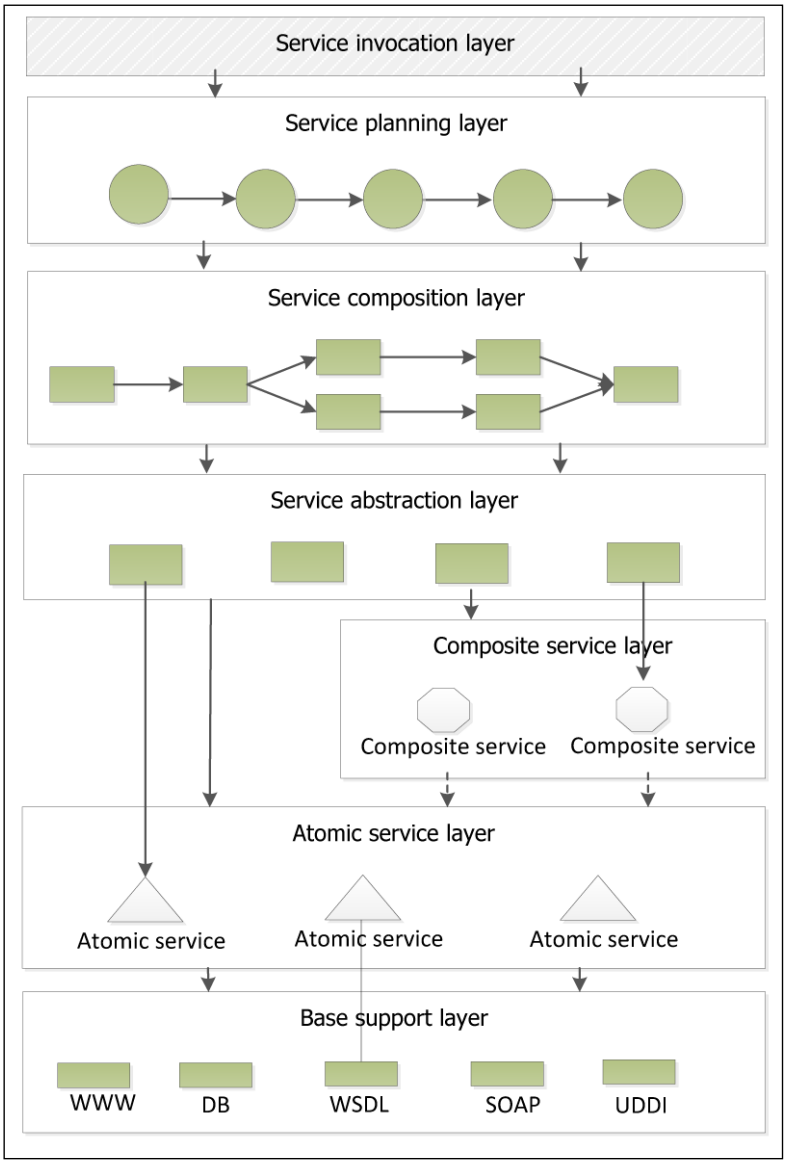

Figure 2: The layered service composition architecture.

(5) Service composition layer: web services are composed in this layer according to semantic associations;
(6) Service planning layer: the invocation sequence of web services is generated here based on the Petri net of the web service composition;
(7) Service invocation layer: web services are sequential invoked in this layer according to the invocation sequence of web service composition.

This layered service architecture does not contain control structures in the service composition layer, which reflects that there are only data associations among the web services based on data flows. However, the composite service layer realizes the integration of the data and control flows based on a concrete service process. In addition, because the Petri nets at the service abstraction layer are abstracted from (composite) services, some control structural relationships may exist. This abstracted model does not reflect a real service's execution. It only needs to get concrete Petri nets from the composite service layer during the execution of the composite services.

4 Structural Relationship Among Web Services

In order to invoke each service in web service composition in good order, the relationship among the web services must first be analyzed. Based on the Petri nets of the web service composition, it is feasible to get these relationships from the structural relation among the transitional elements of the Petri net.

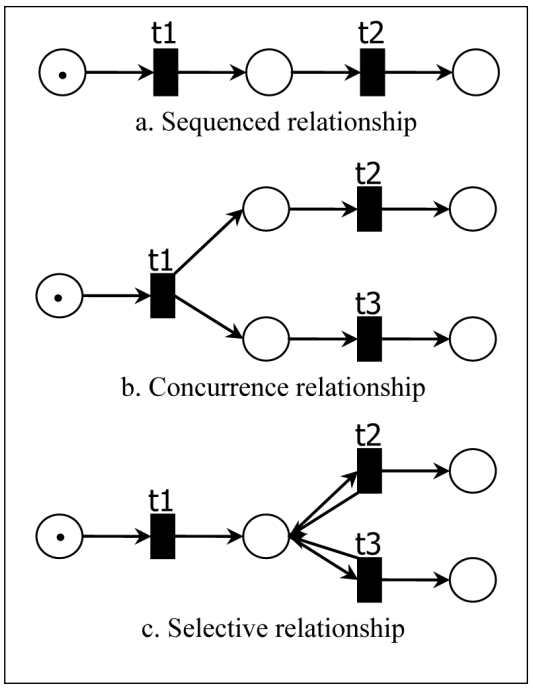

Figure 3: The basic structural relationship among web services.

4.1 Prepositive web services and postpositive web services

Definition 3 (Prepositive web services) For a web service composition and its Petri net model $\sum = (S, T, F, M, L)$ for web service t_i, the prepositive web services can be defined as Pro $(t_i) = {}^\bullet({}^\bullet t_i) - \{t_i\}$, $i \in \{1,2,\ldots,|T|\}$. If ${}^\bullet t_i = \varnothing$, then Pro $(t_i) = \varnothing$;

Definition 4 (postpositive web services) For a web service composition and its Petri net model $\sum = (S, T, F, M, L)$ for web service t_i, the postpositive web services can be defined as Post $(t_i) = (t_i^\bullet)^\bullet - \{t_i\}$, $i \in \{1,2,\ldots,|T|\}$. If $t_i^\bullet = \varnothing$, then Post $(t_i) = \varnothing$;

Conclusion 1 In web service composition for web service t_i and its prepositive service set $Pro(t_i)$, if $\forall\ t_j \in Pro(t_i)$ $(i,j \in \{1,2,\ldots,|T|\}, j \neq i)$, then the input-output association between tj and ti can be discovered. Similarly, for the postpositive service set $Post(t_i)$, if $\forall\ t_j \in Post(t_i)$ $(i,j \in \{1,2,\ldots,|T|\}, j \neq i)$, then the input-output association between t_i and t_j can also be acquired.

4.2 Structural relationships in web service composition

In the Petri nets of web service composition (Chen et al., 2010) or composite web services, it is easy to analyze and acquire three basic structural relationships from the structural view: sequenced relationships (*Sequence*), concurrent relationships (*Concurrence*), and selective relationships (*Choice*). All of these constitute the foundation of structural relationships within a web service composition, shown in **Figure 4**.

For a Petri net of a web service composition $\sum = (S, T, F, M, L)$ with a basic structural relationship *R* belonging to {Sequence, Concurrence, Choice}:

(1) Sequenced Relationship
For two web services t_i, and t_j, the postpositive web service of t_i is a nonempty set $Post(t_i)$, where $i, j \in \{1,2,\ldots,|T|\}$, and $i \neq j$. If $|Post\ (t_i)| = 1$ and $t_j \in Post\ (t_i)$, then the relationship between t_i and t_j is a sequenced relationship (*Sequence*), which can be denoted as $<t_i, t_j> \in Sequence$. If the relationships among $t_1, t_2, \ldots, t_n$ belong to *Sequence* successively, $<t_1,t_2>, <t_2,t_3>, \ldots, <t_k,t_{k+1}>, \ldots <t_{n-1},t_n> \in Sequence$, the corresponding expressed sequence of this relationship is $t_1 t_2 \ldots t_n$. This is shown as t1 and t2 in **Figure 3a**.

(2) Concurrence Relationship
For web service t_i, $i \in \{1,2,\ldots,|T|\}$, the postpositive web service of ti is a nonempty set $Post(t_i)$. If $|t_i^\bullet| > 1$, $|Post\ (t_i)| > 1$, t_{i1} and $t_{i2} \in Post\ (t_i)$, ${}^\bullet t_{i1} \wedge {}^\bullet t_{i2} = \varphi$, then the relationship between t_{i1} and t_{i2} is a *Concurrence* relationship (*Concurrence*), which can be denoted as $<t_{i1}, t_{i2}> \in Concurrence$. Similarly, if there exists a web service t that satisfies $t_1, t_2, \ldots, t_n \in Post(t)$ and with the condition that the relationship between the elements of the set $\{t_1, t_2, \ldots, t_n\}$ is *Concurrence*, then the corresponding expressed sequence is $(t_1 \& t_2 \& \ldots \& t_n)$. This is shown as t2 and t3 in **Figure 3b**.

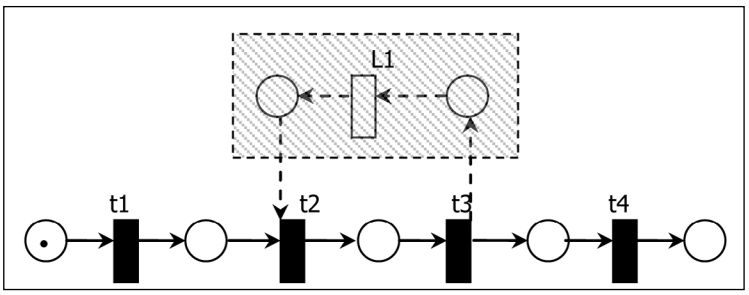

Figure 4: The loop structure.

In addition, from a general perspective, for multiple web services that meet their input parameter values, if there is no data association among the inputs and outputs of the web services, these web services are independent of each other and can be identified as concurrence.

(3) Selective Relationship
For web service t_i, $i\in\{1,2,...,|T|\}$, the postpositive web services of t_i is a nonempty set *Post*(t_i). If $|t_i^\bullet|=1, |Post(t_i)|>1$, and t_{i1} and t_{i2} all belong to *Post*(t_i), then the relation between t_{i1} and t_{i2} is the selective relationship (*Choice*), which can be denoted as $<t_{i1},t_{i2}>\in Choice$. Similarly, if there exists a web service t that satisfies $t_1,t_2,...,t_n\in Post(t)$, with the condition that the relationship between the elements of the set $\{t_1,t_2,...,t_n\}$ is a selective relationship, then the corresponding expressed sequence is $(t_1|t_2|...|t_n)$. This is shown as t2 and t3 in **Figure 3c**.

Theorem 1

(1) If $<t_1,t_2,...,t_n>\in Sequence$, there is only one element order for $\{t_1,t_2,...,t_n\}$ that satisfies $<t_1,t_2>,<t_2,t_3>,...,<t_k,t_{k+1}>,...,<t_{n-1},t_n>\in Sequence$, $|Post(t_i)|=1$ (i=1,2,...,n-2,n-1), and $t_{k+1}\in Post(t_k)$, where k=1,2,...,n-2,n-1;
(2) If $<t_1,t_2,...,t_n>\in Concurrence$, $t_i, t_j\in\{t_1,t_2,...,t_n\}$, and $i\neq j$, then $<t_i, t_j>\in Concurrence$;
(3) If $<t_1,t_2,...,t_n>\in Choice$, $t_i, t_j\in\{t_1,t_2,...,t_n\}$, and $i\neq j$, then $<t_i, t_j>\in Choice$.

Based on the definition of structural relationships, we can get the basic relational expression (sequence) between web services and their postpositive services as follows.

Suppose there is a web service t_i and its postpositive web service is a nonempty set $Post(t_i)$, where $Post(t_i)=\{t_{i1},t_{i2},...,t_{ik}\}$, and $k\in N$. According to the three kinds of basic structural relationships, the relational expression between t_i and $Post(t_i)$ can be denoted as $G(t_i,Post(t_i)) = t_i R(Post(t_i))$, $R\in\{Sequence , Concurrence , Choice\}$.

(1) If $<t_{i1},t_{i2},...,t_{ik}>\in Sequence$, according to theorem 1, definition 5, and the condition $|Post(t_i)|>=1$, it can be concluded that $|Post(t_i)|=1$. If $t_{ik}\in Post(t_i)$, then $G(t_i,Post(t_i)) = t_i t_{ik}$;
(2) If $<t_{i1},t_{i2},...,t_{ik}>\in Concurrence$, then $G(t_i,Post(t_i))=t_i(t_{i1}\&t_{i2}\&...\&t_{ik})$;
(3) If $<t_{i1},t_{i2},...,t_{ik}>\in Choice$, then $G(t_i,Post(t_i))=t_i(t_{i1}|t_{i2}|...|t_{ik})$.

If there exists a web service *t* and many structural relationships in its postpositive service Post(t)($Post(t)\neq\varnothing$), the relational sequence between t and Post(t) can be expressed by a nested basic structural relationship. As for this nested structure, we ignore its detailed and complex relationships and just consider that it only satisfies a single specific structural relationship from an overall perspective, which we call the Service Structural Body (*SSB*). On this basis, the basic structural relationship can be extended and applied to the web service and the Service Structural Body. For example, there are a web service t and its postpositive web service set $\{t_1,t_2, t_3\}$. The relationship between t_2 and t_3 is selective, i.e., $<t_2, t_3>\in Choice$; therefore, their relationship expression (sequence) is $(t_2|t_3)$. The relationship between t_1 and the (selective) Service Structural Body $<t_2,t_3>$ is concurrence, which can be denoted by $<t_1,<t_2,t_3>>\in Concurrence$. Thus their relational sequence can be expressed as $G(t,Post(t))=t(t_1\&(t_2|t_3))$.

4.3 Control structures in composite web services

(1) Loop structural relationship, shown in **Figure 4**
In the Petri net of a composite web service, the loop controller is L_i, $t_1,t_2,...,t_n$ are web services, and $i\in N$, L_i refers to a control transition which controls(influences) the loop structure body $L(t_1,t_2,...,t_n)$. In a loop structure body, there exist basic or nested structural relationships. In a Petri net with loop

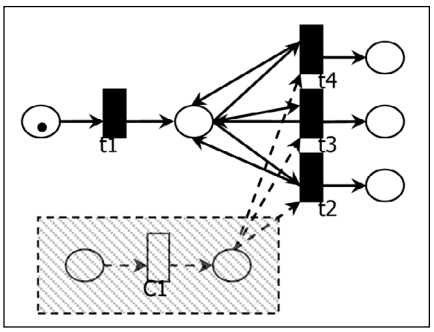

Figure 5: The selective structure with choice conditions.

structures, $|{}^{\bullet}L_i|=|L_i^{\bullet}|=1$. Moreover, we regard the set $\{t_j \mid t_j \in Post(L_i)\}$ as the beginning service of the loop and the set $\{t_k \mid t_k \in Prot(L_i)\}$ as the ending service of the loop. The corresponding loop expression is represented as $Loop(t_1,t_2,...,t_n: L_i) = (L(t_1,t_2,...,t_n):L_i)$.

(2) Selective relationship with choice conditions shown in **Figure 5**
Suppose the selection controller (conditions) is C_i and $t_1,t_2,...,t_n$ are web services, $i \in N$, C_i refers to the control transition that influences the selective execution paths, $|{}^{\bullet}C_i|=|C_i^{\bullet}|=1$, $Post(C_i)=\{t_1,t_2,...,t_n\}$, and $Pro(C_i) = \varnothing$. The corresponding selection expression is $Choice(t_1,t_2,...,t_n: C_i) = (t_1|t_2|...|tn:C_i)$.

5 Invocation of Web Service Based on Petri Net

Based on the analysis of the structural relationship and the PNML+OWL of web service composition, the invocation sequence of the web service composition can be proposed as follows.

5.1 Invocation sequence of web service composition

When considering the complexity of structures of a Petri net of web service composition and its composite services, it is not easy to achieve an automatic analysis and invocation of web services located in the complex structures. Thus, in order to maintain the structural relationship among the web services and promote automatic analysis and execution, the web services and their structural relationships should be presented in the form of symbol sequences called the invocation sequence of web services.

Definition 5 (Invocation sequence of web services) The invocation sequence of web services can be defined as $S = Seq(WS) = R\left(\bigcup_{i=1}^{m} Seq(WS_i)\right)$, where *WS* is a web service set$\{t_i, i=1,2,...,n\}$ and $WS_i \subseteq WS$. This satisfies $\bigcup_{i=1}^{m} WS_i = WS$ and $\bigcap_{i=1}^{m} WS_i = \varnothing$, where t_i is a web service, Seq(WS) represents the invocation sequence based on *WS*, and *R* represents the structural relationships among the web services. If there exists some relationship covering WS_i, namely $R(WS_i)$, then $Seq(WS_i) = R(WS_i)$; otherwise, there must be a web service set WS_i' that can be covered by some relationship that satisfies $WS_i' \subset WS_i$ and $R(WS_i')$.Then $S_i = Seq(WS_i) = Seq\ (R(WS_i') \cup \overline{WS_i'})$.

The following is an example of the invocation sequence of web services in **Figure 1**:

(1) $S_1 = Seq(t_2,t_3) = Loop\ (t_2,t_3:L_1) = (t_2t_3:L_1)$;
(2) $S_2 = Seq(t_1, S_1) = Sequence(t_1, S_1) = t_1S_1$;
$=>S_2 = t_1S_1 = (t_2t_3:L_1)$;
(3) $S_3 = Seq\ (S_2,t_4) = Concurrence(S_2,t_4) = (S_2\&t_4)$;
$=>S_3 = (S_2\&t_4) = ((t_2t_3:L_1)\&t_4)$;
(4) $S_4 = Seq\ (S_3,t_5) = Sequence(S_3, t_5)= S_3t_5$;
$=>S_4 = S_3t_5 = ((t_2t_3:L_1)\&t_4)t_5$
(5) $S_5 = Seq\ (t_6,t_7) = Choice\ (t_6,t_7:C_1) =(t_6|t_7:C_1)$;
(6) $S_6 = Seq\ (S_4, S5) = Sequence\ (S_4, S5) = S_4S_5$;
$=>S_6= S_4S_5= ((t_2t_3:L_1)\ \&t_4)\ t5\ (t_6|t_7:C_1)$

After iterating the above six formulas and replacing similar structural relationship expressions, the invocation sequence of the web services is $S=Seq(t_1,t_2,t_3,t_4,t_5,t_6,t_7)=(t_1(t_2t_3:L_1)\&t_4)t_5(t_6|t_7:C_1)$. In the invocation

sequence of the web services, the priority of structural relationships is regulated as: sequence > concurrence >choice. In addition, the relational structure for the symbol "("and")" should be given a higher priority.

In conclusion, the main function of the invocation sequence of web services is to reflect the structural relationship among the web services, describe their execution sequence, and embody the planning results. Moreover, it can also provide a necessary basis for following coordinate scheduling and execution of multiple web services.

5.2 Invocation scheduling of web services

Definition 6 (Relationship identifier) The relationship identifier R={ ti | Ai | Ci | Li , i=1,2,...,n }, where i is an integer, *ti* represents an atomic web service, *Ai* represents the concurrence relationship, *Ci* represents the selective relationship, and *Li* represents the loop relationship. The invocation scheduling for web services is given as follows:

Step1: Retrieve the input-output associations among the web services from PNML+OWL;
Step2: Simplify the invocation sequence of the web services by relationship identifiers. The purpose of this step is to locate the scope of a certain structural relationship so as to schedule web services in the relationship;
Step3: Extract structural relationships. Based on the specific relationship identifier and the invocation sequence of web services that were generated in Step 2, the web services are further invoked according to their structural relationships.

5.3 Invocation condition of choice sequence

Definition 7 (Choice sequence) If there are N sequences, such as S_1', S_2', S_n', in a selective structure, each sequence is called a choice sequence.

Each first web service $t_{i1},t_{i2},...,t_{ik},...,t_{in}$ can be extracted from sequences $S_1',S_2',...,S_n'$. Suppose t_{pik}(k=1,2,...,n) is an input for each first web service, it is associated with the output p_j of web service t_j. In the web service composition, it satisfies $t_j^{\bullet} = \bigcap_{m=1}^{n} {}^{\bullet}t_{im}$ and $\bigcap_{m=1}^{n} Pro(t_{im}) = t_j$, where Pre ($t_{im}$) is the prepositive service set of web service t_{im}. The semantic association between p_j and p_{ik} is reasoned and obtained, and then the selection is performed as follows:

(1) If the semantic association between p_j and p_{ik} is a parent-son relationship, an instance-class relationship, or a complete equivalence relationship and p_j and p_{ik} are of consistent data types, then the input (p_j) of web service p_j can meet the input (p_{ik})of web service t_{ik}, and the sequence S_i' has been chosen.
(2) If the semantic association between p_j and p_{ik} is a parent-son relationship or an instance-class relationship, then the semantic association between the values of p_j and p_{ik} should be further judged. Suppose the value of p_j is v_{pj} only if the semantic association of v_{pj} and p_{ik} is a parent-son relationship or an instance-class relationship, then the output p_j of web service t_j can meet the input p_{ik} of web service t_{ik}. Thus, the sequence S_i' has been chosen.

5.4 Invocation policy of structural relationship

Suppose a relationship identifier Ri exists, and its structural relationship is R(t1, t2,...,tj), j ∈N. The invocation of web services can be treated as follows:

(1) Atomic web services, that is Ri∈ {ti ; i =1, 2,...,n}. Invoke each web service directly;
(2) Concurrence structure relationship, that is Ri∈ {Ai, i = 1, 2, ..., n} and R(t1, t2,...,tj) = (t1&t2&...&tj). Invoke web service t1, t2, ... tj in concurrent threads;
(3) Selective structure relationship, that is Ri∈ {Ci, i = 1, 2,..., n} and R(t1, t2,...,tj) = (t1|t2|...|tj). Screen out the set S of web services that meet the invocation condition of the choice sequence from t_1, t_2,..., t_j, and then invoke each web service of *S* in concurrent threads;
(4) Selective structure relationship with choice conditions, that is Ri∈ {Ci, i = 1, 2,..., n} and R(t1, t2,...,tj) =(t1|t2|...|tj: ck), where ck is a choice condition. Screen out set S of the web services from t1, t2,..., tj to ck and then invoke each web service of S in concurrent threads;
(5) Loop structure relationship, that is Ri∈{Li, i = 1, 2,... , n} and R(t1, t2,...,tj) = (L(t1, t2,...,tj) : lk), where L(t1, t2,...,tj) is its loop structure body. Invoke the web services of L(t1, t2,...,tj) first, and then execute

the method that published in the service Lk. According to the return value, subsequently re-invoke the web services of L (t1, t2,...,tj) constantly until the value is false.

In practice, there may be a nested structural relationship that contains multiple or different kinds of service structural relationships, just like R (t_1, t2,...,t_j) =(t_1&t_2&...&t_k&R1), where R1 is a relationship identifier of the structural relationship R'(t_{k+1}...,t_j). Therefore, it is necessary to nest the above invocation policies to deal with this relatively complex structural relationship.

5.5 Web services composition invocation algorithm based on Petri net

Inputs: invocation sequence and PNML+OWL file of web service composition
Outputs: invocation results of web service composition

Step 1: Extract input-output associations between web services. From the flow relation sets (the "arc" label) in PNML+OWL, the input-output associations between web services are analyzed and extracted. Then the data association Hash table (*IORelevancyMap*) can be further created. Suppose the output p_i of web service t_i is associated with the input p_j of web service t_j, where i and j are integers, then the corresponding storage format of the Hash table is key= "t_i : p_i ", and value= Hash(key)= "t_j : p_j".

Step 2: Simplify web service invocation sequence. For the invocation sequence of web services *S*, the character *c* can be read from left to right in turn. If *c* is '(' , *c* and the following characters are pushed into a stack until the character that is read from *S* is ')'. If *c* is ')', characters are popped from the stack until the character that pops from the stack is '('. After this, the popped string (characters) that match '(' with ')' are processed as follows:

(1) If the string contains the character '&', the string is denoted as a concurrence structure sequence by the relationship identifier *Ai* (i=1, 2,...,k). *Ai* and its corresponding concurrence sequence t_1&t_2&...&t_n are added into the Hash table *Concurrent Map*. The Hash table *Concurrent Map* is *key* = *Ai* and *value*= Hash (*key*)= "t_1&t_2&...&t_n",
(2) If the string contains the character '|' and does not contain the character ':', the string is denoted as a choice structure sequence by the relationship identifier *Ci*(i=1,2,...,k). *Ci* and its corresponding choice sequence (t_1|t_2|...|t_n) is added into the Hash table *ChoiceMap*. The hash table *ChoiceMap* is *key* =*Ci* and *value*= Hash (*key*)= "t_1|t_2|...|t_n";
(3) If the string contains the character '|' and character ':', the string is denoted as a selective structure sequence with choice conditions by the relationship identifier *Ci*(i=1,2,...,k). *Ci* and its corresponding choice sequence (t|t_2|...|t_n: c_j) are added into the Hash table *ChoiceMap*. The Hash table *ChoiceMap* is *key* = *Ci* and *value*= Hash (*key*)="(t_1|t_2|...|t_n : c_j)";
(4) If the string contains character 'L', the string is denoted as a loop structure sequence by the relationship identifier *Li*(i=1,2,...,k). *Li* and its corresponding loop sequence (L(t_1, t_2,..., t_n) : L_j) are added into the Hash table *LoopMap*. The hash table *LoopMap* is *key* = *Li* and *value*= Hash(key)= "(L(t_1, t_2,..., t_n) : L_j)";
(5) If the stack is empty, the character "/"will be appended onto the ending of the extracted string so as to distinguish each invocation sub-sequence of the web service. The invocation sequence is divided into several invocation sub-sequences of web services. The structural relationship among these invocation sub-sequences is the sequenced relation.

Step 3: Invoke web services according to the invocation sequence generated from step 2, and respectively execute each service structure and its corresponding internal web services.

Suppose the simplified invocation sequence is S', where i and j are integers and i is not equal to j. After traversing S', the relationship identifier R can be extracted in turn. Now invoke the web services as follows.

(1) If R∈{*ti*,i=1,2,...,n}, then according to the data association Hash table *IORelevancyMap*, the output p_j of web service t_j will be discovered, which is associated with the input p_i of web service t_i. Then the value of p_j can be obtained from the outcome results of t_j, which is the input parameter of t_i. Afterwards, according to the acquired interface of t_i, the web service t_i can be fully invoked. Meanwhile, the corresponding execution results will be saved.
(2) If R∈{*Ai*,i=1,2,...,n}, according to the Hash table *ConcurrentMap*, the value(string) corresponding to the key *Ai* can be acquired. Then the concurrence sequences will be parsed from the value, and the

ID	web service	Function	Input	Output
t1	AddService	Addition	a;b	e1
t2	SubstractService	Substraction	c;d	e2
t3	MultiplicationService	Multiplication	E1;E2	r
t5	PowerService	Square	r	r1
t6	AbsService	Absolute value	r	r4
t7	SinService	Sine	r1	r2
t8	CosService	Cosine	R2	r3
t9	SqrtService	Square root	r4	r5

Table 1: The details of computing service.

same operation will be repeated as in step 3. Meanwhile, each execution process should be placed into several concurrent threads.

(3) If R∈{*Ci*,i=1,2,...,n}, according to the Hash table *ChoiceMap*, the value (string) corresponding to the key *Ci* can be acquired. Then the selective sequences will be parsed from the values, which are S1',S2',... ,Sn'. First, according to the executable condition of selective sequences, the sequences that satisfy these conditions are added to a set (*selectedSet*). Second, it is necessary to judge whether *Ci* has selective conditions. If these exist, it should retrieve the published method of this selective condition, and then the selective result will be written into a set (*selectedConSet*). Finally, after the intersection between *selectedSet* and *selectedConSet*, the same operation will be repeated as in step 3. Meanwhile, each execution process should be placed into several concurrent threads.

(4) If R∈{*Li*,i=1,2,...,n}, according to Hash table *LoopMap*, the value (string) corresponding to the key *Li* can be acquired. Then the loop structure body and the loop condition will be parsed from the value. In the following process, the loop structure body will be executed according to step 3, and then the published method of this loop condition will be invoked. The Boolean value of results determines whether to re-execute the loop structure.

6 Experiment

We consider a composite web service of scientific computing as an example to specify the above invocation policy and execution.

There are eight web services in the composite web service of scientific computing (**Table 1**). The input e1 of web service t1 is equivalent to the output E1 of web service t3. The output e2 of web service t2 is equivalent to the input E2 of web service t3.The output r2 of web service t7 and the input R2 of web service t8 is a father-son relationship.

The Petri net model of the scientific computing web service is shown in **Figure 6**, and its invocation sequence is S= (t1&t2) t3 (t5 (t7t8:L1)|t6t9:C1).

In **Figure 6**, C1 is a selective condition of web service *PowerService* and web service *AbsServcie*. The corresponding method of C1 is *choice (double e1, double e2)* and its returned value is an integer, where e1 is an output of web service *AddService* and e2 is an output of web service *SubstractService*. L1 is a selective condition of web service *SinService* and web service *CosService*. The corresponding method of C1 is *is End (double p1, double r3)* and its returned value is a Boolean value, where p1 is the user's input parameter and r3 is an output of web service *CosService*.

From the scientific computing service Petri net, the structural relationship among the web services can be analyzed as follows: (1)<t1, t2>∈*Concurrence*; (2)<t5, t7>∈*Sequence*; <t7,t8>∈*Sequence*; <t6, t9>∈ *Sequence*; (3)<t5, t6>∈*Choice*.

The detailed experimental procedure is described as follows:

(1) Import a semantic file (Computing Service.owl) and the corresponding PNML+OWL document;
(2) Parse PNML+OWL and construct the data association hash table *IORelevancyMap* (**Table 2**),

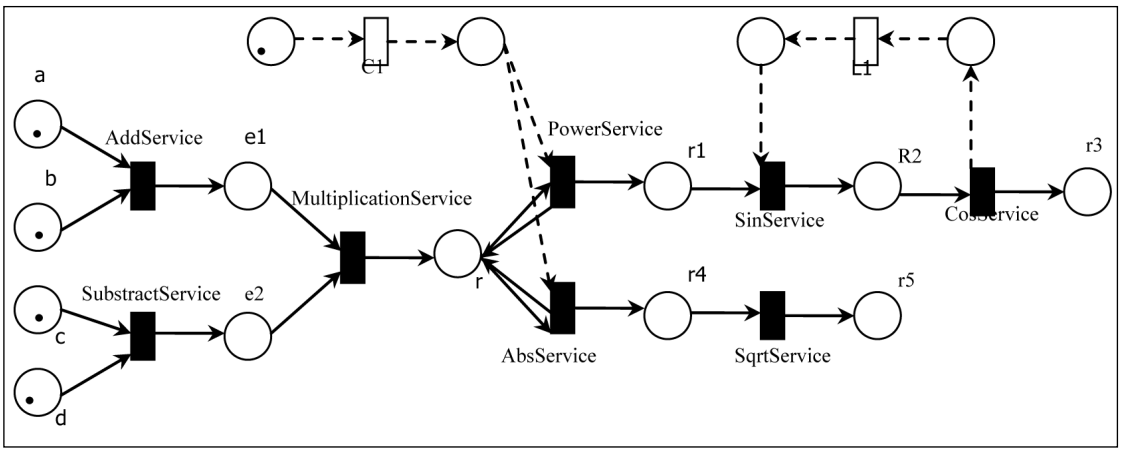

Figure 6: Scientific computing service Petri net.

ti:pi	t3:P6	t3:p7	t5:P9	t6: P11	T7: P13	t8: P15	t9: P17
tj:pj	t1:P2	T2: P5	t3: P8	t3:P8	T5: P10	t7: P14	t6: P12

Table 2: Data association hash table.

ID	P2	P5	P6	p7	P8	P9	P10	P11	P12	P13	P14	P15	P17
name	e1	e2	E1	E2	r	r	r1	R	r4	r1	r2	R2	r4

Table 3: The mapping table between ID and name.

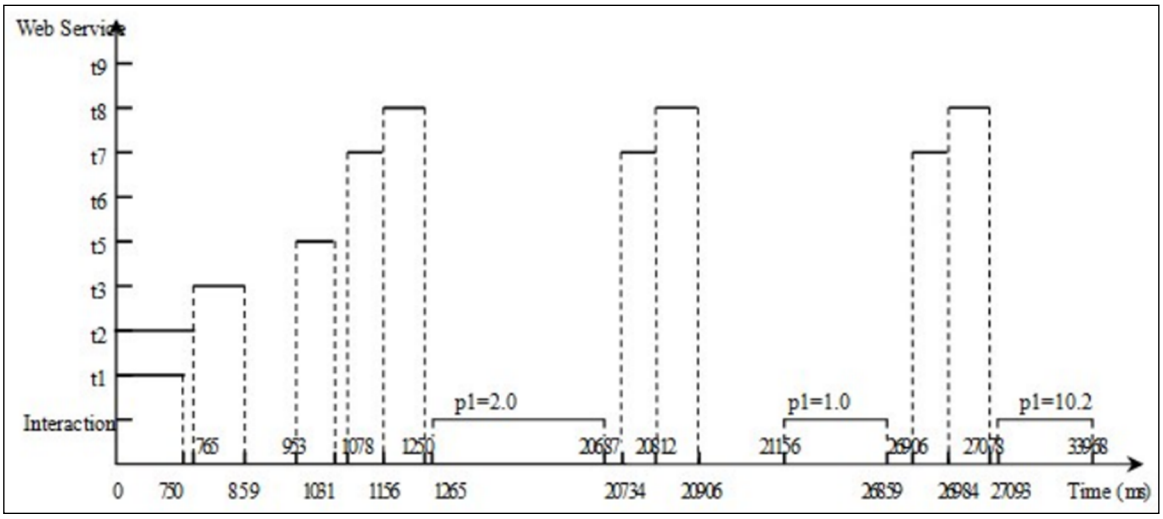

Figure 7: The web services execution time coordinates.

The mapping table between ID and name is as in **Table 3**;

(3) Get a simplified sequence S'=A1/t3C1/ by simplifying the invocation sequence of web service (S);
(4) According to the relationship identifier of S' and the corresponding invocation policies, invoke web services in turn. The execution order of web services is: t2->t1->t3->t5->t7->t8->L1->t7->t8->L1->t7->t8.

From the execution time coordinate picture of web services shown in **Figure 7**, the users interaction restricts the total execution time. The loop condition is executed 3 times, in which *SinService* (t7) and *CosService* (t8) are also invoked 3 times. *AbsService* (t6) and *SqrtService* (t9) are not invoked, which means that the selective structure has choose *PowerService* (t5) as the executable web service rather than *AbsService.* In addition, in the time slot 0–750*ms*, *SubstractServic*e (t2) and *AddService* (t1) are invoked during the same time period, which reflects the concurrence relationship between the web services.

service ID	Service name	Input name & value	Output name & value
t2	SubstractService	c(4.0) d(1.0)	e2(3.0)
t1	AddService	a(8.0) b(3.0)	e1(11.0)
t3	MultiplicationService	E1(11.0) E2(3.0)	r(33.0)
t5	PowerService	r(33.0)	r1(1089.0)
t7	SinService	r1(1089.0)	r2(0.9055)
t8	CosService	R2(0.9055)	r3(0.6172)
t7	SinService	r1(1089.0)	r2(0.9055)
t8	CosService	R2(0.9055)	r3(0.6172)
t7	SinService	r1(1089.0)	r2(0.9055)
t8	CosService	R2(0.9055)	r3(0.6172)

Table 4: The execution result table.

In conclusion, the whole execution of the scientific computing service fully corresponds to the execution flow of the invocation sequence of web services. The execution order and results are correct, and it reflects the structural relationship among the executable web services, which further validates the correctness and effectiveness of this method (**Table 4**).

7 Conclusion

In this paper, the invocation policies of web service composition have been concretely studied. Based on the Petri net of web service composition, the structural relationships are defined and analyzed. Then the corresponding invocation scheduling policies are proposed to describe different structural relationships. Finally, a web service composition execution algorithm is put forward based on Petri net, which can realize the orderly invocation of services within a web service composition. In a nutshell, this study is a good attempt to apply Petri net theory and its analysis methods to the execution of a web service composition.

Further research work may include:

(1) Extending the instance range to more applications so as to validate the effectiveness of this method and improve its performance.
(2) Based on the results of this paper and exceptions collected during the execution of web service composition, further study will focus on running fault detection and its analytical policies with Petri net.

8 Acknowledgements

This work was supported by National Natural Science Foundation of China (60903099).

9 References

Ai, L., Tang, M., & Fidge, C. (2011) Partitioning composite Web services for decentralized execution using a genetic algorithm penalty-based genetic algorithm. *Future Generation Computer Systems 27*(3), pp 157–172.

Cardinale, Y., & Rukoz, M. (2011) A framework for reliable execution of transactional composite Web services. *Proceedings of the International Conference on Management of Emergent Digital Ecosystems*, New York, USA, pp 129–136.

Chen, S., Feng, Z., & Wang, H. (2010) Service Relations and Its Application in Services Oriented Computing. *Chinese Journal of Computers 33*(11), pp 2068–2083.

Codehaus. xfire. Retrieved from the World Wide Web January 5, 2015: http://xfire.codehaus.org/.

Darmstadt, C. R., Kuntze, N., & Velikova, Z. (2009) Secure Web Service Workflow Execution. *Electronic Notes in Theoretical Computer Science (ENTCS) 236*, pp 33–46.

Ding, Z., Wang, J., & Jiang, C. (2008) An Approach for Synthesis Petri Nets for Modeling and Verifying Composite Web service. *Journal of Information Science and Engineering 24*(5), pp 1309–1328.

Jordan, D., & Evdemon, J. (2007) Web services Business Process Execution Language Version 2.0[DB/OL]. Retrieved from the World Wide Web January 5, 2015: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html,

Jüngel, E. K., & Weber, M. (2000) The Petri net markup language. *Petri Net Newsletter 59*, pp 24–29.

Kang, Z., Wang, H., & Hung, P. CK. (2007) WS-CDL+: An Extended WS-CDL Execution Engine for Web service Collaboration. In *IEEE International Conference on Web Services* (ICWS 2007).

Kang, Z., Wang, H., & Hung, P. CK. (2007) WS-CDL+ for Web service Collaboration. *Information Systems Frontiers 9*(4), pp 375–389.

Lapadula, A., Pugliese, R., & Tiezzi, F. (2007) A Calculus for Orchestration of Web Services. In *Programming Languages and Systems, LNCS, 4421*, pp 33–47.

Ma, B., Xiang, D., & Zhang, Z. (2013) Automatic Generation of Petri Net for Web services Composition. *Journal of Chinese Computer Systems 34*(2), pp 332–337.

Ma, B., & Xie, N. (2010) From OWL-S to PNML+OWL for Semantic Web Services. *Second International Conference on Computer Modeling and Simulation*, Sanya, China, pp 326–328.

Ma, B., & Xu, Y. (2009) Integrating PNML with OWL for Petri Nets. *2nd IEEE International Conference on Computer Science and Information Technology*, pp 228–230.

McGuinness, D., & van Harmelen, F. (2011) OWL Web Ontology Language Overview [DB/ OL]. Retrieved from the World Wide Web January 5, 2015: http://www.w3.org/TR/owl-features/

Mendes, R., & Paulo, F. P. (2009) WebFlowAH:An Environment for Ad-Hoc Specification and Execution of Web Services-based Processes. *Proceedings of the 2009 ACM symposium on Applied Computing*, New York, USA, pp 692–693.

Narendra, N. C., & Orriens, B. (2007) Modeling Web service composition and execution via a requirements-driven approach. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, Seoul, Korea, pp1642–1648.

Park, C., & Park, S. (2008) Efficient execution of composite Web services exchanging intentional data. *Information Sciences 178*(2), pp 317–339.

Suzumura, T., Trent, S., Tatsubori, M., Tozawa, A., & Onodera, T. (2008) Performance Comparison of Web Service Engines in PHP, Java and C. *IEEE International Conference on Web Service* (ICWS), pp 385 – 392.

Tan, W., Fan, Y., Zhou, M., & Tian, Z. (2010) Data-Driven Service Composition in Enterprise SOA Solutions: A Petri Net Approach. *IEEE Transactions on Automation Science and Engineering 7*(3), pp 686–694.

Tang, X., Jiang, C., Ding, Z., & Wang, C. (2007) A Petri Net-Based Semantic Web Service Automatic Composition Method. *Journal of Software 18*(12), pp 2991–3000.

Tang, X., Jiang, C., & Zhou, M. (2011) Automatic Web service composition based on Horn clauses and Petri nets. *Expert Systems with Applications 38*(10), pp 13024–13031.

Tsamoura, E., Gounaris, A., & Manolopoulos, Y. (2011) Decentralized execution of linear workflows over Web services. *FutureGenerComput System, 27*(3), pp 341–347.

Valero, V., Cambronero, M. E., Díaz, G., et al. (2009) Petri net approach for the design and analysis of Web Services Choreographies. *The Journal of Logic and Algebraic Programming 78*, pp 359–380.

Wang, Y., Dai, G., Hou, Y., Fang, J., & Ren, X. (2009) Verification of Web Service Orchestration Based on Concurrent Transaction Logic. *Chinese Journal of Electronics 37*(10), pp 2228–2233.

Xiang, D., Ma, B., & Zhang, Z. (2012) Automatic Sharing Synthesis of Petri Nets Based on Semantic. *Journal of System Simulation 24*(11), pp 2237–2242.

Xiong, P., Fan, Y., & Zhou, M. (2010) A Petri Net Approach to Analysis and Composition of Web Services. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 40*(2), pp 376–387.

Yu, W. (2007) Peer-to-Peer Execution of BPEL Processes. *The 19th International Conference on Advanced Information Systems Engineering*, CAISE:Trondheim, Norway.

How to cite this article: Xiang, D, Xie, N, Ma, B and Xu, K 2015 The Executable Invocation Policy of Web Services Composition With Petri Net. *Data Science Journal,* 14: 5, pp.1-15, DOI: http://dx.doi.org/10.5334/dsj-2015-005

Published: 22 May 2015

]u[*Data Science Journal* is a peer-reviewed open access journal published by Ubiquity Press.

OPEN ACCESS