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 \( \Sigma = (S, T, CS, CT, F, CF, M, L) \), 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. \( \tau \) 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 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 \(\Sigma = (S, T, F, M, L)\), 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. \(M: S \cup C \rightarrow \{0, 1, 2, \ldots\}\) is the number of tokens within places;
5. \(L: S \rightarrow D \{\tau\}\) is a semantic markup function, where \(D\) is a set of classes and instances from one domain ontology. \(\tau\) 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;
Service composition layer: web services are composed in this layer according to semantic associations;  
Service planning layer: the invocation sequence of web services is generated here based on the Petri net of the web service composition;  
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.
4.1 Prepositive web services and postpositive web services

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

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

**Conclusion 1** In web service composition for web service \( t_i \) and its prepositive service set \( \text{Pro} ( t_i ) \), if \( \forall t_j \in \text{Pro} ( t_i ) \) (\( i,j \in \{1,2,\ldots,|T|\}, j \neq i \)), then the input-output association between \( t_i \) and \( t_j \) can be discovered. Similarly, for the postpositive service set \( \text{Post} ( t_i ) \), if \( \forall t_j \in \text{Post} ( t_i ) \) (\( i,j \in \{1,2,\ldots,|T|\}, j \neq i \)), 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 (concurrency), 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 \( \Sigma = (S, T, F, M, L) \) with a basic structural relationship \( R \) belonging to {sequence, concurrency, choice}:

(1) **Sequenced Relationship**
For two web services \( t_i \) and \( t_j \), the postpositive web service of \( t_i \) is a nonempty set \( \text{Post} ( t_i ) \), where \( i,j \in \{1,2,\ldots,|T|\} \), and \( i \neq j \). If \( |\text{Post} ( t_i )| = 1 \) and \( t_j \in \text{Post} ( t_i ) \), then the relationship between \( t_i \) and \( t_j \) is a sequenced relationship (sequence), which can be denoted as \( \langle t_i, t_j \rangle \in \text{Sequence} \). If the relationships among \( t_1,t_2,\ldots,t_n \) belong to sequence successively, \( \langle t_1,t_2,\ldots,t_n \rangle \in \text{Sequence} \), the corresponding expressed sequence of this relationship is \( t_1t_2\ldots t_n \). This is shown as \( t_1 \) and \( t_2 \) in Figure 3a.

(2) **Concurrence Relationship**
For a web service \( t_i \), \( i \in \{1,2,\ldots,|T|\} \), the postpositive web service of \( t_i \) is a nonempty set \( \text{Post} ( t_i ) \). If \( |t_i^*| > 1 \), \( |\text{Post} ( t_i )| > 1 \), \( t_j \) and \( t_k \in \text{Post} ( t_i ) \), \( t_j^* = \emptyset \), then the relationship between \( t_j \) and \( t_k \) is a concurrence relationship (concurrency), which can be denoted as \( \langle t_j, t_k \rangle \in \text{Concurrence} \). Similarly, if there exists a web service \( t \) that satisfies \( t_1,t_2,\ldots,t_n \in \text{Post} ( t_i ) \) and with the condition that the relationship between the elements of the set \( t_1,t_2,\ldots,t_n \) is concurrency, then the corresponding expressed sequence is \( t_1^* \& t_2^* \& \ldots \& t_n^* \). This is shown as \( t_2 \) and \( t_3 \) in Figure 3b.
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 $t_i$, $i \in \{1,2,\ldots,|T|\}$, the postpositive web services of $t_i$ is a nonempty set $Post(t_i)$. If $|t_i^*|=1$, $Post(t_i)>1$, and $t_i$ and $t_u$ all belong to $Post(t_i)$, then the relation between $t_i$ and $t_u$ is the selective relationship (Choice), which can be denoted as $<t_i,t_u>\in Choice$. Similarly, if there exists a web service $t$ that satisfies $t_i,t_2,\ldots,t_n \in Post(t)$, with the condition that the relationship between the elements of the set $\{t_i,t_2,\ldots,t_n\}$ is a selective relationship, then the corresponding expressed sequence is $(t_1 |t_2 |\ldots |t_n)$. This is shown as $t_2$ and $t_3$ in Figure 3c.

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

If there exists a web service $t$ and many structural relationships in its postpositive service $Post(t)(Post(t) \neq \emptyset)$, 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 $\{t_1, t_2, \ldots, t_n\}$, and $k\in N$. According to the three kinds of basic structural relationships, the relational expression between $t_i$ and $Post(t)$ can be denoted as $G(t_i,Post(t))= t_iR(\{Post(t_i)\})$, $R\in \{Sequence, Concurrence, Choice\}$.

(1) If $<t_1, t_2, \ldots, t_n>\in Sequence$, according to theorem 1, definition 5, and the condition $|Post(t)|=1$, it can be concluded that $|Post(t)|=1$. If $t_u \in Post(t)$, then $G(t_u,Post(t))=t_u$;
(2) If $<t_1, t_2, \ldots, t_n>\in Concurrence$, then $G(t_u,Post(t))=t(t_1 \& t_2 \& \ldots \& t_n)$;
(3) If $<t_1, t_2, \ldots, t_n>\in Choice$, then $G(t_u,Post(t))=t(t_1 |t_2 |\ldots |t_n)$.

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 $L_1$, $t_1, t_2, \ldots, t_n$ are web services, and $i\in N$. $L_1$ refers to a control transition which controls(influences) the loop structure body $L(t_1, t_2, \ldots, t_n)$. In a loop structure body, there exist basic or nested structural relationships. In a Petri net with loop
structures, \( |L_i| = |L_j| = 1 \). Moreover, we regard the set \( \{ t_i \mid t_i \in \text{Post}(L_i) \} \) as the beginning service of the loop and the set \( \{ t_j \mid t_j \in \text{Prot}(L_i) \} \) as the ending service of the loop. The corresponding loop expression is represented as \( \text{Loop}(t_1, t_2, \ldots, t_n : L_i) = (L(t_1, t_2, \ldots, t_n) : L_i) \).

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

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 \( S = \text{Seq}(WS) = R \left( \bigcup_{i=1}^{m} \text{Seq}(WS_i) \right) \), where \( WS \) is a web service set \( \{ t_i, i = 1, 2, \ldots, n \} \) and \( WS_i \subseteq WS \). This satisfies \( \bigcup_{i=1}^{m} WS_i = WS \) and \( \bigcap_{i=1}^{m} WS_i = \emptyset \), where \( t_i \) is a web service, \( \text{Seq}(WS) \) 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 \( \text{Seq}(WS_i) = R(WS_i) \); otherwise, there must be a web service set \( WS_i' \) that can be covered by some relationship that satisfies \( iWS_i WS_i' \equiv R(WS_i') \). Then \( S_i = \text{Seq}(WS_i) = \text{Seq}(R(WS_i') \cup WS_i') \).

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

1) \( S_1 = \text{Seq}(t_1, t_2) = \text{Loop}(t_1, t_2 : L_1) = (t_1, t_2 : L_1) \);
2) \( S_2 = \text{Seq}(t_1, S_1) = \text{Sequence}(t_1, S_1) = t_1 S_1 \);
   \( \Rightarrow S_2 = t_1 t_1 \);
3) \( S_3 = \text{Seq}(S_2, t_3) = \text{Concurrent}(S_2, t_3) = (S_2 \& t_3) \);
   \( \Rightarrow S_3 = (S_2 \& t_3) = ((t_1, t_2 : L_1) \& t_3) \);
4) \( S_4 = \text{Seq}(S_3, t_4) = \text{Sequence}(S_3, t_4) = S_3 t_4 \);
   \( \Rightarrow S_4 = S_3 t_4 \);
5) \( S_5 = \text{Seq}(t_5, t_6) = \text{Choice}(t_5, t_6 : C_1) = (t_5 | t_6 : C_1) \);
6) \( S_6 = \text{Seq}(S_4, SS) = \text{Sequence}(S_4, SS) = S_4 S_4 \);
   \( \Rightarrow S_6 = S_4 S_4 = ((t_1, t_2 : L_1) \& t_3) t_5 (t_5, t_6 : C_1) \).

After iterating the above six formulas and replacing similar structural relationship expressions, the invocation sequence of the web services is \( S = \text{Seq}(t_1, t_2, t_3, t_4, t_5, t_6, t_7) = (t_1, t_2, t_3, t_4, t_5, t_6, t_7) \). In the invocation
sequence of the web services, the priority of structural relationships is regulated as: sequence > concurrency > 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 \( R = \{ t_i | A_i | C_i | L_i, i = 1, 2, ..., n \} \), where \( i \) is an integer, \( t_i \) represents an atomic web service, \( A_i \) represents the concurrency relationship, \( C_i \) represents the selective relationship, and \( L_i \) represents the loop relationship. The invocation scheduling for web services is given as follows:

1. Retrieve the input-output associations among the web services from PNML+OWL;
2. 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;
3. 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 \( t_{im}(k=1, 2, ..., n) \) 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 \( \bigcap_{m=1}^{n} t_{im} = \bigwedge_{m=1}^{n} \text{Pro}(t_{im}) = t_j \), where \( \text{Pre} (t_{im}) \) 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 \( S_j' \) 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 \( v_{p_j} \) only if the semantic association of \( v_{p_j} \) 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 \( S_j' \) has been chosen.

5.4 Invocation policy of structural relationship

Suppose a relationship identifier \( R_i \) exists, and its structural relationship is \( R(t_{1}, t_{2}, ..., t_{j}), j \in \mathbb{N} \). The invocation of web services can be treated as follows:

1. Atomic web services, that is \( R_i \in \{ t_i | i = 1, 2, ..., n \} \). Invoke each web service directly;
2. Concurrency structure relationship, that is \( R_i \in \{ A_i | i = 1, 2, ..., n \} \) and \( R(t_1, t_2, ..., t_j) = (t_1 \& t_2 \& ... \& t_j) \). Invoke web service \( t_1, t_2, ..., t_j \) in concurrent threads;
3. Selective structure relationship, that is \( R_i \in \{ C_i | i = 1, 2, ..., n \} \) and \( R(t_1, t_2, ..., t_j) = (t_1 | t_2 | ... | t_j) \). 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 \( R_i \in \{ C_i | i = 1, 2, ..., n \} \) and \( R(t_1, t_2, ..., t_j) = (t_1 | t_2 | ... | t_j: c_k) \), where \( c_k \) is a choice condition. Screen out set \( S \) of the web services from \( t_1, t_2, ..., t_j \) to \( c_k \) and then invoke each web service of \( S \) in concurrent threads;
5. Loop structure relationship, that is \( R_i \in \{ L_i | i = 1, 2, ..., n \} \) and \( R(t_1, t_2, ..., t_j) = (L(t_1, t_2, ..., t_j) : l_k) \), where \( L(t_1, t_2, ..., t_j) \) is its loop structure body. Invoke the web services of \( L(t_1, t_2, ..., t_j) \) first, and then execute...
In practice, there may be a nested structural relationship that contains multiple or different kinds of service structural relationships, just like \( R(t_1, t_2, \ldots, t_k) = (t_1 \& t_2 \& \ldots \& t_k, R) \), where \( R \) is a relationship identifier of the structural relationship \( R(t_1, t_2, \ldots, t_k) \). 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, \ldots, k). \( Ai \) and its corresponding concurrence sequence \( t_1 \& t_2 \& \ldots \& t_n \) are added into the Hash table ConcurrentMap. The Hash table ConcurrentMap is key = \( Ai \) and value = Hash(key) = “\( t_1 \& t_2 \& \ldots \& t_n \)”;
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, \ldots, k). \( Ci \) and its corresponding choice sequence \( t_i \& t_j \) are added into the Hash table ChoiceMap. The Hash table ChoiceMap is key = \( Ci \) and value = Hash(key) = “\( t_i \& t_j \)”;
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, \ldots, k). \( Ci \) and its corresponding choice sequence \( t_i \& t_j \) are added into the Hash table ChoiceMap. The Hash table ChoiceMap is key = \( Ci \) and value = Hash(key) = “\( t_i \& t_j \)”;
4. If the string contains character ‘\:', the string is denoted as a loop structure sequence by the relationship identifier \( Li \) (i=1, 2, \ldots, k). \( Li \) and its corresponding loop sequence \( L(t_1, t_2, \ldots, t_n) : L_j \) are added into the Hash table LoopMap. The Hash table LoopMap is key = \( Li \) and value = Hash(key) = “\( L(t_1, t_2, \ldots, t_n) : L_j \)”;
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 \( R \in \{Ai\} \), 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 \( p_i \) 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 \( t_j \), the web service \( t_i \) can be fully invoked. Meanwhile, the corresponding execution results will be saved.
2. If \( R \in \{Ai\} \), 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
same operation will be repeated as in step 3. Meanwhile, each execution process should be placed into several concurrent threads.

(3) If \( R \in \{ C_i, i=1,2,\ldots,n \} \), according to the Hash table \( \text{ChoiceMap} \), the value (string) corresponding to the key \( C_i \) can be acquired. Then the selective sequences will be parsed from the values, which are \( S_1', S_2', \ldots, S_n' \). First, according to the executable condition of selective sequences, the sequences that satisfy these conditions are added to a set (\( \text{selectedSet} \)). Second, it is necessary to judge whether \( C_i \) 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 (\( \text{selectedConSet} \)). Finally, after the intersection between \( \text{selectedSet} \) and \( \text{selectedConSet} \), the same operation will be repeated as in step 3. Meanwhile, each execution process should be placed into several concurrent threads.

(4) If \( R \in \{ L_i, i=1,2,\ldots,n \} \), according to Hash table \( \text{LoopMap} \), the value (string) corresponding to the key \( L_i \) 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 \( e_1 \) of web service \( t_1 \) is equivalent to the output \( E_1 \) of web service \( t_3 \). The output \( e_2 \) of web service \( t_2 \) is equivalent to the input \( E_2 \) of web service \( t_3 \). The output \( r_2 \) of web service \( t_7 \) and the input \( R_2 \) of web service \( t_8 \) 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 \( S = (t_1 \& t_2) t_3 (t_5 (t_7 \& t_8 : L_1) | t_6 \& t_9 : C_1) \).

### Table 1: The details of computing service.

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

In Figure 6, \( C_1 \) is a selective condition of web service \( \text{PowerService} \) and web service \( \text{AbsServcie} \). The corresponding method of \( C_1 \) is \( \text{choice (double e1, double e2)} \) and its returned value is an integer, where \( e_1 \) is an output of web service \( \text{AddService} \) and \( e_2 \) is an output of web service \( \text{SubtractService} \). \( L_1 \) is a selective condition of web service \( \text{SinService} \) and web service \( \text{CosService} \). The corresponding method of \( C_1 \) is \( \text{is End (double p1, double r3)} \) and its returned value is a Boolean value, where \( p_1 \) is the user’s input parameter and \( r_3 \) is an output of web service \( \text{CosService} \).

From the scientific computing service Petri net, the structural relationship among the web services can be analyzed as follows: \( (1)<t_1, t_2> \in \text{Concurrence}; (2)<t_5, t_7> \in \text{Sequence}; (3)<t_5, t_9> \in \text{Sequence}; (3)<t_9> \in \text{Choice} \).

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 \( \text{fORelevancyMap} \) (Table 2),
(3) Get a simplified sequence $S' = A_1/t_3C_1$/ 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: $t_2 \rightarrow t_1 \rightarrow t_3 \rightarrow t_5 \rightarrow t_7 \rightarrow t_8 \rightarrow L_1 \rightarrow t_7 \rightarrow t_8 \rightarrow L_1 \rightarrow t_7 \rightarrow t_8$.

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 $SinService$ ($t_7$) and $CosService$ ($t_8$) are also invoked 3 times. $AbsService$ ($t_6$) and $SqrtService$ ($t_9$) are not invoked, which means that the selective structure has choose $PowerService$ ($t_5$) as the executable web service rather than $AbsService$. In addition, in the time slot 0–750 ms, $SubstructService$ ($t_2$) and $AddService$ ($t_1$) are invoked during the same time period, which reflects the concurrence relationship between the web services.
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


