CPN Model Based Standard Feature Verification Method for REST Service Architecture

Jing Liu(✉), Zhen-Tian Liu, and Yu-Qiang Zhao

Inner Mongolia University, Hohhot 010021, China
liujing@imu.edu.cn

Abstract. The representational state transfer (REST) service architecture is widely used in large-scale and scalable distributed web systems. If the REST service architecture does not comply with its standard feature constrains, it can result in degraded performance or low scalability of the REST-based web systems. Therefore, in order to enhance the quality of system designing, it is necessary to verify whether the system design meets the standard feature constrains of the REST service architecture. In this paper, we propose a standard feature constrains verification method for REST service architecture based on Colored Petri Nets (CPN) model. Firstly, five standard feature constrains of the REST service architecture are modeled using the CPN. Then a verification method is proposed based on synchronized matching of the execution paths in model state space. Lastly, we validate the usability and validity of the proposed verification method using a practical course management web system based on the REST service architecture. Experimental results show that our method can effectively confirm whether the web application system design based on REST service architecture conforms to the standard feature constrains of the REST service architecture. Besides, it can also provide intuitive and feasible execution data when the standard feature constraints are not met, which can facilitate the defects location and correction of the following design of application systems.

Keywords: REST service architecture · colored Petri nets model · Verification of standard features

1 Introduction

In 2000, Fielding first proposed the representational state transfer (REST) service architecture [1]. It refers to a new style abstracted in combination with the design principles of the hyper text transfer protocol (HTTP) standard and the uniform resource identifier (URI) standard, which is resource-oriented and emphasizes resource-centricity. At present, REST service architecture is widely used in collaborative web systems. Many web application systems are developed based on REST service architecture, such as Amazon shopping website

H. Gao et al. (Eds.): CollaborateCom 2018, LNICST 268, pp. 688–707, 2019.
https://doi.org/10.1007/978-3-030-12981-1_48

and Google search engine [2]. The advantages of the REST service architecture include the use of a browser as a client to simplify software requirements, the use of caching mechanisms to increase access speed, the use of stateless communication to increase server scalability, and reducing the complexity of the web system development to improve the scalability.

Now, some web application systems applying REST service architecture do not comply the standard feature constraints of REST service architecture, thus causing many corresponding problems. For example, failure to meet stateless constraints will damage the scalability of the system, which will affect system load balancing. Failure to meet client-server constraints will increase the overhead of the system server and increase response time. Failure to meet cacheable constraints will result in more time overhead, because each time you retrieve a resource, you need to request it from the server [3]. Therefore, in order to prevent the above problems, when designing a web application system based on the REST service architecture, it is necessary to verify whether the system design meets the standard feature constrains of the REST service architecture. So that, we can improve the quality of R&D of web systems based on REST service architecture.

There are several formal methods to verify whether REST service architecture complys standard feature constrains, such as finite state machine (FSM) based method and communicating sequential processes (CSP) based method. The FSM-based method does not support tiering and cache acknowledgment, while the CSP-based method uses a higher degree of abstraction and a more complex description. Therefore, in view of the advantages of the colored Petri nets (CPN) model in visualization, hierarchical modeling, complex data description, concurrent behavior description and dynamic execution [4–6], we propose a CPN model based standard feature constrains verification method for REST service architecture. Firstly, CPN modeling is performed on the standard feature constraints of the REST service architecture, including client-server constrain, cacheable constrain, stateless constrain, layered constrain, and uniform interface constrain, etc. Secondly, the verification of path synchronization matching based on the model state space is implemented. That is, based on the CPN model of the application system and the CPN model of the standard feature constraints, the respective execution paths in the model state space are synchronously matched. If the path can be executed synchronously, the application system meets the REST standard feature constraints. Lastly, we use the course management web system based on REST service architecture as an example to verify the availability and effectiveness of the above verification methods. The experimental results show that the verification method we proposed can effectively confirm whether the web application system design based on REST service architecture conforms to the standard feature constraints of REST service architecture. In addition, this method also can provide intuitive and feasible execution data when the standard feature constraints are not met, so as to facilitate the defects location and correction of the following design of application systems.

This paper is organized as follows. Section 2 introduces related research. Section 3 gives the CPN modeling process of five standard feature constraints in the REST service architecture. Section 4 elaborates the core idea and concrete algorithm of the verification method based on the model state space execution path synchronization matching. Section 5 applies verification methods to specific web systems, so as to verify their availability and effectiveness. Section 6 concludes the paper.

2 Relevant Technology

2.1 REST Service Architecture

The REST architecture is resource-centric, so resources are the most critical abstraction in the REST architecture, and any unit can be a resource. At the same time, all resources must have a uniform resource identifier (URI), the operation of the resource does not change the resource identifier, and all operations are stateless. RESTful web services mainly use four methods in the HTTP protocol, including POST, GET, PUT, and DELET. Among them, the POST method is to add new data, that is, adding a resource without an ID. The GET method is to read the data and get an existing resource. The PUT method is to update the data, update a resource or add a resource without an ID, and replace the current state of the resource with the given representation information. The DELETE method is to delete a resource, and the deletion is idempotent which is an important attribute. Idempotent is that sending multiple requests has the same impact on resource status as sending one request [7].

The main elements of the REST architecture include data unit, connector, and component. The data unit is mainly composed of resources, resource identifiers, indicators, presentation metadata, resource metadata and control data. The connector acts as a unified interface for each component to communicate with each other and access resources. It is mainly composed of client, server, cache, parser and channel. The connector encapsulates the underlying implementation and communication mechanism of the resource. Components primarily include user agent, origin server, and intermediary, which are divided based on their role in the application.

2.2 CPN Method

We use CPN technology to model the REST service architecture. The experimental tool is CPN Tools, which can effectively support the verification analysis of the model. CPN is a formal modeling method that evolved from the traditional Petri Net, so we can say that CPN is a high-level Petri Net. CPN has strong mathematical modeling ability, which is closely related to mathematics. Besides, it is often used for modeling and analysis of complex and concurrent systems. The CPN modeling language has a mathematical definition that combines both grammar and semantics. The verification method involves mathematical formula attributes and computer-aided proofs, which are implemented by the model.

The CPN verification method we used includes the following two methods, simulation method and state space method. The simulation method verifies the state transition of each step in the system, and verifies whether the system model meets the expectations [8]. The state space verification method generates a corresponding state space after the system simulation is executed. The state space is the executable path of the system. It shows all the reachable state and state changes of the CPN model with a directed graph, where the node represents the state, the arc represents the occurred event. The state space can be automatically generated. Based on the above characteristics, we use the CPN method to model the REST service architecture application. By simulating the resource changes of the application service through state transitions, it can be effectively verified.

CPN is defined as a nine-tuple, CPN = (P, T, A, Σ, V, C, G, E, I) [9]. The definition of each element is shown in Table 1. And the CPN method we used is defined strictly in accordance with the standard.

Table 1. Definition of CPN elements

Name	Definition	Name	Definition
P	Set of places	T	Set of transitions
A	Set of arcs	Σ	Set of colour sets
V	Set of variables	C	Colour set function
G	Guard function	E	Arc expression function
I	Initialisation function		

2.3 Related Work

Based on the notion of MROP, Sergio et al. present a metamorphic testing approach for the automated detection of faults in RESTful Web APIs [13]. Irum et al. explore the usage of formal application of Event-B on the REST architectural style [15]. And they successfully address inconsistency design issue, model checking of service specifications and the state-explosion problem that may arise due to a large number of resources. Costa et al. proposed a method, tool and guiding principle for evaluating REST architecture [7]. They evaluated their method in the architecture trade off analysis method (ATMM) scenario, considering various attributes and verifying the correctness of the method. Xi et al. mainly used the CSP method to formally model the REST architecture [10]. Firstly, they analyzed the calling process of the resource, abstracting four main process components, including user agent, intermediary, origin server, and resource. Then they showed the communication of four processes. Secondly, they analyzed and validated the standard feature constraints of the REST service architecture based on the Process Analysis Toolkit (PAT) model verification tool and sequential logic description. Ting et al. also used CSP methods to model and analyze

the RESTful web services [11]. They abstracted the REST architecture model into three modules, client, server, and resource. Then the process in the REST architecture and its specific method behavior were described by CSP. Finally, the first-order logic was used to describe the stateless attributes of the web service and verified based on the PAT tool. In addition, Adhipta et al. used hierarchical CPN to simulate the behavior of the web services system model [12]. They performed functional verification based on the generated state space, proving that the web services extension model can improve the communication transmission performance of information. From the above research, we known that it is feasible to model, analyze and verify the web application system using the CPN model.

In summary, the current verification work on the standard features of the REST service architecture is mainly focused on the CSP-based method, but this type of verification method has a high degree of abstraction and a complicated description. Therefore, we make full use of the advantages of CPN modeling technology, and propose a standard feature verification method for REST service architecture based on CPN model. This method can effectively confirm whether the web application based on the REST service architecture conforms to the standard feature constraints of the REST service architecture. When the standard feature constraints are not met, it can also provide intuitive and feasible execution data, which can facilitate the defects location and correction of the following design of application systems.

3 Standard Feature Constraint Modeling of REST Service Architecture Based on CPN Model

3.1 Process View of the REST Service Architecture

The REST service architecture emphasizes the resource invocation process. Everything is resource-centric. Therefore, the process view of the REST service architecture is used to describe the interaction of REST architecture resources, showing the interaction between different components. Referring to Jensen et al., Fig. 1 shows an example of the interaction of the four main components in the REST service architecture process view.

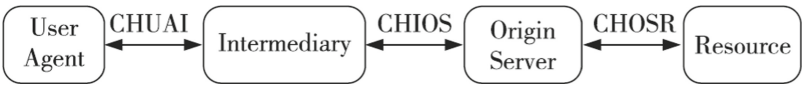

Fig. 1. Four main parts of the REST service architecture application model

The process view of the REST service architecture mainly includes four components, user agent, intermediary, origin server and resource. The interaction between components is through channels and interfaces, and all have

a unified interface. User agents, intermediary and origin server have an internal caching mechanism [8]. The channels between each other are described as CHIOS, CHOSR, and CHUAI, etc., which implement information transmission and reception.

Specifically, when a user agent (i.e., a client) sends a request message, it first requests its internal cache to get the resource. If the resource exists in its internal cache, it is directly returned to the user agent. If the resource does not exist in its internal cache, then a request message is sent to the external network intermediary. Here, the internal channel between the two is defined as CHUAI. The request message can be described as a four-tuple (GET, PUT, POST, DELETE) [14], and all messages are represented in the same four-tuple form. The specific meaning of the elements in the four-tuple is described in Sect. 2.1. Intermediary is mainly composed of proxy or gateway and also has caching function. The main purpose of intermediary is to forward messages and transfer information. The user agent sends the request information to the intermediary, and the intermediary has two ways to query the resources. One is to request resources from its internal cache, and the other one is to request resources from the origin server, where it communicates via the internal channel CHIOS. The resources required by the user agent are stored on the origin server, which is the only component that can communicate with the resource. The user agent can communicate with the origin server through the intermediary. After receiving the request resource information, the origin server checks its cache. If the cache does not have the resource, the origin server sends the request information to the resource and returns the requested resource to the intermediary. Communication between the origin server and the resource takes place via the CHOSR channel.

3.2 CPN-Based Standard Feature Constraint Modeling

Client-Server Constrain. The user agent and origin server act as two separate components that are separated by a unified interface. The client-server constraint applies the C/S architecture feature to standardize the REST architecture. REST is a typical C/S architecture, which guarantees that communication can only be initiated unilaterally by the client in the form of request - response. The REST architecture emphasizes the thin server, the server only handles resource-related operations, and all display work should be on the client. The user agent sends request resource information to its internal cache or origin server, ignoring the absence of resources. The information requested by the user agent must exist in its internal cache or origin server, and all request messages sent by the user agent will be replied to by its cache or origin server. Figure 2 shows the CPN model of the client-server standard feature constraints in the REST service architecture. Firstly, the client sends a request for resource information msg, which is described as a variable of type Msg 1, Msg 1 is described as a record type in CPN modeling:

colset Msg1=record m:MSG * f:FORMAT * s:SENDER * r:RECEIVER;

The four elements are all string types, the MSG describes the content of the request resource, the FORMAT describes the format of the request resource,

the SENDER describes the sender of the request message, and the RECEIVER describes the sender of the response message. Secondly, we should query whether the resource information exists in the client's internal cache. If the resource is in its cache (exemplified by the first element component m = "1" in expression on the output arc of the RequestUC transition), the response information is returned by the cache to the user agent. If the request resource does not exist in the cache (exemplified by the first element component m = "no" in expression on the output arc of the RequestUC transition), then the resource needs to be requested again from the origin server. Finally, the resource information requested by the client is returned to the user agent.

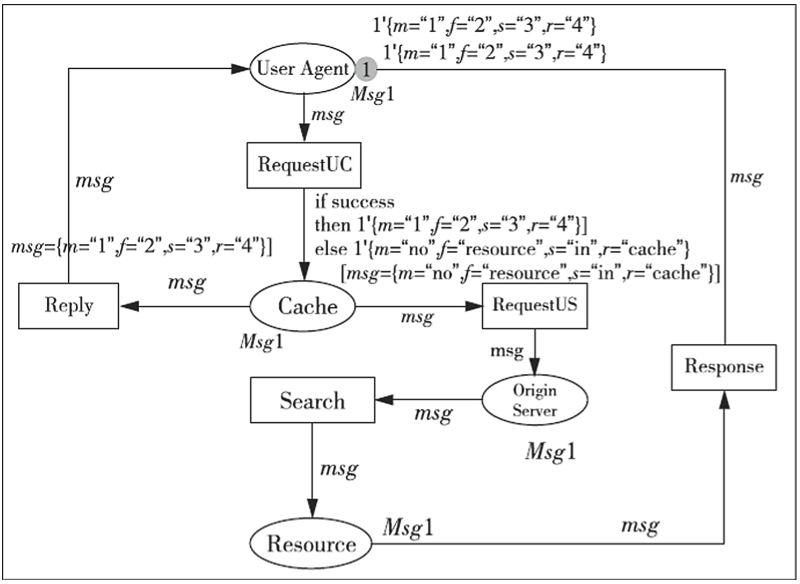

Fig. 2. CPN model with client-server constraints

Cacheable Constrain. User agent, intermediary and origin server all have a caching mechanism, and request resource information can be stored in the buffer, which can improve the inefficiency caused by stateless features. Cachable constraints are used to verify whether the user agent, intermediary and origin server all have caching capabilities, and response information can be retained in these three internal caches. Cacheable attributes effectively improve access performance, reduce system response time, and facilitate user agent. In this paper we only consider cacheable request information. If the user agent sends the request information for the first time, there is no information in the cache, so the resource information should be requested from the origin server. Otherwise, the user agent gets the resources directly from its internal cache without having to access the server. If the two request resource information functions are the same, the second one can directly access the cache to obtain resource information.

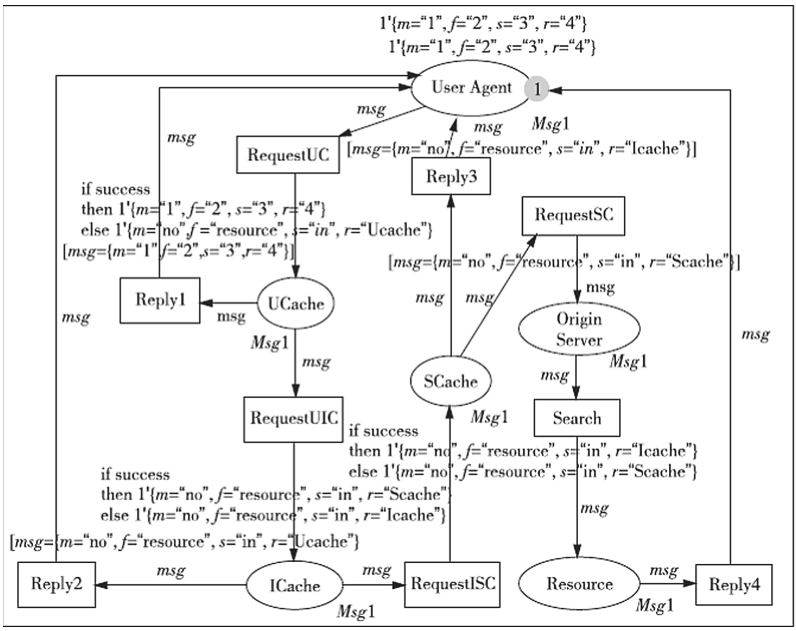

Fig. 3. CPN model with cachable constraints

Figure 3 shows the cacheable standard feature constraint model for the REST service architecture. Firstly, the user agent sends request resource information, and the request resource information msg is described as above. Secondly, it should query whether the resource information exists in the cache of the user agent. If the resource exists in its internal cache (exemplified by the first element component m = "1" in expression on the output arc of the RequestUC transition, namely RequestUserAgentCache), the information is returned directly to the user agent by its internal cache. If the request resource does not exist in the internal cache of the user agent (exemplified by the first element component m = "no" in expression on the output arc of the RequestUC transition), then the request information needs to continue to be sent to the network intermediary. If the resource information stored in its cache (exemplified by the fourth element component r = "Ucache" in expression on the output arc of the RequestUIC transition, namely RequestUserIntermediaryCache.), it is returned to the user agent. If the request resource information does not exist in the intermediary cache (exemplified by the fourth element component r = "Icache" in expression on the output arc of the RequsetUIC transition), the request information needs to be sent to the origin server. If the resource information exists in the origin server cache(exemplified by the fourth element component r = "Icache" in expression on the input arc of the RequestSC transition, namely RequestUserAgentServer-Cache), it is returned to the user agent. Finally, if the resource information does not exist in the origin server cache (exemplified by the fourth element compo-

nent r = "Scache" in expression on the input arc of the RequestSC transition), it is necessary to request information from the data resource, and finally return a response message to the user agent.

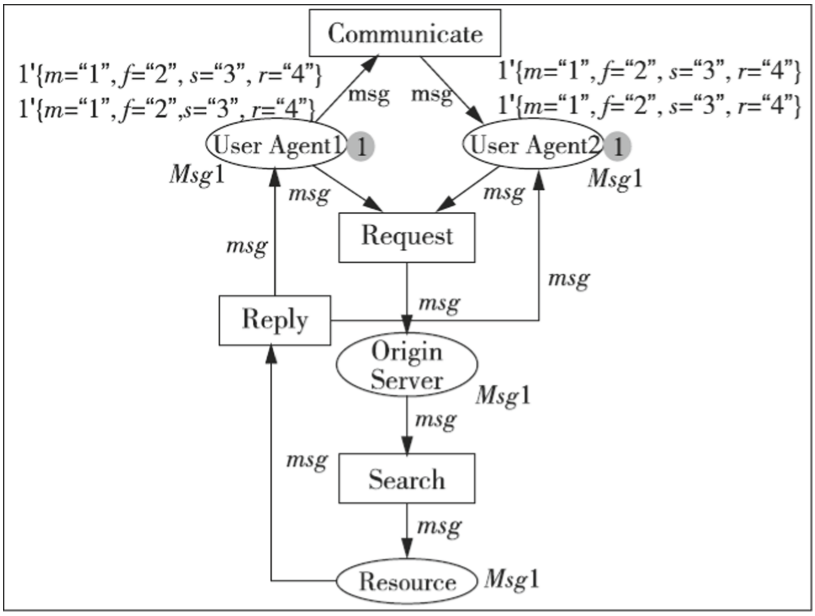

Fig. 4. CPN model with stateless constraints

Stateless Constrain. The server in the REST service architecture does not store any session state information with the client. All status information only exists in the communication message, that is, the request message sent by the user agent to the server must contain all information to understand the message. Stateless means that session state information remains in the user agent and the server is stateless. If the request information comes from different user agents at the same time, but the request resource information of the multiple user agents is the same, the origin server feeds back the same resource information to the user agents. The state of the REST service architecture refers to the state of the client. The representation of each resource on the client is a state of the client. The status information can be described as a triplet (id, data, oper), id is the id of the resource identifier, data is the content of the information, and oper is the specific operation, such as GET, PUT, POST, and DELETE.

Figure 4 shows the stateless standard feature constraint model of the REST architecture. First, the communication content between the two user agents is included in the request information, and the user agent only saves the communication content of both. Second, the user agent sends the request resource information to the origin server through the channel Request, and then returns

the requested resource to the user agent. If the resource information requested by multiple user agents is the same, the response information obtained must also be consistent (the response information obtained by the two user agents is {m = "1", f = "2", s = "3", r = "4"}), indicating that the REST architecture server is stateless.

Layered Constrain. The layered constraint mainly checks whether the REST service architecture application conforms to the three-tier architecture. Layered constraints increase the independence between the layers and improve the scalability of the system. In a REST service architecture web application system, a component can only communicate with its neighboring components. That is, if the user agent wants to send the request information, it can only send it to its internal cache or intermediary, and cannot directly send it to the origin server. Similarly, the origin server cannot directly return a response message to the user agent. This constraint breaks down the REST architecture into layers of several levels. The mutual communication between other components is also the same. In the layered constraint, only the three-tier REST application system is considered in the REST architecture, namely the user agent layer, the middleware layer and the server layer.

Figure 5 shows the layered standard feature constraint model of the REST architecture. Firstly, the REST architecture is a standard three-tier architecture application. Secondly, the user agent can only send the request resource information to the intermediary (the interaction between the user agent and the intermediary is performed through the channel CHUAI), and then the intermediary continues to send the request information to the origin server (the interaction between the intermediary and the origin server is performed through the channel CHIOS), and finally returns the requested resource information to the user agent through the above interface. Among them, the three components are divided into three layers by the channel CHUAI and the channel CHIOS.

Uniform Interface Constrain. The unified structure constraint guarantees the consistency of the message format between component communication, that is, checking whether the interface of the REST architecture application is consistent, and improving the interactivity and reusability of the system. In this constraint, the request message sent by the user agent and the message format returned by the origin server must be consistent, that is, the identifiers of the resources are consistent. The request message is described as a quad representation $\{\text{msg}, \text{r\_format}, \text{sender}, \text{receiver}\}$, where msg represents the requested resource, r_format represents the format of the resource, sender represents the request sender of the resource, and receiver represents the request recipient of the resource.

Figure 6 shows the unified interface standard feature constraint model for the REST architecture. The request information sent by the user agent to the origin server is consistent with the format of the response information of the server (exemplified by the first element component m = "1" in expression on the

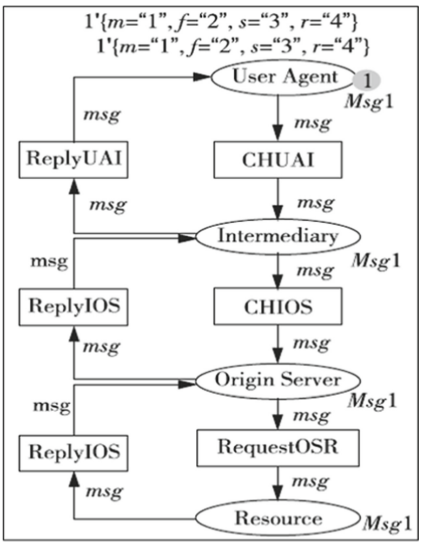

Fig. 5. CPN model with layered constraints

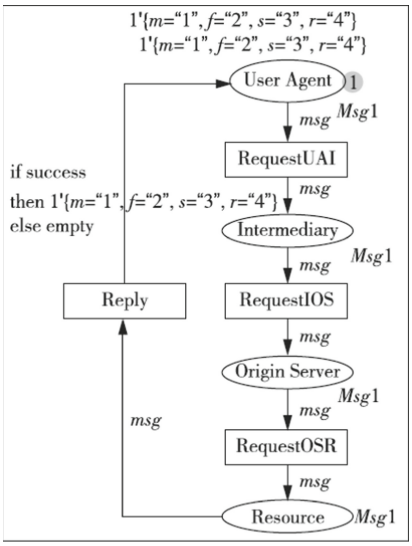

Fig. 6. CPN model with uniform interface constraints

output arc of the Reply transition). The information of the sender and receiver should focus on the same resource identifier, which reflects the unified interface standard features of the REST architecture.

4 REST Service Architecture Standard Feature Verification Method

This section proposes a standard feature verification method for REST service architecture. Based on the CPN model of the application system and the CPN model with standard feature constraints, we perform synchronous matching on the respective execution paths in the model state space. If the path can be executed synchronously, the application system meets the REST standard feature constraint. If the standard feature constraints are not met, the intuitive and feasible execution data is provided, which can facilitate the defects location and correction of the following design of application systems.

4.1 Core Idea of the Verification Method

The core idea of the verification method proposed in this paper can be summarized as constructing the CPN model of the application system and the CPN model of the standard feature constraint, respectively generating their respective state spaces, and obtaining the corresponding execution path sets of each model. We perform a synchronous path matching algorithm based on the model

state space, and check whether the state space execution path of the application system model synchronizes the path containing the standard feature constraint model. If it is included, the verification is successful, otherwise there is a non-conformity. By using the state difference set to locate the application system model and the defect errors in the system design, the non-conformity status and path are found, and the modification and perfection are made to meet the REST service architecture standard feature constraints.

Specifically, performing a synchronous path matching algorithm based on a model state space is the core. First, we give a key concept, the synchronization node. Assume that a node of the REST service architecture standard feature constraint model state space is identified by M*, and a node of the application system model state space is identified by M. If the Marking value of M* is equivalent to the function of M, that is, the key segment data in the two node tags is the same, and the functions of the subsequent executable transition description are also the same, then the two nodes can be called a synchronization node. The synchronization of the state space execution path includes the relationship of the synchronization node and the transition path. And the execution order of the synchronization node and the transition path is also consistent and orderly. The key part of the algorithm is that the state space execution path completes the synchronous execution matching. Intuitively, from the perspective analysis of the state space execution path generation, the possible enabled transition is fired from the initial state of the application system model. After one or more transition fired, a synchronization node that can be executed simultaneously with the start node of the standard feature constraint model is obtained. Then, the synchronization node is fired. After performing one step operation, one or more transition may be fired, and again get the next synchronization node that can be executed simultaneously with the starting node of the standard feature constraint model. Matching such synchronization nodes in turn, and firing corresponding enabled transitions until reaching the termination node of the standard feature constraint model, the synchronization matching method ends. At this point, it indicates that the application system meets the feature constraint of the standard REST service architecture. If the synchronization matching fails in any intermediate node, it indicates that the application does not meet the feature constraints of the standard REST architecture. Finding non-compliant features based on the REST service architecture application system can be achieved by looking up the difference set of the path set in the state space. By comparing the path generated by the application system model state space with the path generated by the corresponding standard feature constraint model state space, it is judged which path node is terminated. Then the token information reflected by the node identifier can provide intuitive application system execution data. That is, when the application system runs to this state, there is an execution state data that does not meet the constraint, which is used to help the system designer complete the positioning and correction of the application system design defects to meet the standard REST service architecture constraints.

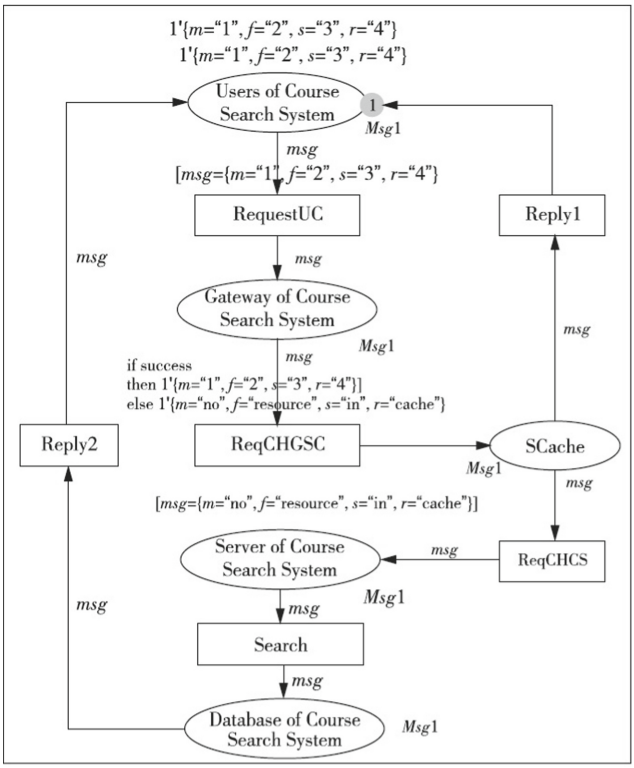

Fig. 7. CPN model for the course query subfunction

4.2 Algorithm Description of the Verification Method

Assume: Nodes* ={M0, M1, ..., Mm}, Arcs* ={A0, A1, ..., Am}

$$\text{Nodes} = \{\text{N0}, \text{N1}, ..., \text{Nn}\}, \text{Arcs} = \{\text{B0}, \text{B1}, ..., \text{Bn}\}$$

Nodes* describes the nodes and path sets of the standard feature constraint model state space, and the Nodes describes the nodes and path sets of the application system model state space. The core of the algorithm can be described as follows. Firstly, the standard model is matched within the node and path range of the application system model, and the initial synchronization nodes of the two models are found(shown in the ①). Then, finding the successor synchronization nodes of the two models. If found, the transitions of the two models each fired to start the next search (shown in the ②). If not found, the application system model continues to fired the transition, looking for a synchronization node that matches the standard feature constraint model (shown in the ③). Finally, if the synchronization node is successfully matched to the terminate synchronization node, the verification is successful, otherwise the verification fails (shown in the ④).

Algorithm 1. SynchronizedPathMatch

```
function SynchronizedPathMatch

    int m,n,i,j,k;

  if ((M0⊆Nodes)∧(A0⊆Arcs)) then
      Model can be synchronized;
    ①
    while i<n do
      if (Ni==M0 and Bi==A0) then
        Find the initial synchronization node;
      else
        i++;
      end if
    end while

      j =i+1;
      k =1;
    ②
    while (j≤n and k≤m)  do
      if (Nj==Mk and Bj==Ak) then
    {
        Find the next synchronization node;
        k++;
        j++;
    }
      else
        ③
        while (j≤n) do
          if (Nj==Mk and Bj==Ak) then
           Find the synchronization node;
          else
           j++;
          end if
        end while
      end if
    end while
    ④
    if (k==m) then
      return true
    else
      return false
    end if
  end if
end function
```

5 The Example of the Standard Feature Verification Method for REST Service Architecture

This section takes the course management web system based on the REST service architecture as an example to verify the availability and effectiveness of the verification method given in the previous section. It is confirmed that the verification method we proposed can be used to verify whether the web application system design based on the REST service architecture conforms to the standard feature constraints of the REST service architecture.

5.1 Web Application System Modeling Based on REST Service Architecture

This section conducts CPN modeling on the subfunctions of the query course in a course management web system. The constructed CPN model of the application system focuses on resource interaction and communication in the process of querying the course. As shown in Fig. 7, the communication behavior between the course query client, the course query system gateway, the course query system server, and the course query system database is modeled and analyzed. Then we use the verification method based on the model state space to perform path synchronization matching. So that we can confirm whether the model meets the five standard feature constraints of the REST service architecture.

In the model, the query request resource information msg is sent by the user of the course query system, and the data of the request information msg is defined as: colset Msg1 = record m: MSG * f:FORMAT * s:SENDER * r:RECEIVER, where MSG, FORMAT, SENDER and RECEIVER are both string types. The request information is transmitted to the gateway of the course inquiry system through the channel ReqCHUG, and then the gateway transmits the request information to the cache of the course query system server (Server) through the channel ReqCHGSC. If the request resource information is cached in the SCache of the server (exemplified by the first element component m = "1" in expression on the output arc of the ReqCHGSC transition). the server directly returns a message to the user of the course query system through the channel Reply1. If the course query system server does not have the requested resource in the cache (exemplified by the first element component m = "0" in expression on the output arc of the ReqCHGSC transition), the request message is sent through the channel Search to the database of the course query web system. Then the request resource information is returned from the database to the system user. Figure 8 shows the state space corresponding to the CPN model of the course query subfunction.

5.2 Standard Feature Constraint Verification Example and Analysis

Based on the CPN model of the course query subfunction given in Fig. 7, it is tested by the five REST service architecture standard feature constraints in

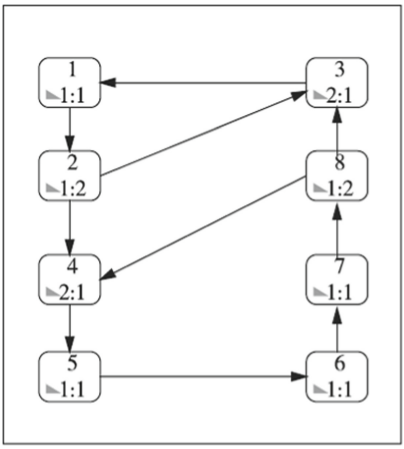

Fig. 8. State space of the course query subfunction model

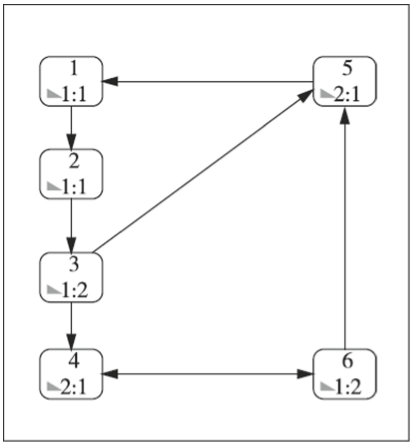

Fig. 9. State space of standard client-server constraints

Sect. 4.2 to verify whether the course query subfunction web application is a standard REST architecture application.

(1) Checking the standard feature constraint of the client-server and comparing it with the state space execution path of the standard client-server constraint in Sect. 3.2. we can find that the REST architecture model does not meet the client-server constraints. The user function of the course query system is equivalent to the function of the user agent in the standard REST architecture model. The gateway function of the course query system is equivalent to the intermediary in the standard REST architecture model. The server function of the course query system is equivalent to The functionality of the origin server in the standard REST architecture model, and the database function of the course query system is equivalent to the functionality of the resources in the standard REST architecture model. After verification by the synchronous path matching algorithm, the node 1 in Fig. 8 and the node 1 in Fig. 9 are initial synchronization nodes. Then through the transition path, we find that the REST architecture application model lacks the caching capabilities of its client. When obtaining the request information, the system first checks whether the cache exists. If it does not exist, continue to request the resource information from the origin server. Therefore, the course query subfunction web system should add client caching function.

(2) Verifying the cacheable standard feature constraints, and comparing it with the state space execution path of the standard cacheable constraint model in Sect. 3.2. we can find that the REST architecture model does not meet the cacheable constraint. After verification by the synchronous path matching algorithm, it can be found that the node 1 in Fig. 8 and the node 1 in Fig. 10 are initial synchronization nodes. From the subsequent transition path, we find that the client and the gateway of the course query system lack the

caching function. Therefore, the course query subfunction web system should add a corresponding caching mechanism.

(3) The layered standard feature constraint is tested. By performing path verification on its state space, we can find that the REST architecture model satisfies the layered constraint. Through the comparison with the state space execution path of the standard layered constraint of Sect. 3.2, the course query system users can only interact through the gateway of the course query system, and the database of the course query system can only communicate with the server. The subfunction web system meets the layered standard feature constraints. At the same time, the stateless and unified interface standard feature constraints are tested. By comparing the state space execution path of the standard stateless constraint in Sect. 3.2, we find that the course query system of the REST architecture meets the stateless constraint. All information about the message can be included in the request and response. The main content is stored in the server, and the server is stateless. In addition, we compare the state space execution path of the standard unified interface constraint of Sect. 3.2, and find that the model meets the unified interface constraint. The message format of the request and response is the same, both in the form of msg. When the same request information is sent twice, the response information is the same, so the uniform interface constraint is met.

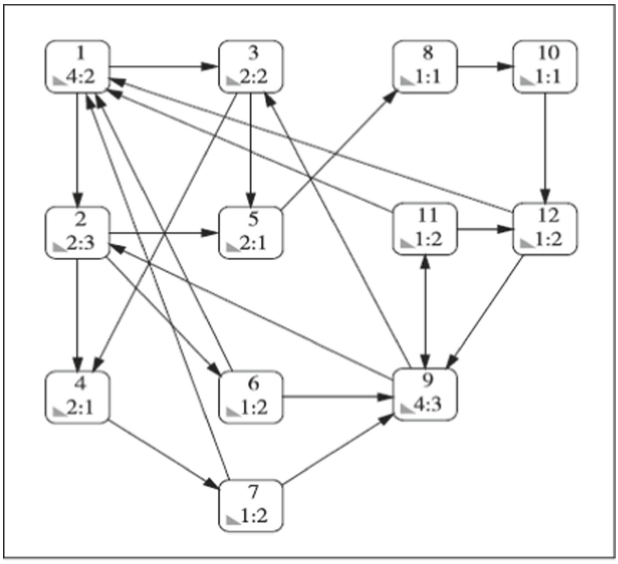

Fig. 10. State space of standard cachable constraints

Through the above analysis, Fig. 11 shows the modified CPN model of the course query subfunction application system based on REST service architecture.

After using the method we proposed, the application system meets the standard feature constraints of the REST service architecture. As can be seen from the comparative analysis of Figs. 7 and 11:

(1) Figure 11 adds the Users Cache function to Fig. 7, which can add a cache function to the client of the course management web system.
(2) Figure 11 adds the Gateway Cache feature to Fig. 7, which adds a cache function to the gateway of the course management web system.

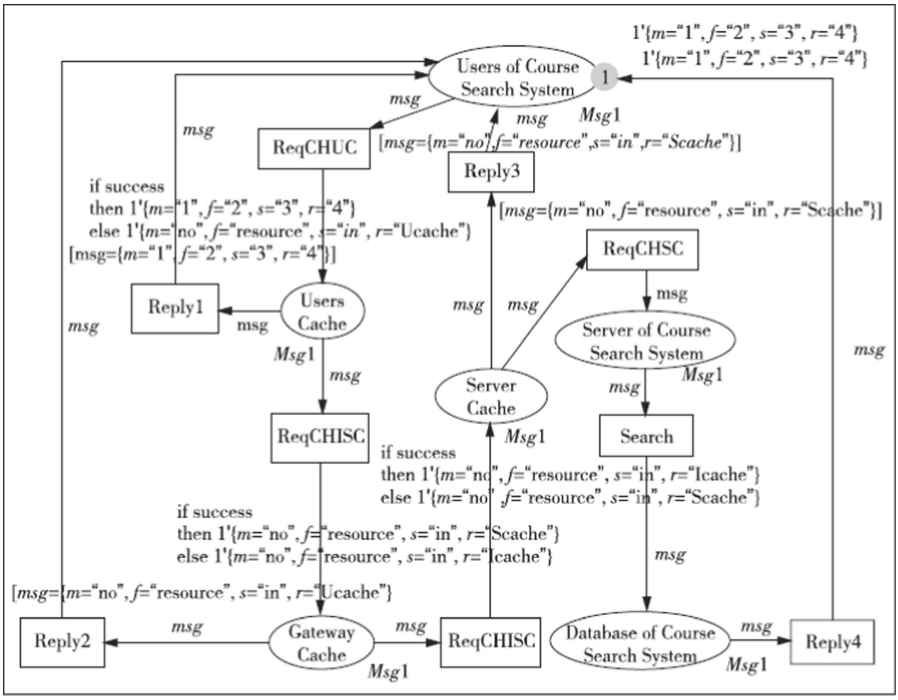

Fig. 11. Corrected CPN model of the course query subfunction

Adding the above two buffers can make the system more complete. When the client of the course management web system sends the request information, it first checks whether the resource exists in its internal cache. If the resource exists in its cache (exemplified by the fourth element component r = "4" in the expression on output arc of the ReqCHUC transition), the response message is returned directly to the client. If there is no client cache (exemplified by the fourth element component r = "Ucache" in the expression on output arc of the ReqCHUC transition), the resource needs to be requested from the gateway of the course management web system. If the resource is not in the client cache (exemplified by the fourth element component r = "Ucache" in the expression on

output arc of the ReqCHUC transition), the resource needs to be requested from the gateway of the course management web system. If the resource exists in the gateway's cache (exemplified by the fourth element component r = "Ucache" in the expression on output arc of the ReqCHUG transition), the response message is directly returned to the client.

6 Conclusion

Before implementing a web system based on the REST service architecture, it is crucial to verify whether the system design meets the standard features of the REST service architecture, and effectively improve the quality of the development of the web system based on the REST service architecture. We propose a standard feature verification method for REST service architecture based on CPN model. Firstly, the CPN model description of five standard feature constraints of REST service architecture is given. Next, we propose a verification method based on model state space to perform path synchronization matching. Based on the CPN model of the application system and the CPN model with standard feature constraints, the respective execution paths in the model state space are synchronously matched. If the paths can be executed synchronously, the application system meets the REST standard feature constraints. Finally, taking the course query function system based on REST service architecture as an example, the availability of the above verification method is confirmed. Applying the verification method we proposed can effectively confirm whether the web application system design based on REST service architecture conforms to the standard feature constraints of REST service architecture. When the standard feature constraints are not met, we provide intuitive and feasible execution data to facilitate the defects location and correction of the following design of application systems. The next step is to strengthen the formal description and validation of the core process of the verification method, and apply this method to practical objects with more typical application systems based on REST service architecture, so that we can refine execution details and improve the availability of methods.

Acknowledgment. This work was supported in part by the National Natural Science Foundation of Chain (No. 61662051, No. 61262017).

References

1. Fielding, R.T.: Architectural styles and the design of network-based software architectures, p. 303. University of California, Irvine (2000)
2. Paganelli, F., Turchi, S., Giuli, D.: A web of things framework for RESTful applications and its experimentation in a smart city. IEEE Syst. J. **10**(4), 1412–1423 (2017)
3. Song, Y., Xu, K., Liu, K.: Research on web instant messaging using REST web service. In: IEEE Symposium on Web Society, pp. 497–500 (2010)

4. Liu, J., Ye, X., Zhou, J.: Colored Petri net hierarchical model of complex network software and model integration verification method. High-Tech Commun. **23**(11), 1139–1147 (2013)
5. Benabdelhafid, M.S., Boufaida, M.: Toward a better interoperability of enterprise information systems: a CPNs and timed CPNs -based web service interoperability verification in a choreography. Procedia Technol. **16**, 269–278 (2014)
6. Sun, L.: Dynamic composition modeling and validation OD web services based on hierarchical colored Petri nets. China University of Petroleum, Dongying (2011)
7. Costa, B., Pires, P.F., Merson, P.: Evaluating REST architectures-approach, tooling and guidelines. J. Syst. Softw. **112**, 156–180 (2016)
8. Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri nets and CPN tools for modeling and validation of concurrent systems. Int. J. Softw. Tools Technol. Transf. **9**(3–4), 213–254 (2007)
9. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modeling and Validation of Concurrent System, pp. 95–188. Springer, Berlin (2009). https://doi.org/10.1007/b95112
10. Wu, X., Zhu, H.: Formalization and analysis of the REST architecture from the process algebra perspective. Future Gen. Comput. Syst. **56**, 153–168 (2016)
11. Ting, Y.: Formal Modeling and Analysis of RESTful Web Services. East Chain Normal University, Shanghai (2015)
12. Adhipta, D., Hassan, M.F., Mahmood, A.K.: Web services extension model simulation in hierarchical colored Petri net. In: International Conference on Computer & Information Science, pp. 741–746. IEEE (2012)
13. Segura, S., Parejo, J.A., Troya, J., et al.: Metamorphic testing of RESTful web APIs. IEEE Trans. Softw. Eng. **PP**(99), 1 (2017)
14. Garriga, M., Mateos, C., Flores, A., et al.: RESTful service composition at a glance. J. Netw. Comput. Appl. **60**(C), 32–53 (2016)
15. Rauf, I., Vistbakka, I., Troubitsyna, E.: Formal verification of stateful services with REST APIs using event-B. In: IEEE International Conference on Web Services, pp. 131–138. IEEE Computer Society (2018)