Intelligent Automation And Soft Computing, 2019

Vol. 25, no. 3, 573–583
https://doi.org/10.31209/2019.100000112

The Design and Implementation of a Service Composition System Based on a RESTful API

Wang Hui[1], Sun Guang-Yu[2,5], Zhang Qin-Yan[2], Liu Kai-Min[3], Xi Meng[3], Zhang Yuan-Yuan[4]

[1] Inner Mongolia Medical University, Hohhot 010050
[2] Zhejiang University, Hangzhou 310027
[3] College of Computer Science and Technology, Zhejiang University, Hangzhou 310027
[4] College of Information Technology, Zhejiang Chinese Medical University, Hangzhou 310053
[5] Binhai Industrial Technology Research Institute of Zhejiang University, Tianjin, 300301

ABSTRACT
With the current explosion of mobile applications and smart devices, more organizations are beginning to expose Web APIs, which makes APIs more widely used. How can these APIs be managed and utilized safely and effectively for businesses? It is not easy to say. Today's Web services mainly include traditional structured WSDL and unstructured RESTful. A RESTful architecture can effectively constrain and help to achieve a simpler, lighter, and more scalable system. How to uniformly organize and merge RESTful APIs is also a problem to be solved. To solve the above problems, this article has designed an API management system that can realize API service composition. This system brings together the open-source project WSO2 API Manager and the RESTful API service composition model. Based on the platform, developers can perform service management and service combination more efficiently and conveniently.

KEY WORDS: Open API, REST architecture, RESTful API, API manager, service combination.

1 INTRODUCTION

WITH the progress of web services, various organizations are operating their services around the world gradually. Organizations build a business process on the Internet and provide web services to their partners for sharing. With the continuous development of economic globalization, the demand for information sharing and collaboration between contemporary enterprises and different corporate organizations is becoming increasingly urgent, which why Open API (Open Application Programming Interface) was developed. Open API is a type of open application programming interface. Sneps-Sneppe and Namiot (2012) proposed M2M applications and Open API: what could be next? Chang (2015) has proposed a REST-based Open API architecture for cloud computing. Microsoft, YouTube and Facebook have opened their APIs to the public and are transferring their core competitiveness to the open service platform. In recent years, some domestic enterprises have also established their own open platforms, such as Baidu API Store, Taobao open platform (TOP), Tencent open platform, and so on

A service composition function, which depends on an API management system, is an important component of that. API management enables the integrated API to provide its maximum capability. With API management, companies can publish Web services to APIs not only safely and reliably but also massively. API management can promote the implementation of APIs within departments, partners and developers. At the same time, API management can also benefit from the business provided by the operation organization. API management should provide all the tools involved, including setting up roles, creating usage plans, and providing throttling strategies and data analysis. Moreover, functions provided by single services are limited and cannot meet the complex business requirements. Therefore, service composition is required for APIs to complete a complex business implementation. Maleshkova et al. (2010) have proposed investigating web APIs on the

CONTACT Xi Meng ximeng@zju.edu.cn

World Wide Web, which can not only make use of more resources but also increase business value.

On the basis of the previous description, we have designed a service composition system based on a RESTful API to solve problems such as the lack of unified management and standards and many other defects. Our system can not only manage the entire lifecycle of the API but also achieve richer functionality through the integration of multiple APIs.

In the second section, the relevant research work is introduced. The third section presents the architecture design of the service composition system. The fourth section uses a tourism instance as a business requirement to verify the effectiveness of the system. The fifth section summarizes the full article.

2 RELATED WORKS

2.1 Open API

2.1.1 Open API Overview

OPEN API is an application program interface that is open-ended and uses web technologies such as SOAP and JavaScript to interconnect websites. Open API's contribution to the issue of processing resource integration lies in the following. First, the external access to resources is open to the public and can provide services for resource collocation. Second, resources can be protected by limiting the interface number and provision frequency. The difference between Open API and the traditional application program interface is that Open API does not limit the creation of new applications. Open API provides innovative solutions that make the interconnection between websites more flexible and friendly, provides integration of network information resources or services, and integrates the foundation with third parties. API publishers expose their resources through open interfaces. API application developers can obtain data in different formats (such as XML, JSON, etc.) and pass specified protocols, URL addresses, and functions that conform to API specifications. Open API can also integrate the resources and services of multiple API publishers to provide users with new hybrid applications.

The openness of resources and services is not the right embodiment of that of Open API. The openness of Open API is reflected in the three roles that can be used in the conversion of APIs. More accurately, the boundary between providers, publishers, and users is opened as well. API providers own data, technologies, and services. Services are opened by publishers. In reality, API providers and publishers have developed a large number of APIs.

2.1.2 Field of Application

Currently, Open API on the Internet can be roughly divided into the following categories based on the business areas of the services it provides:

(1) Search API: The search engine service provider mainly implements the search function for the website, opens the search function and provides a search API call. Developers can also use their search results to assemble and compose new Web applications. Such APIs include Google, Yahoo, and so on

(2) Geographic information API: The geographic information API is the most useful Open API and provides a two-dimensional vector for one-dimensional text information, enabling a more accurate description of geographic locations. A large part of the information on the Internet concerns geographical locations. Therefore, the daily call volume of APIs for geographic information is very large. Google, Yahoo and Bing have all opened their own map APIs to the public and can cover the entire globe. At present, domestic Baidu, 51Map, and so on provide open capabilities for geographical information within China.

(3) Text information API: This type of API is mainly used in Internet applications, such as information websites, blogs, and forums, and its basic feature is that it is more convenient for users to obtain or publish information by using words, such as on Twitter, Douban, and so on

(4) Multimedia type: Today, multimedia APIs mainly provide pictures and video resources. People can use these APIs to obtain rich picture or video content. Compared with text information, picture and video information is more colourful. Photo-sharing sites, such as Yahoo's Flickr, and video-sharing sites, such as Google's YouTube, open their multimedia resources by providing Open APIs.

(5) E-commerce API: E-commerce APIs can disseminate enterprise and commodity information. The fundamental goal of such APIs is to attract developers to develop various additional functions and to increase the revenue level of websites by increasing the number of visits and trading volume; on this basis, they will be promoted. Some of the proceeds are shared with developers and attract developers to continue to develop and operate. Typical representatives include Amazon and Taobao.

(6) Life service API: The open capabilities provided by the Life Services API are mainly related to weather inquiries, traffic inquiries, identity card inquiries and inquiries about the rapid increase in demand in recent years. Such APIs provide scenes that are commonly used in people's daily lives to facilitate people's lives. In Baidu's API Store and Easy Source's Show API, there is a special classification for such APIs.

(7) Users and relationships API: User relationship information in Internet information gradually becomes

more and more important with the popularity of social networking sites, such as Facebook, where developers can use Facebook's open API to develop substantial applications on their social networking platforms, which not only benefits developers but also makes social networks more colourful.

2.2 Traditional Web Services & RESTful Web Services

Web service is no longer a novelty. In recent years, it has guided the development of the Internet as a basic principle. However, REST is a new concept proposed by Roy Fielding in his doctoral thesis. That began the competition between traditional Web services and RESTful Web services.

2.2.1 Traditional Web Services

Technologies of traditional web services are mainly based on XML-SOAP-WSDL-UDDI. Erl (2010) has proposed a field guide to integrating XML and web services. Fensel et al. (2005) have proposed web service modelling ontology. In the service architecture, XML is the basis, UDDI is the discovery layer, WSDL is the description layer, SOAP is the package layer, and the HTTP protocol carrying SOAP objects implements the transport layer. The network layer in the service architecture is used to provide communication, addressing routing and other functions, which is the same as the network layer in the TCP/IP network model.

(1) XML (Extensible Markup Language) is an HTML-like plain text markup language. XML is designed and used to transmit and store data, not to display data. Yergeau et al. (1998) have proposed extensible markup language (XML) 1.0. HTML can be used to display data, which is the main difference between them.

(2) SOAP (simple object access protocol) is a protocol that provides an information exchange function implemented on HTTP on the basis of XML. Ryman (2000) has proposed simple object access protocol (SOAP) 1.2. SOAP can describe the format of service information transmission. SOAP consists of four components: an encoding constraint, encapsulation structure, RPC and binding. An ordinary XML document can be used to represent a SOAP message.

(3) WSDL (Web Services Description Language) is also a technology based on XML to describe network services and how to access network services. Christensen et al. (2003) have proposed web services description language (WSDL). WSDL is written in XML, so it is presented by an XML document as well.

(4) UDDI (Universal Description, Discovery and Integration) is a directory service that developers can use to register and search for Web services. Curbera et al. (2002) have proposed unravelling the web services web: an introduction to SOAP, WSDL, and UDDI. Popularly speaking, UDDI is a directory used to store relevant information of Web services. UDDI is implemented by WSDL service description language. UDDI can help consumers search Web services conveniently and facilitates the use and calling of Web services. UDDI has been recognized by academia and industry and has been widely applied.

2.2.2 RESTful Web Services

Before Dr. Roy Fielding proposed the concept of REST, REST was called "HTTP Object Model," and it was not completed. The official name of REST is "Representational State Transfer." It is expected that each developer will have a deep understanding of how to design a good Web application: a Web-based Network in which a user can select the link (state transition) and that leads to the next page, i.e., the next state is passed and presented to the user. Fielding (2000) has proposed architectural styles and the design of network-based software architecture. The "next page" passed to the user is used to indicate the "representation" here and may be a normal HTML page or a normal XML page or may only be some data and services. The “Representational” refers to those.

For a simple example, suppose a user uses the following logic to request a book's resources on a book vendor's Web server. URL: http://www.books.com/computers/Thinking_in_Java. The physical representation of a resource is then responded to the user, assuming it is Thinking_in_Java.html. A user selects a link, such as a picture, price, or book content description in the presentation page, to specify the next action so that the program state can be maintained by the user, as shown in Figure 1.

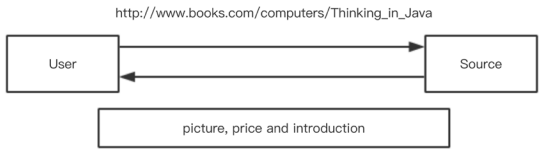

Figure 1. State Transition of REST

The REST architecture is a common set of architectural styles, and its design principles apply to any aspect. However, the web services are the most widely used at present. RESTful web services refer to web services with the REST architectural style. RESTful Web Services are resource-centric Web Services. Resource Identifiers are implemented using the Uniform Resource Identifier URI, and unified methods are implemented using HTTP standard methods. Surhone et al. (2013) have proposed a uniform resource identifier. Lopes and Oliveira (2002) have proposed a uniform resource identifier scheme for SNMP. Totty et al. have proposed a definitive guide of HTTP. Web services present different forms of page content to the user, that is, the representation

of resources. A user opens the next page by clicking on a link in the page to promote changes in the state of the entire web service.

The following describes the characteristics of RESTful Web services, that is, the constraints that the design of RESTful Web services needs to comply with:

(1) Client-Server: The client-server constraint says that the system is divided into two parts: client and server. The client implements the display interface and input data. The server implements the processing of information and storage of data, which improves not only the cross-platform portability of the user interface but also the system's scalability.

(2) No status: The stateless constraint emphasizes that the client-initiated request does not depend on the previous communication; the request itself must include all information required by the server to process the request, and no context state information needs to be stored on the server. This specification improves the reliability and scalability of the system. The disadvantages of the system are also obvious. The system increases the overhead for the client to send duplicate request data and reduces system performance.

(3) Caching: Stateless constraints make the server unable to store state information, and the client must carry all the information in the sent request data, which leads to the loss of system performance. The existence of cache constraints can improve the negative effects of statelessness. If the resource is cacheable, the representation of the resource is saved in the client.

(4) Unified interface: Different from other architectural styles, the core feature of the REST architectural style is unified interface constraints, which is manifested in RESTful Web services, where everything on the network is abstracted as resources. The unified interface emphasizes that all resources are accessed through a common interface.

(5) Hierarchical system: Layered system constraints divide the system's structure into multiple layers based on the system's capabilities. Components in each layer can communicate with neighbouring components without knowing all non-adjacent components. Each level of components can evolve independently without changing the functionality of other layer components.

(6) Code on Demand: Code on Demand constraints are optional and allow the client to be extended, which shows that the server sends the program to be run to the client; the program is downloaded and executed by the client. For example, client functionality is extended by downloading and executing a Java applet or scripted code.

2.3 Service Composition Methods and Models

The combination of Web Services has attracted extensive attention in the areas of process management, artificial intelligence, and other research. The combination of Web Services can be classified according to different standards. In accordance with the level of automation, it can be divided into the manual service composition method, semi-automatic service composition method and automatic service composition method. According to different types of task classification, it can be divided into the business process-driven method and real-time task solution method. The former mainly depends on workflow technology, and the latter mainly depends on artificial intelligence theory. The composition method of Web services can be divided into service compilation and service orchestration and choreography. Yang (2012) has proposed research on a formalization model for web service compilation. Peltz (2003) has proposed web service orchestration and choreography. Service preparation requires a central scheduling node to dominate the Web services that need to be used and to coordinate services to achieve differential operations; service orchestration does not require the central control node to dominate the entire orchestration process. Service orchestration is a point-to-point method of transferring information between services. Finalize the combination. Not only that, service composition methods can also be divided into static combinations and dynamic combinations.

According to the importance of control and data, the service composition model has three types: a process-based model, data-based model, and knowledge management-based model. The process-based model emphasizes control logic and maps tasks in the process into services. This model does not focus on the importance of data, although it has the advantages of convenience and portability. The focus of service aggregation and combination is not the control process but rather data. The control process only needs to play an auxiliary role. In summary, the realization of service aggregation and combination operations is more appropriate to use data-based service aggregation and combination models. In addition, the model based on knowledge management mainly relies on artificial intelligence technology that is not yet mature. Thus, the model used in this article is a data-based service aggregation and composition model.

3 SERVICE COMPOSITION SYSTEM DESIGN

3.1 System Component Architecture

THERE are four main types of components in the system: service publishing/management components, service store components, service composition components, and management components.

3.1.1 Service Publishing / Management Components

In the development of the API, it is usually necessary for professional developers who understand APIs, interfaces, and documents to be involved. In the management of the API, people who are familiar with the API business also need to be present. Therefore, professional API developers can easily use this component.

API development and management are in the system API Publisher interface. The web interface is a structured user graphical interface. API developers can develop, test, and maintain APIs in the API Publisher and can also perform API management-related operations, such as API publishing, lifecycle management, and flow control.

The API Publisher's access address is: https://<HostName>:9443/publisher and must be accessed through the https protocol.

3.1.2 Service Store Components

The API Store exposes collaborative interfaces to API publishers so that they can publish and promote their own APIs and provides API users with the ability to independently register, discover, subscribe, use, and evaluate APIs. The URL of the API Store is: https://<YourHostName>:9443/store and can only be accessed through the https protocol.

3.1.3 Service Composition Components

Service Aggregation and Combination Component API Combination are implemented on the basis of a data-based service composition model. The main functions include acquisition, aggregation and combination design, and issuance of RESTful services.

API Combination can obtain the APIs available on the API Manager platform and aggregate and combine the services available on the platform. To present the internal structure of the service and provide an interface that is easy for the user to develop, the component uses a graphical approach. As a result, developers can easily implement the services they need to receive the services they need through aggregation and combination of services.

As shown in Figure 2, a functional block diagram of service aggregation and composite components is shown. Several functional modules of the component can be understood through that.

(1) Interaction module with API Manager platform: The interaction with the service platform is the basic function of this function module, such as obtaining the service and uploading and synthesizing the code and document after completion, so that the user and the platform can seamlessly interact with each other.

(2) Model operation module: All the graphical elements that need to be used in the RESTful service aggregation and combination model are provided by this function module. Users can create, modify, and delete the model to achieve the purpose of designing the graphical service description and designing service aggregation and combination.

(3) Grammar module: This function module is mainly used to check and parse the RESTful service description and RESTful service aggregation and combination syntax. Through the definition of some rules in the model, it is determined whether the graphical representation of the developer's design accords with the defined rules, and corresponding feedback information is provided to the developer in real-time.

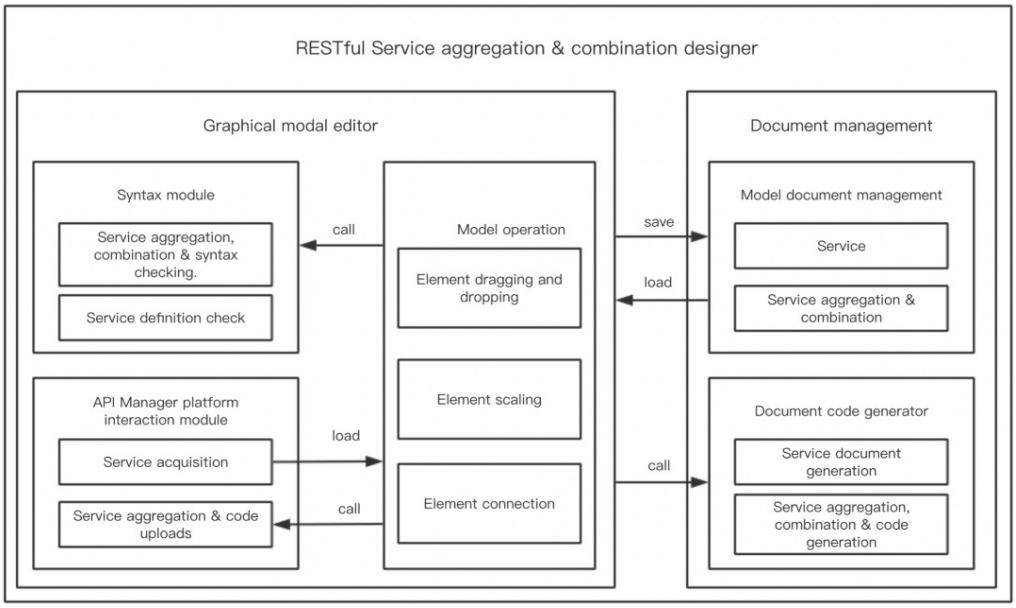

Figure 2. Service composition component functional structure

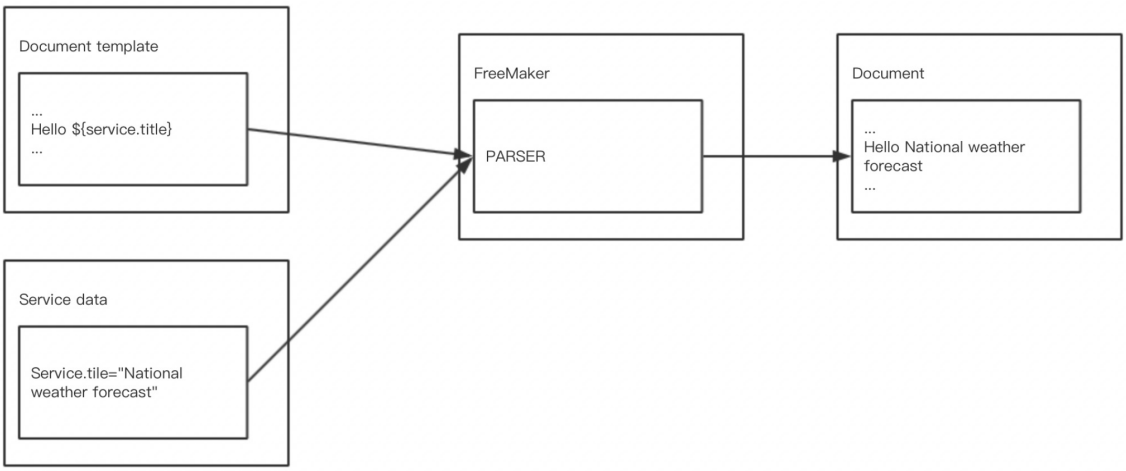

Figure 3. Diagram of how FreeMarker generates document samples

(4) File Management Module: This function module can long-term convert the graphical element representation designed by the developer into a specific description file. The file is saved in JSON format because the model design borrows from JSON design ideas. If you want to save a JSON file, you should specify the structure of the save file.

(5) Document code generator module: This function module has two main functions. The module can generate RESTful service documents according to the developer's service model description or generate service aggregation and combination code according to the service aggregation and combination description designed by the developer and upload the code to the platform. If you want to obtain a uniform style of document, the first step is to define a uniform document template. The second step is to fill in the service information in the appropriate location. The work of this module can be done by calling the FreeMarker tool. FreeMarker is a template engine that was developed using Java. FreeMarker generates text output based on templates. As Figure 3 shows, the FreeMarker tool first modifies the symbols in the template into real data and then outputs the document.

3.1.4 System Management Components

The system management implementation mainly uses the API Gateway component and the Key Manager component.

(1) The API Gateway is mainly responsible for protecting and managing API calls and intercepts API requests to apply policies (such as throttling policies) and manages API statistics. After validating the policy, Web service calls are transmitted by the API gateway to the real backend. If the service call is a token request, the API gateway passes the service call to the key manager. The URL of the API Gateway is: https://localhost:9443/carbon, accessible only through https protocol.

(2) The Key Manager mainly ensures that the client's normal work is not destroyed, and the processing service call is a token request operation. The validity of the OAuth tokens for the subscription and invocation APIs is the connection between the API Gateway and the Key Manager as a whole. The process of obtaining the access token is as follows. First, the subscriber creates a new application and generates an access token for it. This step is completed in the API Store. Then, the API Store notifies the API Gateway that an access token needs to be obtained. Finally, the API Gateway connects to the Key Manager to obtain an access token. The process of verifying the access token is similar to the process of obtaining the token. The API Gateway connects to the Key Manager and obtains the token from the database and then verifies the access token.

All tokens used for authentication are based on the OAuth 2.0 protocol. Secure API authorization is provided by OAuth 2.0 compliant key management. The API Gateway supports API authentication using OAuth 2.0.

The API Gateway has a cache function. When it is not enabled, every time an API call is received, the call authentication is triggered. Use the API Gateway cache to avoid frequent connection to the Key Manager when the API Gateway receives an API call request.

3.2 Database Architecture

The system uses 3 databases, and the following databases are shared among server nodes.

(1) User Manager Database: Stores user and user role information. This information is shared between the Key Manager server, Store, and Publisher. Users can access the Publisher creation API and access the Store consumer API.

(2) API Manager Database: Stores API and API subscription information. The Key Manager server

uses this database to store user access tokens used to verify API calls.

(3) Registry Database: Stores shared information between Publisher and Store. When the API is published through the Publisher, the API is made available in the Store by sharing the registry database. Although information is typically shared only between Publisher and Store components, if you plan to configure a multi-tenant environment (creating and using tenants), you also need to share information for this database between the Gateway and Key Manager components.

3.3 Users and Roles

This system provides four different roles.

(1) Admin: Provide APIs for the server and management API Gateway. Administrator privileges include creating users in the system, assigning roles, managing system security, and more. The administrator role is available in the initial system. The default account name is admin, and the password is admin.

(2) Creator: Professional developers familiar with API technology are called creators. You can use the API Publisher to publish APIs to the API store. The rating and feedback information given by the API user is fed back to the creator in the API store. Creators can add APIs to the API Store but cannot control the API lifecycle.

(3) Publisher: The publisher manages a set of APIs and dictates the API's life cycle, subscriptions, and flow control. Publishers can also access statistics for all APIs.

(4) Subscriber: Subscribers find and subscribe to APIs in the API Store. In addition, they can read documents, rate APIs, provide feedback, and so on

3.4 API Lifecycle

The difference between an API and a service is that the API is the published interface while the service is a program that is executing in the background. APIs have their own lifecycle, independent of the back-end services they depend on. The API publisher manages the API lifecycle in the API Publisher interface.

The following is a list of all the states in the API lifecycle:

(1) Created: After the API is created by the creator, although the API is added to the API Store, the user cannot see this API in the API Store, nor is it deployed to the API Gateway.

(2) Prototyped: The API is deployed as a prototype and published in the API Store. An API prototype is often used to receive feedback on its usability, which users can emulate but cannot add to their own application.

(3) Published: When the API lifecycle is in this period, users can find it in the API Store and add it to their own applications.

(4) Deprecated: The API is still deployed in the API Gateway, but it can no longer be called. The old version of the API will automatically become deprecated after the new version of the API is released.

(5) Retired: After the API expires, it will be removed from the API Gateway and API Store.

(6) Blocked: API access is restricted, the API cannot be called at this time, and it is not displayed in the API Store.

3.5 Access Tokens

The access token is in the form of a simple string that is transmitted via the HTTP header in the request, for example, "Authorization: NtBQkXoKElu0H1a1fQ 0DWfo6IX4a." Both API users and application security require an access token for verification. If the token transmitted by the request is invalid, the request is discarded. The access token works well regardless of SOAP or REST calls.

There are two types of access tokens:

(1) User access token: The authentication of the API end user is through the user access token. API consumers can call APIs from third-party applications, such as mobile applications, which is allowed by the user access token. Users can generate or update user access tokens through the API management platform.

(2) Application access token: The application is authenticated by the application access token. An application refers to a logical collection of APIs. Developers can access all APIs associated with an application through a single application access token.

3.6 Throttling Tiers

Throttling Tiers can limit the number of times the API is successfully called in a certain interval. They can be applied in the following scenarios:

(1) Protect APIs from DoS attacks.

(2) Adjust traffic based on the availability of infrastructure.

(3) Services can be provided to users at three levels: APIs, applications, and resources.

API publishers can define the throttling strategy at three levels: API level, application level, and resource level. As shown in Figure 4, a user's request limit is ultimately determined by the aggregate output of all flow control layers. Here is an illustrative example: if two users subscribe to an API at the Bronze level (bronze level, which can achieve a maximum of 1000 requests per minute), both of them use the application "App1" to subscribe, and the application "App1" is set again. With 1000 requests per minute, there is no limit to the flow control layer for all resource levels. Under this condition, although both users can call this API 1000 times per minute, the theoretical limit for each user is 500 requests per minute because of the limit of 1000 requests per minute set at the application level.

Here are the different levels of throttling, which correspond to the three levels of restriction in Figure 4:

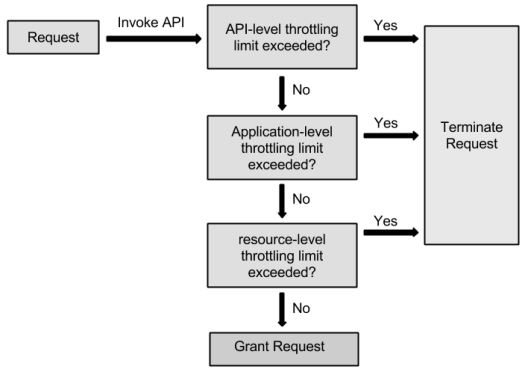

Figure 4. Get request restriction process

3.6.1 API-Level Throttling

Developers can define API-level flow control levels on the API Publisher interface. After API-level flow control is set and the API is published, API subscribers can use different levels when subscribing to the API.

After selecting a different level at the time of subscription, the maximum number of requests for the API to be called by the subscriber within unit time is limited by the subscription level. The default levels are as follows:

(1) Bronze: Allows 1000 requests per minute.
(2) Silver: Allows 2000 requests per minute.
(3) Gold: Allows 5000 requests per minute.
(4) Unlimited: Allows Unlimited requests.

Subscribers can log in to the API Store and use the specified tier consumer APIs only if the subscriber belongs to a role that allows access. In the API Store, the subscriber sees a filtered list of ratings based on the subscriber's role. Only the allowed levels of access for the role are displayed on the page. By default, anyone is allowed access to all levels.

3.6.2 Application-Level Throttling

Developers can create applications on the API Store interface and define application-level flow control levels at the same time. An application is a collection of one or more APIs that are required to subscribe to the API. The application not only allows developers to access an API set using a single application access token but also licenses multiple subscriptions to a single API using different traffic control levels.

Application-level default traffic configuration levels are as follows:

(1) Bronze: Allows 1000 requests per minute.
(2) Silver: Allows 2000 requests per minute.
(3) Gold: Allows 5000 requests per minute.
(4) Unlimited: Allows Unlimited requests.

3.6.3 Resource-Level Throttling

The API consists of one or more resources. Similar to a method or function in the API, each resource handles a specific type of request. Developers can open the API Publisher interface and set resource-level flow control to the HTTP method that calls API resources. The default flow control level is as follows:

(1) Bronze: Allows 1000 requests per minute.
(2) Silver: Allows 2000 requests per minute.
(3) Gold: Allows 5000 requests per minute.
(4) Unlimited: Allows Unlimited requests.

4 INSTANCES

THIS section uses the travel scene as an example to use the system to implement the combined functions of the service. The services required for service portfolio demonstration are a taxi calling service, bus inquiry service, weather inquiry service, hotel inquiry service, train ticket inquiry service and ticket inquiry service. It is assumed here that these services have already been deployed on the service management platform.

Combining the habits of most people in daily life, we can find that, when the weather is raining, people are more willing to travel by car and have less tendency to travel by bus. Because the bus may need to transfer and because the time it takes to stop at the platform will take a long time, the advantage of taxiing is reflected; not only does one not need to transfer, but it is also quicker and easier. Similarly, it is observed that for those who like to travel by car, when deciding between traveling by train or by plane, the latter is more likely.

The first thing that needs to be done is to add the required services. Import the required services to the component on the platform. Then, perform the add route operation. Add an "XOR" route to this tour instance. A logical expression is used to determine if there is rain on the day. If there is rain, a taxi search service will be used. If there is no rain, the bus service will be used. In the travel example, according to the weather field in the weather query service output data, it is determined whether the weather is rainy on the day of the trip to determine whether to use the taxi calling service or the bus inquiring service. As shown in Figure 5, set the specific event attributes. The variable types can be value types, string types, and event types. String types can use the equal(), notEqual(), and contains() functions. Time and value types can use the ">," "<," "=," and "!=" operators.

Next, add a filter. When the train inquiry service and the ticket inquiry service are used, the returned data include a plurality of train ranking lists and a plurality of flight departure lists. To select the portion of the data that is most suitable for combination

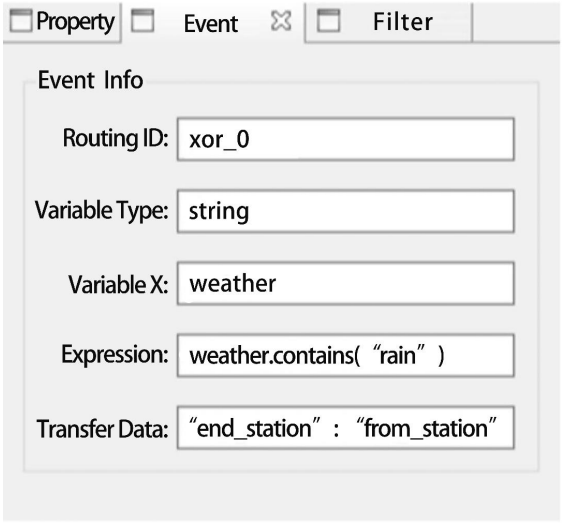

Figure 5. Event settings

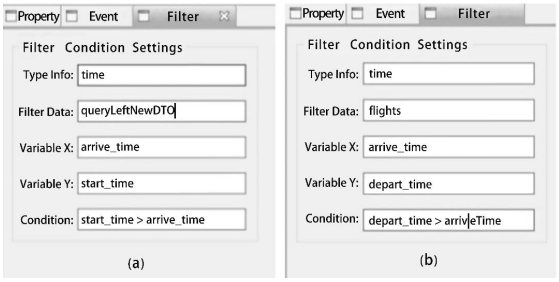

Figure 6. Set filter property instance

display, a filter is required. To make the combined services logically coherent, it is necessary to concatenate the services in chronological order. Thus, we first need to add a filter between the public transportation inquiry service and the train ticket inquiry service and then configure the attributes of the filter. The specific data type that needs to be set is used for comparison. The data type is consistent with the data type of the event condition judgment. Finally, set the data to be filtered, such as the queryLeftNewDTO field, which indicates the list of all trains. The data type of the data to be filtered previously set is generally an array type. This type generally includes the field used for comparison, which is represented by the variable y, such as the start time information of the driving time in the output result of the train inquiry service; the previous service filter value is represented by the variable x, such as arrival time in the output of the public transit query service arrive_time.

To filter the data with the properties of this attribute, you need to set the relationship between these two variables x, y in the condition. The specific settings are shown in Figure 6 (a). Similarly, set the filter between the taxi search service and ticket search service. The specific filter property settings are shown in Figure 6(b).

Finally, to achieve the setting of the event attributes between services, we mainly focus on the setting of the parameter transfer. Figure 7 presents the final rendering of the service portfolio for tourism examples. The dashed line represents the filter, and the straight line represents the event.

Generate the corresponding service combination code through the "File" -> "Save" option. After the code is generated, the assembled API is uploaded to the API Manager platform, and API call testing can be performed using the API Console in the API Store component.

5 CONCLUSION

THE service composition system is mainly composed of two parts: a service management platform and service composition editor. The system can bring the following benefits to the developer management API:

(1) Accelerated API usage: The API's call speed can be increased, which relies on the API management platform to provide a set of web interfaces that can display API documents and test experiences.

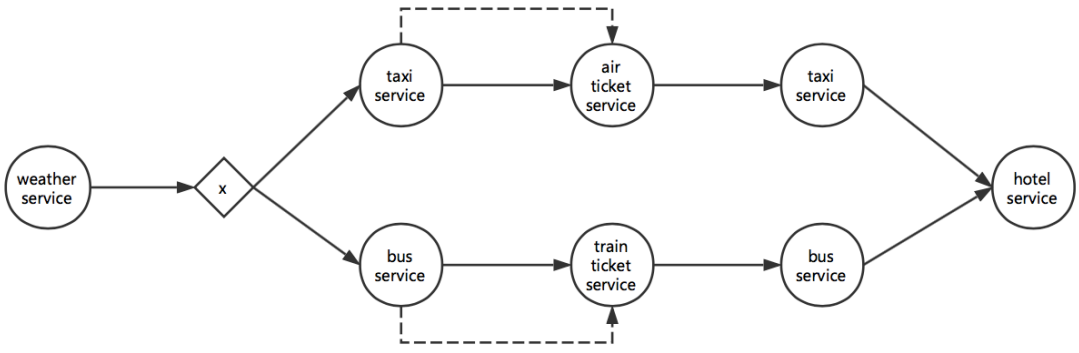

Figure 7. Routing service portfolio final rendering

(2) Improve the discoverability and utility of APIs: currently, many organizations hold hundreds of APIs, but sometimes, the greater the number of APIs, the lower the team's work efficiency because of improper management of the API, such as useless documentation, fragmented experience, and limited discoverability. Therefore, it is necessary to establish an API management platform with a consistent appearance to improve team work efficiency.

(3) API service aggregation and combination: Through integration of API functions via technical means, the API functions formed after aggregation and combination are more powerful or meet different needs through different methods of aggregation and combination.

(4) Protection API: The API is protected by methods such as authentication and throttling.

There are some advantages and disadvantages. Although this article proposes a service management platform that can provide service combinations, the service combination module and service management platform are still two relatively independent systems. The service composition module needs to actively travel to the service management platform to acquire the API; the module and platform are not perfectly combined. In addition, the service aggregation and combination module can only obtain structured services on the service management platform each time the service is acquired. The versatility of the system is limited to a certain degree such that the aggregation and combination functions of the service are restricted and cannot obtain a maximized value. We will improve the above defects one by one in future research work.

6 ACKNOWLEDGMENTS

THIS research was supported by the National Key Research and Development Program of China under grant No.2017YFB1401202 and Zhejiang Province medical and health science and technology platform project No.2017KY497.

7 REFERENCES

C.Y. Chang, (2015). A REST-based Open API architecture for cloud computing. Advanced Science and Industry Research Center.

E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, (2003). Web services description language (WSDL). Encyclopedia of Social Network Analysis and Mining. Springer New York. 146–159.

F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, (2002). Unraveling the web services web: an introduction to soap, WSDL, and UDDI. Internet Computing IEEE. 6(2), 86-93.

T. Erl, (2010). Service-Oriented Architecture: A field guide to integrating XML and Web services. Prentice Hall PTR.

D. Fensel, F. M. Facca, E. Simperl, and I. Toma, (2005). Web service modeling ontology. Applied Ontology. 1(1), 77-106.

R. T. Fielding, (2000). Architectural styles and the design of network-based software architectures. University of California, Irvine.

R. P. Lopes, and J. L. Oliveira, (2002). A uniform resource identifier scheme for SNMP. IP Operations and Management, 2002 IEEE Workshop on. IEEE. 85-90.

M. Maleshkova, C. Pedrinaci, and J. Domingue, (2010). Investigating web APIs on the world wide web. Eighth IEEE European Conference on Web Services. IEEE Computer Society. 107-114.

C. Peltz, (2003). Web services orchestration and choreography. Computer, 36(10), 46-52.

A. Ryman, (2000). Simple object access protocol (SOAP) 1.2. International Conference on Software Engineering. IEEE Computer Society.

M. Sneps-Sneppe, and D. Namiot, (2012). M2M applications and Open API: what could be next?. Internet of Things, Smart Spaces, and Next Generation Networking. Springer Berlin Heidelberg. 429-439.

L. M. Surhone, M. T. Tennoe, and S. F. Henssonow, (2013). Uniform resource identifier. Betascript Publishing. 84 - 87.

B. Totty, D. Gourley, M. Sayer, A. Aggarwal, and S. Reddy, (2002). HTTP: the definitive guide. Oreilly Media, 215(11), 403-410.

X. B. Yang, (2012). Research on formalization model for web service compilation. Computer Engineering, 38(7), 276-278.

F. Yergeau, T. Bray, J. Paoli, C. M. Sperberg-Mcqueen, and E. Maler, (1998). Extensible markup language (xml) 1.0. World Wide Web-internet & Web Information Systems. 39(4), 620–622.

8 NOTES ON CONTRIBUTORS

Wang Hui received a bachelor's degree and a master's degree in computer science from Inner Mongolia University of Technology in 2002 and 2010, respectively. He is currently the director and senior engineer of computer network management center of affiliated hospital of Inner Mongolia Medical University. His research interests included computer big data, medical information and medical Internet of things.

Sun Guang-Yu received the MS degree in Geotechnical Engineering from Zhejiang University of Technology in 2012. He is working at Zhejiang University, engaged in transformation of scientific and technological achievements. His research interests include service computing, computer big data and business process management.

Zhang Qin-Yan received the B.S. degrees in computer science from Zhejiang University, China, in 2005. She is currently a teacher with the school of continuing education, Zhejiang University. Her research interests include service computing and business process management.

Liu Kai-Min is a master student in software engineering at Zhejiang University, China. He received the M.S. in software engineering from Zhejiang University in 2018. His research interests include service computing, business process management and compiler.

Xi Meng is a Ph.D. student in Computer Science at Zhejiang University, China. He received his B.S. degree in Computer Science from Zhejiang University in 2017. His research interests include service computing, data mining and machine learning.

Zhang Yuan-Yuan received the MS degree in Computer Science from Hangzhou Dianzi University in 2007. She is an associate professor of College of Information Technology of Zhejiang Chinese Medical University. Her research interests include Medical Information System and SOA-based medical software.