Service Encapsulation for Middleware Management Interfaces

Xing Chen, Xuanzhe Liu, Xiaodong Zhang, Zhao Liu, Gang Huang

Service Encapsulation for Middleware Management Interfaces

Xing Chen, Xuanzhe Liu*, Xiaodong Zhang, Zhao Liu, Gang Huang
Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing, China
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing, China
{chenxing08, liuxzh, zhangxd10, liuzhao, huanggang}@sei.pku.edu.cn

***Abstract*—Middleware has become the popular runtime infrastructure of modern IT systems. Due to the complex application contexts and various user requirements, more and more applications are now making use of different middleware platforms. It might require the cooperation of several types of middleware, and result in challenging issues for collaborative management of heterogeneous middleware. To solve this problem, it may be a feasible solution to make management interfaces delivered in the form of services and used in service-oriented styles. In this paper, we propose an approach to encapsulating middleware management interfaces into Web services. First of all, we introduce middleware management problems induced by the development of requirements. Then, we present the technical challenges and illustrate the details about how to enable different management interfaces to be in service-oriented styles. Followed by identified technical challenges, we evaluate the approach and primarily discuss how to manage middleware systems collaboratively based on management services.**

Keywords- middleware management; service encapsulation

I. INTRODUCTION

Middleware has become the popular runtime infrastructure of modern IT systems. It plays a crucial role in system management, which aims at keeping middleware working properly and ensures that it can fulfill application requirements at runtime.

To meet various requirements, applications have become more and more complex. Not all applications are running on the same type of middleware, because there is no single type of middleware that can serve various applications in all cases. We need to manage PKUAS (an application server) [1], JBOSS [2], MQSeries [3] and other middleware systems together. Obviously, the heterogeneous middleware platforms have different management capabilities and interfaces. As a result, we need some ways to hide the heterogeneity or reduce the complexity and difficulty of controlling such heterogeneity.

On the other hand, applications increasingly make use of different middleware resources and distributed platforms. To keep applications running properly, it is necessary to manage different middleware systems in the distributed platform collaboratively. Furthermore, today's applications may make use of middleware resources that are provided by the World Wide Web. In enterprises and other organizations, this leads to a distributed application infrastructure in which different elements of this infrastructure are owned and managed by different organizations. This dependency on infrastructure outside an organization's management domain has to be taken into consideration for system management. For example, the Pet Store application running on Google App Engine [4] may use the storage services of Zoho CloudSQL [5]. To keep the Pet Store application running properly, the application container provided by Google App Engine and the database provided by Zoho CloudSQL must be managed globally, instead of managing them separately. To accommodate the ever-increasing number of requests, the computational resources of the application container are enlarged so that the application can get more connected requests at a time. It further results in more requests to the database, which has already been busy in dealing with concurrent transactions. The increasing number of requests makes the workload of the database heavier, which leads to the overall decrease in performance. So it has to be taken into consideration for collaborative management of different middleware systems in the distributed platform. Current systems typically rely on a centrally managed configuration management database (CMDB) [6] to store information on a system's configuration. But this approach is unable to solve the problem of the distribution of management responsibility of applications. It requires a decentralized approach to system management that takes into account the distribution of management information and execution of management processes across organizational boundaries.

The issues of heterogeneity and openness make middleware management complex and difficult. Web service is a defacto industrial standard for encapsulation (which can deal with the heterogeneity issue in the management of middleware), interoperation (which can deal with the open accessibility issue in the management of middleware), flexibility, composition, etc. To solve the problem of collaborative management of middleware, it may be a feasible solution to enable management interfaces delivered in the form of services and used in service-oriented styles [7][8].

However, current middleware systems typically do not provide management functionalities in a service-oriented style. Most of middleware systems may provide web consoles for administrators and some of them may provide configuration files. In some scenarios, there may be exiting management scripts to fulfill complex management tasks. Some middleware systems may provide APIs. So we need

*corresponding author: liuxzh@sei.pku.edu.cn to encapsulate these different kinds of middleware management interfaces to Web services. Furthermore, there are different styles of services, including SOAP Web service and RESTful Web service [9]. Management services in different styles may be needed in different scenarios of middleware management. Hence, we should enable management interfaces to be in different service-oriented styles as required. The heterogeneous middleware management interfaces and the different service-oriented styles make service encapsulation for middleware management interfaces complex and difficult.

The objective of this paper is to propose an approach to encapsulating middleware management interfaces into Web services. The rest of this paper is organized as follows. Section 2 gives an overview of our approach. Section 3 illustrates how to transfer middleware management interfaces to management functionalities in the unified model. Section 4 presents the details about how to enable management functionalities in the unified model to be delivered in the form of services. Section 5 evaluates the encapsulation approach and Section 6 gives some discussions. Finally, we discuss related work in Section 7 and conclude this paper in Section 8.

II. Approach Overview

In this section, we give a brief overview of our service encapsulation approach for middleware management interfaces, which enables middleware management interfaces to be in service-oriented styles, as shown in Figure 1. Firstly, we transfer management interfaces into management functionalities in the unified model. Secondly, we enable management functionalities to be in service-oriented styles.

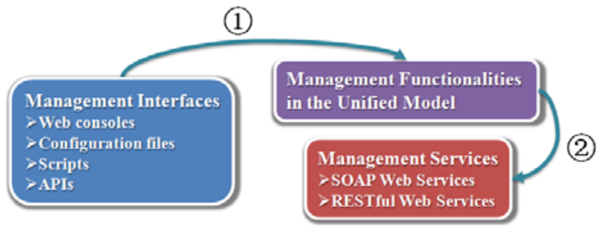


Figure 1. the Overview of Our Approach

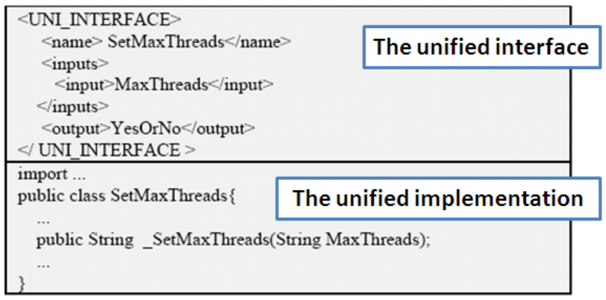
There are different types of general middleware management interfaces. Some types of management interfaces, like APIs and Scripts, may be invoked directly. But other types of management interfaces, like web consoles, just promise that we can check the parameters of middleware and they do not provide the approach to reading and updating the values of the parameters through any routine interfaces. The heterogeneity of middleware management interfaces makes the encapsulation complex and difficult. So we present a unified model to solve this problem. To encapsulate management interfaces into Web services, we firstly transfer management interfaces into management functionalities in the unified model.

Given the management functionalities in the unified model, we then enable them to be in service-oriented styles. Management services in different styles may be needed in different scenarios of middleware management. On the one hand, SOAP Web service may expose different complex operations, and the style of SOAP Web service is functionality-centric. Most management functionalities are to fulfill some management task, so they are suit to be in the style of SOAP Web service. On the other hand, RESTful Web service just has four operations, including “Get”, “Post”, “Put” and “Delete”. The style of RESTful Web service is resource-centric and only part of management functionalities are suit to be in the form of RESTful Web service, which operate on resources of middleware, such as parameters, database resources and so on. For example, we may read or update values of parameters of middleware through the “Get” or “Put” operations and create or delete database resources through the “Post” or “Delete” operations. But there are some complex management functionalities to fulfill a series of management tasks, which are usually provided by management scripts. Because their operations are complex and these complex management functionalities do not operate on resources of middleware, they are not suit to be in the form of RESTful Web service. In practice, we should decide which style of services the management functionalities are suit to be in.

Management services may be used to support collaborative management of middleware. For instance, SOAP management services may be assembled to business processes, in order to fulfill more complex management tasks. RESTful management services may be integrated, in order to gain views that transcend the perspective of a particular domain for administrators. Although this paper does not aim at the approach to collaborative management of middleware, we plan to give some discussion about how to make it feasible based on management services in Section 6.

III. The Unified Model of Interfaces

The unified model we proposed includes the unified interface and the unified implementation, as shown in Figure 2. There are some important metrics in the unified interface, which needs to be defined, such as the operation name, the inputs and the output. The unified implementation follows the constraints of the unified interface and has the specific middleware management functionality.


The unified interface
The unified implementation

```
<UNI_INTERFACE>
   <name> SetMaxThreads</name>
   <inputs>
     <input>MaxThreads</input>
  </inputs>
   <output>YesOrNo</output>
</ UNI_INTERFACE >
```

```
import ...
public class SetMaxThreads{
   ...
   public String  _SetMaxThreads(String MaxThreads);
   ...
}
```

Figure 2. the Unified Model

We provide the mapping rule to fulfill the transformation from middleware management interfaces to management functionalities in the unified model, as shown in Figure 3. The content of the mapping rule includes the unified interface and the raw middleware management interface. The interfaces in the content of the mapping rule are the same as the unified interfaces in the model. The raw middleware management interfaces, which descript the details how to invoke management functionalities in routine interfaces, are intended to be generated to the unified implementations. The formats of the raw middleware management interfaces are defined in advance and different among types of management interfaces. To transfer management interfaces into management functionalities in the unified model, different contents of the mapping rule should be provided, according to the types of the management interface. In the following paragraphs, we plan to discuss the mapping rule for four different middleware management interfaces. Although there may be some other types of middleware management interfaces, we may do the transformation similarly.

A. The Mapping Rule for APIs

We may invoke APIs to manage middleware systems directly. So to fulfill the mapping rule, we just have to provide the information about calling the objective method, including the package name, the method name and so on, as shown in Figure 3.

B. The Mapping Rule for Scripts

The mapping rule for scripts is similar to the rule for APIs. We should provide the information about invoking the objective script, such as the path, the script's name and so on, as shown in Figure 3.

C. The Mapping Rule for Configuration Files

The interfaces of configuration files just provide the configurable parameters but not any methods to be invoked directly. We provide a rule to generate the routine interfaces from configuration files.

According to the rule, we should provide the "file_path" and the "xml_element", as shown in Figure 3. The "file_path" shows the location of the configuration file of middleware. The "xml_element" is a fragment of the configuration file, which locates the objective parameter that we need in the configuration file, as shown in Figure 4. Depend on the "xml_element", we may access the objective parameter. Through reading or updating the value of the parameter in the configuration file, we may fulfill the management functionalities to get or set the parameter of middleware. So given the information above, we may generate the objective routine interfaces.

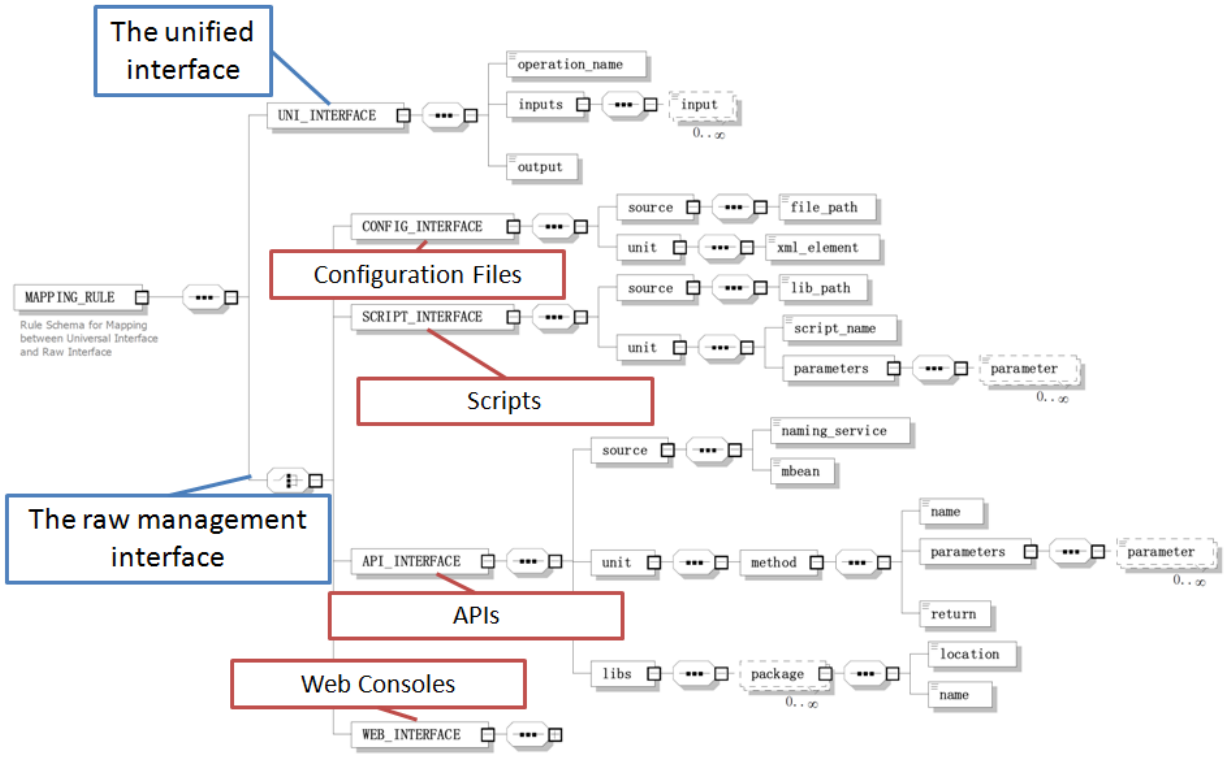


Figure 3. the Mapping Rule for Middleware Management Interfaces

```xml
  <?xml version="1.0" encoding="UTF-8" ?>
- <pkuas>
  - <ServiceManager start="true">
    - <Service Class="pku.as.web.tomcat5.Server" Name="tomcat5" LoaderDir="catalina5">
      - <Properties Name="Connector">
          <Set Name="maxThreads" Value="900" />
        </Properties>
      </Service>
    </ServiceManager>
  </pkuas>
```

The objective parameter

Figure 4. an Example of the "xml_element"

D. The Mapping Rule for Web Consoles

The interface of web consoles is similar to configuration files'. But it is more complex and difficult to read and update values of parameters on the web console than in the configuration file. Although the interface does not provide the method invoked to read or update parameters and there are not any local configuration files, we may access the parameters on the web consoles as administrators do.

What administrators do on the web console is in essence to send requests to the server of the web consoles. So we may also read and update parameters through sending requests to servers of web consoles. However, there is a problem in this process. Management interfaces provided by web consoles are not in the service-oriented style and they are stateful. For instance, administrators' logging in the server, interring the web container page and setting the value of the parameter are intended to send three different requests to the server. Once a request is sent to the server, the session state may be changed and kept. Only if the session state is suitable, the request is intended to be fulfilled. So if we just send the last request, the parameter cannot be set. Therefore, we should keep the sample sequence of the requests from logging in the server to setting the value of the parameter. Then we may set it through sending the requests modified from the sample sequence.

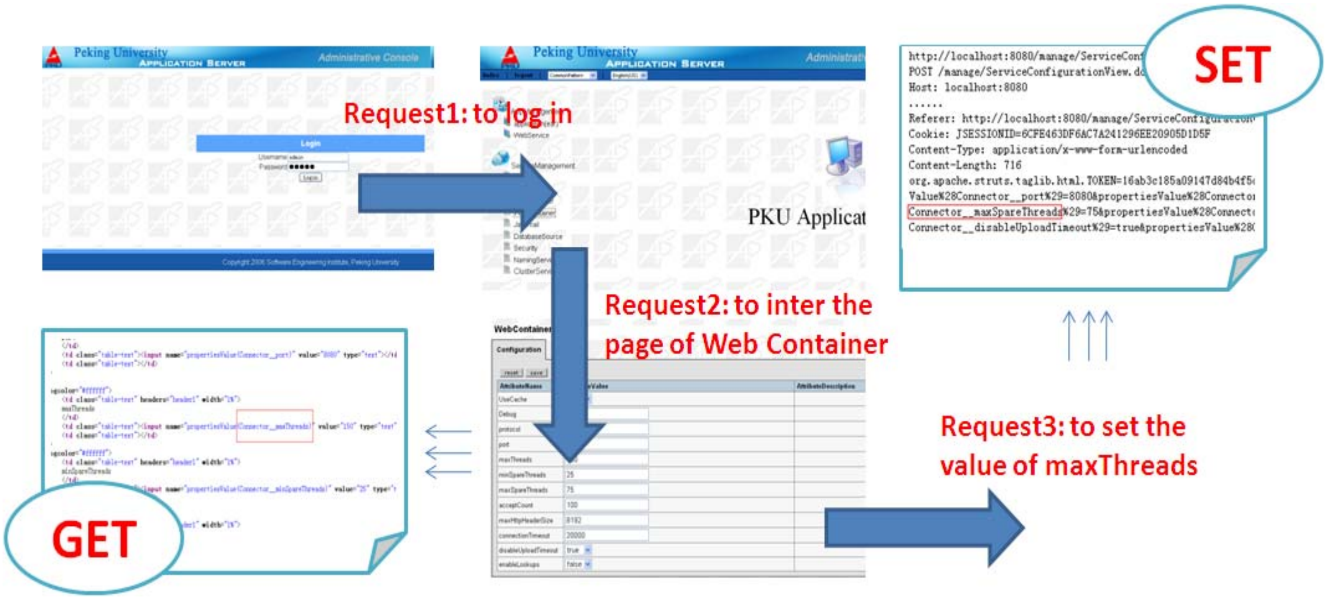
We define the mapping rule for web consoles, as shown in Figure 5. The "requests" are http headers and it may be acquired by some exiting tools, such as Live HTTP headers [10]. The "element" indicates which parameter we plan to read or update. The "method" defines whether the function aims at getting or setting the parameter. In the implementation, we send the sequence of requests to the server. If the method is to read the parameter, we intend to search the value of the parameter on the last page. So we should analyze the content of the last page returned from the server. If the method is to update the parameter, we intend to send the sequence of requests with the new value of the parameter. So we should modify the value of the parameter in the last request of the sample sequence.

```
<MAPPING_RULE>
  <UNI_INTERFACE>
    <name> SetMaxThreads </name>
    <inputs>
      <input>MaxThreads</input>
    </inputs>
    <output>YesOrNo</output>
  </ UNI_INTERFACE >
  < CONSOLE_INTERFACE >
    <requests>
      <request>/home/pku/pkuas/requests/1</request>
      <request>/home/pku/pkuas/requests/2</request>
      <request>/home/pku/pkuas/requests/3</request>
    </requests>
    <element>Connector__maxThreads</element>
    <method>set</method>
  </ CONSOLE _INTERFACE>
</MAPPING_RULE>
```

The sample sequence of requests

Figure 5. the Mapping Rule for Web Consoles

Now we take the "max threads" of the connector for an example, which is a parameter of PKUAS [1]. PKUAS provides the web console for administrators to adjust parameters and we intend to access the value of the "max



```
http://localhost:8080/manage/ServiceCon
POST /manage/ServiceConfigurationView.do
Host: localhost:8080
......
Referer: http://localhost:8080/manage/ServiceConfiguration
Cookie: JSESSIONID=6CFE463DF6AC7A241296EE20905D1D5F
Content-Type: application/x-www-form-urlencoded
Content-Length: 716
org.apache.struts.taglib.html.TOKEN=16ab3c185a09147d84b4f5
Value%28Connector__port%29=8080&propertiesValue%28Connector
Connector__maxSpareThreads%29=75&propertiesValue%28Connect
Connector__disableUploadTimeout%29=true&propertiesValue%28
```

```
  </td>
  <td class="table-text"><input name="propertiesValue(Connector__port)" value="8080" type="text"></td
  <td class="table-text"></td>

gcolor="#ffffff">
  <td class="table-text" headers="header1" width="1%">
  maxThreads
  </td>
  <td class="table-text"><input name="propertiesValue(Connector__maxThreads)" value="150" type="text"
  <td class="table-text"></td>

gcolor="#ffffff">
  <td class="table-text" headers="header1" width="1%">
  minSpareThreads
  </td>
  <input name="propertiesValue(Connector__minSpareThreads)" value="25" type="

  der1" width="1%">
```

Figure 6. the Mapping Rule for Web Consoles threads" in this way too. We plan to achieve management functionalities to get and set the value of the "max threads" in routine interfaces, as shown in Figure 6.

Firstly, we should simulate administrators to get and set the value of the "max threads", in order to acquire the sample sequence of requests sent to the server of the web console. In this instance, the operation to get the parameter has 2 requests, including the requests to log in and to inter the page of the web container. The operation to set the parameter has 3 requests, including the two requests above and the request to update the value of the "max threads".

Secondly, we should make sure the "element", which represents the parameter, in the request and the content of the web page. In this instance, the "element" is "Connector__maxThreads". When we plan to get the value of the parameter, we should send the sample sequence of requests, locate the position of the element in the content of the last page returned and get the value. On the other hand, when we plan to set the value of the parameter, we should locate the position of the element in the last request of the sample sequence, modify the value and send the sequence of requests.

Finally, we may generate the objective management functionality in the unified model, according to the content of the mapping rule.

IV. Management Services Generation

We distinguish the styles of SOAP and RESTful Web services and believe that they are may be used in different scenarios. As we mentioned above, most of management functionalities are suit to be in the style of SOAP Web service and only part of management functionalities, which operate on resources of middleware, are suit to be in the style of RESTful Web service.

Therefore, in this paper, we encapsulate most management functionalities into SOAP Web services and just encapsulate management functionalities to create, read, update or delete resources of middleware into RESTful Web services.

A. SOAP Management Services Generation

We may enable most middleware management functionalities to be in the form of SOAP Web service. There are some exiting works to do the transformation, such as Axis2 [11]. Given management functionalities in the unified model, we may generate management services through the JAVA2WSDL functionality of Axis2.

B. RESTful Management Services Generation

RESTful Web services represent some resources and have the "Get", "Post", "Put" and "Delete" operations. Management functionalities to operate on resources of middleware may be in the form of RESTful Web service.

To generate a RESTful management service, we should define it and map its operations to specific management functionalities in the unified model. For example, we define a RESTful management service as the "max threads" and then map its "Get" and "Put" operations to the functionalities to read and update the value of the "max threads".

RESTful management services may have just one or two operations. For instance, some parameters change from the state of the middleware systems, like the utilization rate of the thread pool. They cannot be created, updated or deleted. If the management service represents this type of parameter, it just has the "Get" operation.

To fulfill the mapping from the operations to specific management functionalities, we provide a method of JAVA2RESTful, as shown in Figure 7. When there comes a HTTP request, the request is parsed to the functionality name and the parameters. Then the objective management functionality is intended to be invoked. Finally, the result is generated in the form of JSON [12], as the response.

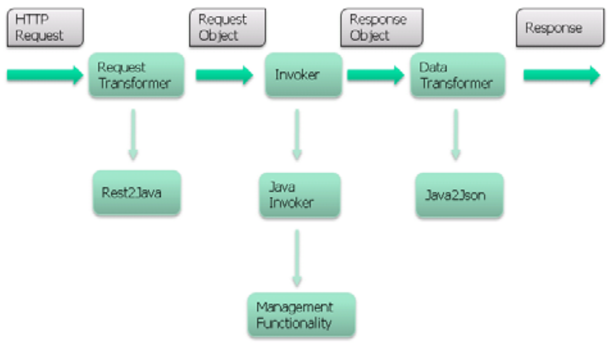


Figure 7. the Approach to JAVA2RESTful

Given the facts above, we may generate RESTful management services from management functionalities in the unified model.

V. Evaluation

We have provided a tool to encapsulate middleware management interfaces into Web services according to the approaches above, which transfer contents of the mapping rule to management services for administrators, as shown in Figure 8. We plan to do some experiments to prove that the encapsulation approach effectively decreases the time to enable middleware management interfaces to be in service-oriented styles.

To enable management interfaces to be in the service-oriented styles, we have to descript the details how to invoke management interfaces, whether tool assisted or manually, but the tool may cut down the time to do coding. We invite some administrators, who are familiar with Web services, to encapsulate middleware management interfaces into Web services, whose functionalities are to read or update the values of the parameters of middleware. The facts show that the tool may cut down about half of time to fulfill the tasks, as shown in Figure 9.

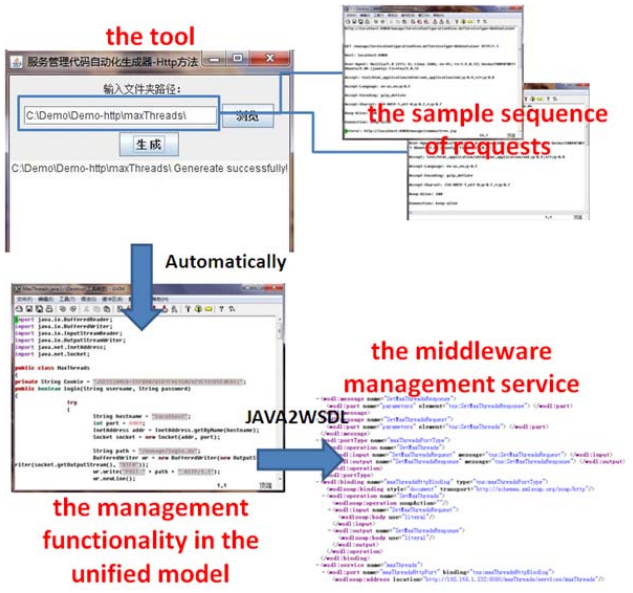

the tool
服务管理代码自动化生成器-Http方法
输入文件夹路径:
C:\Demo\Demo-http\maxThreads\
浏览
生成
C:\Demo\Demo-http\maxThreads\ Genereate successfully!
the sample sequence of requests
Automatically
JAVA2WSDL
the middleware management service
the management functionality in the unified model

Figure 8. The Tool for Management Services Encapsulation

Management Services		APIs	Scripts	Configuration Files	Web conslos
Tools Assisted	SOAP	5 min	4 min	8 min	14 min
	RESTful	5 min	4 min	8 min	14 min
Manually	SOAP	9 min	8 min	21 min	41 min
	RESTful	11 min	10 min	23 min	41 min

Figure 9. Comparison of Tool Assisted and Manually

VI. Discussion

In this paper, to aim at the issues of heterogeneity and openness in middleware management, we propose an approach to encapsulating middleware management interfaces into Web services. There are still several open issues to further address. Firstly, middleware management styles based on management services should be discussed, in order to validate that it is necessary to encapsulate management interfaces into Web services. Secondly, it takes much time to fulfill the content of the mapping rule, in order to transfer different management interfaces into management functionalities in the unified model. Thirdly, the encapsulation approach may be adopted in other similar scenarios. Finally, to support actual inter-operation between heterogeneous middleware products, further research should be done in inter-operation between middleware management services.

A. *Middleware Management Styles Based on Management Services*

Some management styles are introduced to illustrate how to manage middleware systems based on management services, as shown in Figure 10. Management functionalities of middleware systems are exposed in service-oriented styles, and the management services are deployed locally and can be invoked locally or in other domains. These management services may be used for collaborative management of middleware.

As we mentioned above, management services in the style of SOAP Web service are functionality-centric and ones in the style of RESTful Web service are resource-

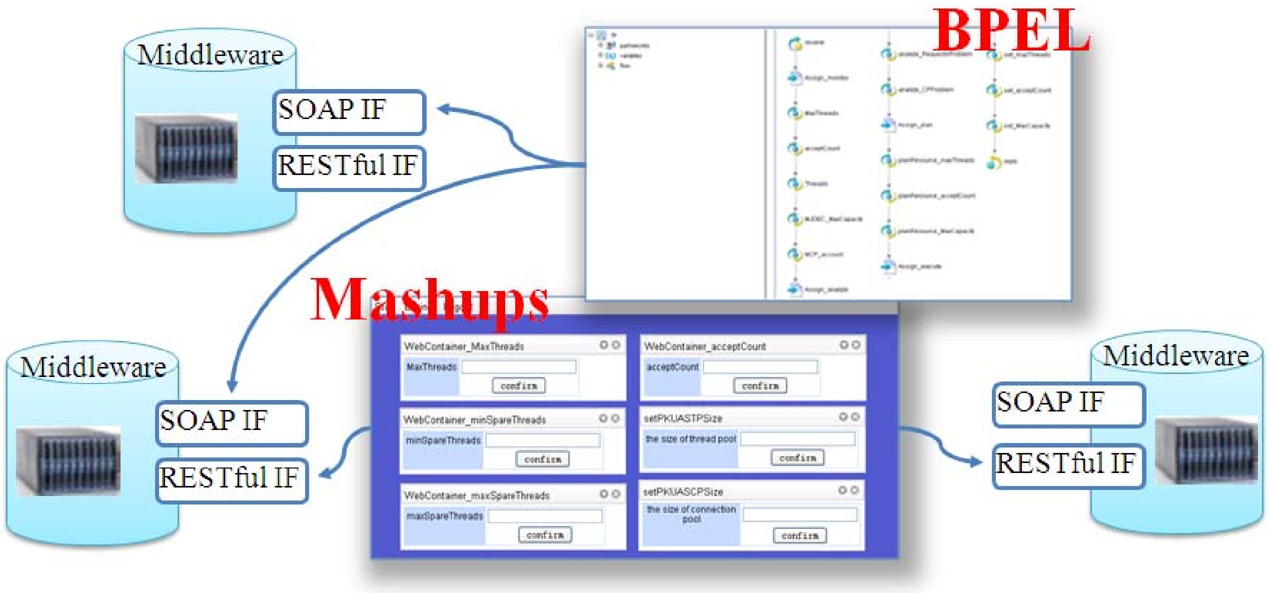


Figure 10. Middleware Management Styles Based on Management Services

centric. Different styles for collaborative management of middleware may be based on different types of Web services.

Management services in the style of SOAP Web service are to fulfill some management tasks and they may be assembled to be a business process, in order to fulfill more complex management tasks. For example, such middleware management functions as monitoring some relevant parameters, checking the possible middleware failures and restarting the middleware system can be assembled together to implement the task to guarantee the system reliability.

Management services in the style of RESTful Web service represent some resources of middleware and they may be integrated in the forms of mashups, ATOM and so on, in order to gain required views. For instance, in reference [15], the management interface is represented as an XML document that can be retrieved at the URL using an HTTP GET request, and the objective of feed management is the creation of mashups that enable interested stakeholders to gain views that transcend the perspective of a particular domain.

Given the facts above, although implementations of the management interfaces are different among middleware systems, they may be unified in service-oriented styles. On the other hand, one tenet of our approach is to use Web-based approaches to deal with cross-domain management issues due to the high level of standardization of Web services interaction. So management services may meet the issues of heterogeneity and openness in middleware management.

There are some challenges in middleware management styles based on management services. For example, management services in SOAP style may be composited to be a business process, and sometimes administrators are needed to involve in the process, in order to fulfill some management tasks. There need to be some approaches to enable it. Furthermore, some automatic mechanisms are needed for assembling or integrating management services. We are interested in these issues and have been doing research in middleware management styles based on management services.

B. Automatic Generation for the Mapping Rule

It takes much time to fulfill contents of the mapping rule, in order to transfer different management interfaces into management functionalities in the unified model. The contents of the mapping rule descript how to invoke different management interfaces and sometimes they are hard to achieve. For instance, to accomplish the content of the mapping rule for web consoles, administrators should provide the sample sequence of requests which are sent to the server of web consoles. But the requests are not easy to acquire. Although there are some tools to help achieve the requests, administrators should select valid ones from them. Similarly, it takes about 8 minutes to fulfill the content of the mapping rule for each configuration file. Some automatic mechanisms need to be introduced to solve this problem.

C. Reusing the Encapsulation Approach

We propose an approach to encapsulating middleware management interfaces into Web services. This encapsulation approach may be adopted in other similar scenarios.

Web service is a defacto industrial standard for encapsulation, interoperation, flexibility, composition, scalability, etc. In order to meet the issues of heterogeneity, openness and so on in software systems, it is a feasible and attractive solution to encapsulate interfaces into Web services [7][8].

This paper aims at the encapsulation approach for middleware management interfaces, such as APIs, Scripts, Web consoles and Configuration files. Other software systems may have some similar interfaces. These interfaces may be encapsulated to Web services in the same way. So the encapsulation approach may be reused in other scenarios.

D. Further Research in Service Inter-Operation

There exist many levels of the inter-operation between middleware management services, including format, syntax and semantic. The objective of this paper is just to address the format issue. To support actual inter-operation between heterogeneous middleware products, further research should be done. It seems that we may need the standards in the middleware management domain such as standard data structure, standard operations, parameters, etc. There are some exiting works to address the service inter-operation and we also have made some efforts in the inter-operation between middleware management services.

VII. Related Work

There are some similar tools which are used to manage middleware systems through exiting management interfaces, such as Hyperic [13] and Tivoli [14]. For instance, the agents of Hyperic are deployed to the computers which middleware systems are running on. The agents discover middleware systems on the computers and collect management interfaces. Then administrators may control the systems through the agents. But the agents do not have the abilities to control all kinds of middleware systems, but just some types previously defined. On the other hand, Hyperic system relies on a centrally managed configuration management database (CMDB) to store information on a system's configuration. As we mentioned above, middleware systems in different organizations' management domains cannot be managed centrally.

Heterogeneity and openness make middleware management much more complex and difficult. Some current approaches based on management services are raised above. For instance, the approach in reference [15] is based on RESTful access to configuration information across domain boundaries, RESTful representation of service process state information as a basis for service process integration and ATOM-based distribution of updates as a novel foundation for service management. The approach in reference [7] is based on SOAP management services and the management services are composited to fulfill new management tasks. The approaches above may prove that it makes sense to encapsulate middleware management interfaces into Web services.

There are some exiting works [16] about approaches to designing Web services too. But the characteristics of middleware management interfaces are not taken into account. On the other hand, this paper presents an approach to encapsulating middleware management interfaces into Web services.

VIII. Conclusion and Future Work

In this paper, we propose an approach to enabling middleware management interfaces to be delivered and used in service-oriented styles. This paper makes the following contributions.

Firstly, we introduce middleware management problems induced by the development of requirements. Heterogeneity and openness make middleware management complex and difficult. It may be a feasible solution to encapsulate middleware management interfaces into Web services.

Secondly, we propose an approach to enabling middleware management interfaces to be in service-oriented styles. We present a unified model for management interfaces and provide the mapping rule to do the transformations of four general management interfaces. Then we discuss the details about how to encapsulate management functionalities in the unified model into Web services.

Finally, we validate that the approach that we proposed may effectively decrease the time to encapsulate middleware management interfaces into Web services. Then, some management styles are introduced to illustrate how to manage middleware systems based on management services.

As we mentioned above, there are some open issues for this paper. We pay our attention to management styles based on management services and have made some progresses, such as the "Human-Computer Interaction" (HCI) service and the approach to automated composition of middleware management services. On the other hand, we have been doing further research based on this service encapsulation approach. We attempt to construct a middleware cloud based on lots of management services, including middleware management services and some management services on virtualization. The customers of the middleware cloud may customize and manage many middleware products through management services, such as PKUAS, APUSIC [17], JBOSS and so on.

Acknowledgment

The research leading to these results is partially supported by the National Key Basic Research and Development Program of China under Grant No. 2009CB320703; the National Natural Science Foundation of China under Grant No. 60821003, 60933003; the High-Tech Research and Development Program of China under Grant No. 2009AA01Z16; and the EU FP7 under Grant No. 231167.

References

[1] Mei Hong，Huang Gang．PKUAS：an architecture-based reflective component operating platform. In the 10th IEEE International Workshop on Future Tmnds of Distributed Cornputing Systems. Suzhou, China, 2004, 163—169.

[2] JBoss Labs. JBoss Application Server. http://labs.jboss.com/jbossas/.

[3] IBM. MQSeries. http://www.mqseries.net/.

[4] Google. Google App Engine. http://code.google.com/intl/appengine/appengine/.

[5] Zoho. Zoho CloudSQL. http://cloudsql.wiki.zoho.com/.

[6] Estublier, J. 2000. Software configuration management: a roadmap. In Proceedings of the Conference on the Future of Software Engineering (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM, New York, NY, 279-289.

[7] LI Ying, HUANG Gang, LIU Tiancheng, YANG Jie, LIU Zhao. Towards Management as a Service. Journal of Frontiers of Computer Science and Technology. 2008, 2, (4): 346-355.

[8] Yang, B., Wang, H., and Chen, Y. 2008. Management as a Service for IT Service Management. In Proceedings of the 6th international Conference on Service-Oriented Computing (Sydney, Australia, December 01 - 05, 2008). A. Bouguettaya, I. Krueger, and T. Margaria, Eds. Lecture Notes In Computer Science, vol. 5364. Springer-Verlag, Berlin, Heidelberg, 664-677.

[9] Fielding, R. T. and Taylor, R. N. 2002. Principled design of the modern Web architecture. ACM Trans. Internet Technol. 2, 2 (May. 2002), 115-150.

[10] Daniel Savard. Live HTTP Headers. http://livehttpheaders.mozdev.org/.

[11] Apache. Axis2. http://ws.apache.org/axis2/.

[12] Standard ECMA-262 3rd Edition. A Subset of JavaScript. http://www.json.org/.

[13] Hyperic. The Open Source IT Management Tool. http://www.hyperic.com/.

[14] IBM. Tivoli. http://www-01.ibm.com/software/tivoli/.

[15] Ludwig, H., Laredo, J., Bhattacharya, K., Pasquale, L., and Wassermann, B. 2009. REST-based management of loosely coupled services. In Proceedings of the 18th international Conference on World Wide Web (Madrid, Spain, April 20 - 24, 2009). WWW '09. ACM, New York, NY, 931-940.

[16] Laitkorpi, M., Selonen, P., and Systa, T. 2009. Towards a Model-Driven Process for Designing ReSTful Web Services. In Proceedings of the 2009 IEEE international Conference on Web Services - Volume 00 (July 06 - 10, 2009). ICWS. IEEE Computer Society, Washington, DC, 173-180.

[17] Kingdee-Middleware. Apusic Platform. http://www.apusic.com/.