WSRush: A Web Service Ready Application Platform

Chun-Hsiung Tseng[1], Yung-Hui Chen[2], Yan-Ru Jiang[1]
[1]: Department of Information Management, Nanhua University
[2]: Department of Computer Information and Network Engineering, Lunghwa University of Science and Technology, Taoyuan City, R.O.C.
Yung-Hui Chen is the corresponding author
Email: lendle_tseng@seed.net.tw

Abstract

In this research, WSRush, a platform that can be used to simplify the development of Web service applications is proposed. With the platform, developers simply inherit from a provided software module and then the platform will automatically transform developer provided modules into Restful based Web services. The WSRush platform consists of three layers: the core engine and service provider interface, the Web interface, and the Web service utilities. For most developers, relying on the Web service utilities layer is sufficient. But they can still extend the functionality of WSRush by inheriting from the core engine layer and the Web interface layer.

Keyword: WebService, SOA, Platform

Introduction

The wide adoption of Web service technologies has demonstrated high impacts on software development methods. Protocols such as SOAP[1] and Restful[2] help develop clear separations among functionalities, data models, and user interfaces. Despite of its elegance, Web service technologies are still considered complex for inexperienced programmers. As a teacher offering programming courses in information management department, the researcher has tried various methods to make students familiar with the concept of Web service, however, only few students can really understand the technology and even fewer students can utilize it in real world software projects. Considering the importance of Web service technologies, it will be beneficial to help students and inexperienced get acquainted with them as soon as possible. As a result, the researchers developed WSRush, a Web service ready application platform.

Why is it a challenge to get acquainted with the concept of Web service technologies? Perhaps a reason is there are just too many concepts to learn. To develop a Web service based application, usually, developers have to handle the following issues:

1. choose an appropriate protocol; Web service applications usually adopt the SOAP or the Restful protocol, but there are numerous related protocols to choose
2. deal with input/output data formats properly; xml and JSON are two frequently data formats adopted in Web service applications, however, developers must be familiar with a set of tools/technologies to cope with these data formats
3. figure out a seamless way to integrate Web service technologies with their development environment; Web service technologies themselves are neutral to development environments, but developers will need adequate integration to reduce the difficulties of using Web service technologies in their projects

Certainly, there are already tools helping developers face the above issues. However, the diversities of these tools cause additional burdens for developers. Furthermore, the vendor-lock-in problem will occur in the solution look up phase.

The goal of this research is to reduce the complexities of adopting Web service technologies by providing a platform that has built-in Web service natures, requires no further integration, and keep the neutrality. The platform has the following characteristics:

1. the platform does not require developers to handle the protocol part; developers simply inherit from a provided software module (in the current stage, the platform is implemented in Java[3], so developers have to extends from a provided Java class) and then the platform will automatically transform developer provided modules into Restful based Web services
2. all input/output data will follow the JSON data formats and the platform provides libraries to help developers handle the encode and decode of JSON data
3. the platform implements tools to manage global resources such as database connections so system resources can be effectively utilized and developers do not have to deal with resource management
4. the platform itself integrates several industrial-standard technologies such as jQuery[4],

[1] https://en.wikipedia.org/wiki/SOAP

[2] https://en.wikipedia.org/wiki/Representational_state_transfer

[3] http://www.oracle.com/technetwork/java/javase

[4] https://en.wikipedia.org/wiki/JQuery

Spring Framework[5], and Jersey[6], etc.; the popularity and standardization of these technologies will reduce the risk of vendor-lock-in

With these characteristics, the researchers believe the proposed platform will help in-experienced developers and students manage the Web service technology. Besides, experienced developers will also benefit from the platform with its well integrated set of technologies.

In this research, we will at first give a detailed introduction to the functionality and architecture of the platform. Then, the design of the platform will be presented, followed by discussions about the design decisions.

Related Works

The importance of the service oriented architecture (SOA) can never be overestimated. The definition of SOA is explained in detail in [1]. Web service standards are the central idea of SOA. In [2], Web services are defined as the integration technology preferred by organizations implementing service-oriented architectures. The adoption of SOA is extremely wide. Trkman and Kovačič discusses the phases of SOA adoption [3]. The research of Luthria and Rabhi [4] summarized factors influencing the adoption of SOA: the perceived value of SOA to the organization, the organizational strategy, organizational context or culture, organizational structure, potential implementation challenges, and given the current environment of increasingly stringent regulations and accountability requirements, the governance or the management of such technology. Among them, our research focuses on reducing the implementation challenges, especially for inexperienced developers and students. As stated in the work of Lopez, Casallas, and Villalobos, "*providing environments where students go beyond learning some concepts and specific technologies to truly apprehend the complexity involved in SOA is a major challenge* [5]." The work of Paik, Rabhi, Benatallah, and Davis designed a dedicated virtual teaching and learning space for students to learn SOA [6]. Spillner implemented SPACEflight, which is a versatile live demonstrator and teaching system for advanced service-oriented technologies [7]. Furthermore, it is proved that using Web services in introductory programming classes can enhance students' learning performance [8]. Frameworks such as Spring Framework[5] and Ruby on Rails[7] provide similar functionality with regards to the creation of Web services. The uniqueness of the proposed framework is its simplicity. Most comparative frameworks are full-stack frameworks. For example, Spring Framework covers many aspects of the life cycle of a Web service such as controllers, interceptors, and dependency injection technologies, etc. The proposed framework is much simpler and covers only the execution, management, and input/output of Web services. The characteristic makes it much easier to integrate the proposed framework with other technologies.

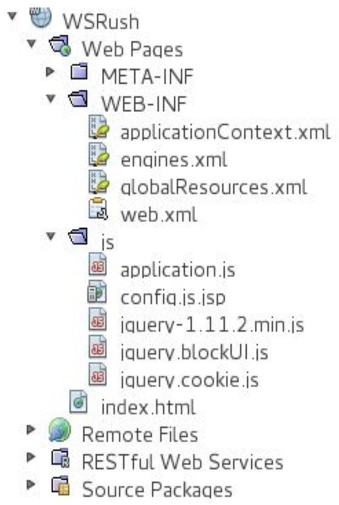
WSRush Platform

The WSRush platform consists of three layers: the core engine and service provider interface, the Web interface, and the Web service utilities. The core engine and service provider interface is the kernel of the platform. It handles the life cycle of the platform, manages the execution of services, and provides resource management utilities. Besides, this part includes application interfaces to allow extensions. The Web interface implements the HTTP layer. A version of Jersey, which is a widely adopted implementation of the Restful protocol, is deployed in this part. A Web service request will first be intercepted by the Web interface and then dispatched to the corresponding services registered in the core engine. Additionally, a Web service utilities layer is included. The layer provides utilities to be used by developers to configure and access the platform. A set of configuration files such as *engines.xml* for registering services and *globalResources.xml* for managing resources are included in this layer. Furthermore, some JavaScript modules are also included to assist developers.

Currently, WSRush is packaged as a standard Java Web application. To use it, simply deploy the package file (a standard zip file with the .war file extension) to a Java-enabled environment with JDK[8] 7.0 or higher and a Java application server supporting Servlet[9] 2.4 or higher. The current implementation of WSRush has been tested against Tomcat 7.0 and Glassfish 4.0, which are two certified Java application servers. After copying the WSRush package file to the application directory of the target application server, the file will be automatically unpackaged, and developers can start developing their applications based on the unpackaged Web application structure. Note that despite that WSRush itself is a Web application, this does not impose restrictions on the types of applications based on WSRush. As long as the Restful protocol is adhered, WSRush can be utilized in various types of applications.

The directory structure of WSRush is illustrated in the figure below:

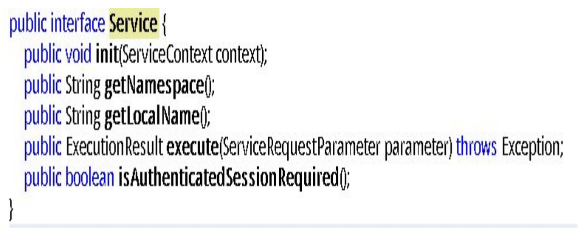
[5] https://en.wikipedia.org/wiki/Spring_Framework
[6] https://en.wikipedia.org/wiki/Project_Jersey
[7] http://rubyonrails.org/
[8] https://en.wikipedia.org/wiki/Java_Development_Kit
[9] http://docs.oracle.com/javaee/5/tutorial/doc/bnafd.html



```
WSRush
  Web Pages
    META-INF
    WEB-INF
      applicationContext.xml
      engines.xml
      globalResources.xml
      web.xml
    js
      application.js
      config.js.jsp
      jquery-1.11.2.min.js
      jquery.blockUI.js
      jquery.cookie.js
    index.html
  Remote Files
  RESTful Web Services
  Source Packages
```

Figure 1. the directory structure of WSRush

To deploy a Web service, one has to at first implement the ilab.apprush.core.Service interface, and then register the Web service in the engines.xml configuration file. The interface is shown in figure2. Note that developers are required to realize the specific application logic of their system only, the Web service protocol itself is handled by the platform.



```
public interface Service {
    public void init(ServiceContext context);
    public String getNamespace();
    public String getLocalName();
    public ExecutionResult execute(ServiceRequestParameter parameter) throws Exception;
    public boolean isAuthenticatedSessionRequired();
}
```

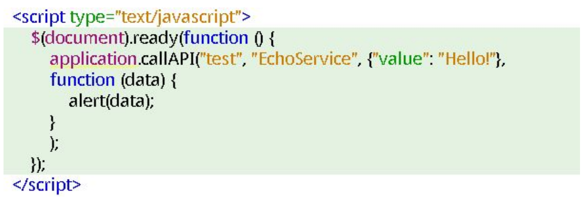
Figure 2. the Service interface

In WSRush, executions of Web service instances are managed by engines. A engine is a software module provides the run time execution environment for its registered Web service instances. Three type of engines are pre-bundled in the package file of WSRush: the SimpleUnsafeEngineImpl, the SimpleUnsafeScriptableEngineImpl, and the SimpleEngineImpl. Developers can implement engines for their own needs by implementing the ilab.apprush.core.Engine interface, but the pre-bundled engines should be enough in most circumstances. The SimpleUnsafeEngineImpl is the most simple one. As its name suggests, the type of engine imposes no security constraint. Hence, services registered with the type of engine is open to all requests provided that the network settings allows it. The SimpleUnsafeScriptableEngineImpl is similar with the SimpleUnsafeEngineImpl with the difference that the type of engine embeds script interpreters. As a result, developers can register script-based Web services with the type of engine. Although the execution of scripts is slower than compiled binaries, scripting-based Web services can be modified on-line and is usually easier to develop. The SimpleEngineImpl accepts both compiled and script based Web services and additionally implements security mechanisms. That is, user name and password can be applied to restrict the access to services.

To invoke Web services, simply issue HTTP requests to WSRush instances. The endpoint of the requests is in the following form:
http://*host*/ws/dispatcher?namespace=*namespace*&localName=*localName*
host, namespace, and localName are parameters. Function parameters to be passed to Web services should be encoded as a JSON[10] map and embedded into the POST packet to be sent. If the application relying on WSRush is a Web application, some JavaScript helper functions are shipped with WSRush to simplify the invocation process. In application.js, which is a JavaScript library included in WSRush, an *ApplicationClass* class is defined. The class provides a method

callAPI(namespace, localName, parameters, successCallback, failCallback)

for Web service invocation. Using the method, developers simply fill the required parameters, and the *successCallback* and *failCallback* will automatically be invoked when needed. Additionally, The method will handle the authentication part if the invoked Web service requires authentication. A very simplified example of utilizing the JavaScript method is shown below:



```
<script type="text/javascript">
  $(document).ready(function () {
    application.callAPI("test", "EchoService", {"value": "Hello!"},
    function (data) {
        alert(data);
    }
    );
  });
</script>
```

Figure 3. a JavaScript example

As a case study, we have implemented a set of Web services to augment a famous e-learning platform, the OLAT e-learning management system[11], with affective learning technologies. To implement the needed functionality, we simply create two Web services via the online editing tool provided by WSRush as two scripting files registered against a implementation of the SimpleUnsafeScriptableEngineImpl. The Web services are automatically published and we simply access the functionality via AJAX calls. During the process, there is almost no need to touch the source code of OLAT, which is convenient.

Design

As stated above, WSRush is composed of three

[10] http://www.json.org/
[11] http://www.olat.org/

parts: the core engine and service provider interface, the Web interface, and the Web service utilities. The core engine and service provider interface layer provides kernel components. At this layer, no Web service protocol is assumed. The design simply focuses on the management of services. Two central concepts are engines and services. The most fundamental methods of engines are defined in the ilab.apprush.core.Engine interface:

1. init: implement this method to initialize the engine
2. start: implement this method to handle the start up of the engine
3. stop: implement this method to handle the shutdown procedure of the engine
4. executeService: implement this method to handle clients' requests of executing services

Implementing all of these methods from scratch can be tedious, so some abstract classes are included to simplify implementation and future extension. Figure 4 shows the class hierarchy of engine classes. As shown in figure 4, there are various abstract engine classes to simplify the implementation. Among them, *AbstractSimpleEngine* and *AbstractScriptableEngine* are two major classes. Developers who want to extend the WSRush platform may want to start from these two classes. *AbstractSimpleEngine* defines how engine classes interact with service classes. It has an internal service map which maintains a list of registered services. Upon receiving a request, it dispatches the request to corresponding services according to the specified namespace and localName parameters. *AbstractScriptableEngine* extends *AbstractSimpleEngine* by incorporating script interpreters, the VMs.

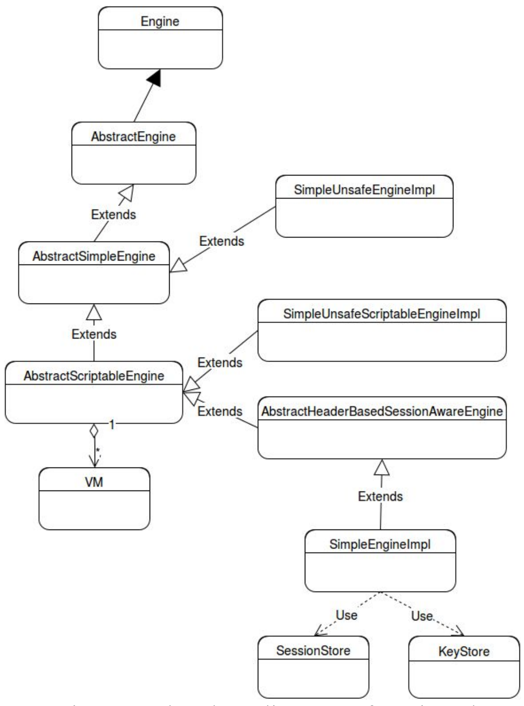


Figure 4. the class diagram of engine classes

The Web interface layer handles HTTP requests. The core class in this layer is the ilab.apprush.core.services.system.DispatcherService class, which is a Restful endpoint implemented with Jersey. *DispatcherService* accepts both GET and POST requests. Original request objects will be wrapped into *WrappedHttpRequest* objects to provide identical access methods for the two types of requests. Additionally, a *SpringConfigurator* class is defined to incorporate configurations loaded via SpringFramework configuration files. The figure below demonstrates the service dispatching flow:

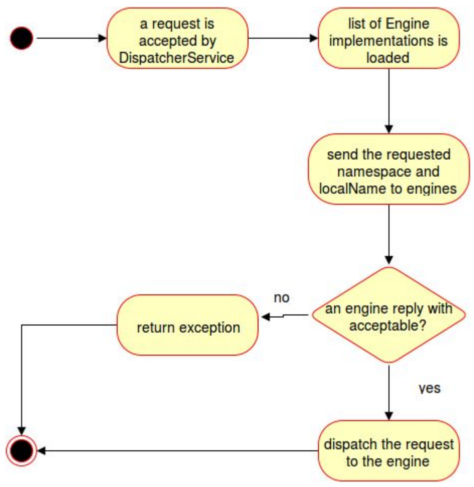

a request is accepted by DispatcherService
list of Engine implementations is loaded
send the requested namespace and localName to engines
an engine reply with acceptable?
no
return exception
yes
dispatch the request to the engine

Figure 5. the activity diagram

Note that in the current implementation, if a service is registered in multiple engines, only the first engine in the list will be contacted.

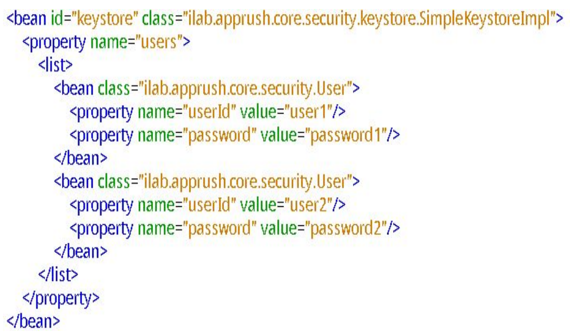
For developers that do not want to extend or modify the core of WSRush, relying on the Web service utilities layer is enough. In addition to the application.js module mentioned earlier, a dynamic configuration file is included. The file will automatically discover the IP address of WSRush server and thus can be used for obtaining the URL for the endpoint. The configuration file is only applicable for Web applications. To use the file, simply use the following statement:

<script src="js/config.js.jsp" type="text/javascript"></script>

Furthermore, for Web applications, an authentication dialog will be generated automatically when needed. To use the feature, a *Keystore* instance has to be registered via the *globalResource.xml* configuration file. The ilab.apprush.core.security.keystore.Keystore interface declares methods for implementing a *Keystore*:

1. getKey: given a (user name, password) pair, the method returns a temporarily valid key
2. invalidateKey: disable the specified
3. verifyKey: check whether the specified key is valid or not

A simplified implementation of *Keystore*, the ilab.apprush.core.security.keystore.SimpleKeystoreImpl is included in WSRush. To use *SimpleKeystoreImpl*, a static user account list is needed. The list can be registered in the *globalResource.xml* configuration file. An example is shown below:



```
<bean id="keystore" class="ilab.apprush.core.security.keystore.SimpleKeystoreImpl">
  <property name="users">
    <list>
      <bean class="ilab.apprush.core.security.User">
        <property name="userId" value="user1"/>
        <property name="password" value="password1"/>
      </bean>
      <bean class="ilab.apprush.core.security.User">
        <property name="userId" value="user2"/>
        <property name="password" value="password2"/>
      </bean>
    </list>
  </property>
</bean>
```

Figure 6. a sample configuration file for *Keystore*

Conclusions and Future Work

In this manuscript, a Web service ready application platform, the WSRush platform, is proposed. The platform provides some ready-to-use infrastructures to simplify the development of Web service applications. With the proposed platform, developers can concentrate on their application logic and simply leave the protocol handling and resource management tasks to WSRush. Furthermore, the platform is highly extensible, developers can customize WSRush to their needs by extending from the core classes.

The current implementation of WSRush is stable and usable. In the future, we will improve WSRush in the following ways:

1. add script online editing and management functionality: WSRush already supports script-based Web services but developers need remote file transferring tools to deploy script-based Web services; in the future, we will implement online editing and management tools to simplify the process
2. graphical user interface: the current version is packaged as a standard Web application, however, it stills lacks GUI for system configuration and management; in the next step, we will augment it with GUI to enhance its usability

Acknowledgements

The paper was prepared in collaboration with my lab members in Department of Information Management, Nanhua University, Taiwan. Also, this research is partly supported by MOST104-2511-S-343 -003.

References

[1] as Erl, T. (2005). Service-Oriented Architecture (SOA) Concepts, Technology and Design.

[2] Erl, T. (2004). Service-oriented architecture: a field guide to integrating XML and web services. Prentice Hall PTR.

[3] Trkman, P., Kovačič, A., & Popovič, A. (2011). SOA adoption phases. Business & Information Systems Engineering, 3(4), 211-220.

[4] Luthria, H., & Rabhi, F. (2009). Service oriented computing in practice: an agenda for research into the factors influencing the organizational adoption of service oriented architectures. Journal of theoretical and applied electronic commerce research, 4(1), 39-56.

[5] Lopez, N., Casallas, R., & Villalobos, J. (2007, May). Challenges in creating environments for SOA learning. In SSDSOA'07: ICSE Workshops 2007, pp. 9-9.

[6] Paik, H. Y., Rabhi, F. A., Benatallah, B., & Davis, J. Service learning and teaching foundry: A virtual SOA/BPM learning and teaching community. In Business Process Management Workshops, pp. 790-805, September 2010.

[7] Spillner, J. (2011, September). SPACEflight—A versatile live demonstrator and teaching system for advanced service-oriented technologies. In Microwave and Telecommunication Technology (CriMiCo), pp. 455-456, 2011.

[8] Hosack, B., Lim, B., & Vogt, W. P. (2012). Increasing student performance through the use of Web services in introductory programming classrooms: results from a series of quasi-experiments. Journal of Information Systems Education, 23(4), 373.