

The Design and Case Study of the WSRush Platform

Chun-Hsiung Tseng¹, Yung-Hui Chen², Yan-Ru Jiang³, Jia-Rou Lin⁴

¹ Department of Communications Engineering, Yuan Ze University, Taiwan

² Department of Computer Information and Network Engineering, Lunghwa University of Science and Technology, Taiwan

³ Department of Information Management, Yuan Ze University, Taiwan

⁴ Department of Information Management, Nanhua University, Taiwan

lendl_tseng@seed.net.tw, cyh@mail.lhu.edu.tw, treelazy821006@gmail.com, 10325209@nhu.edu.tw

Abstract

The popularity of cloud computing makes developing Web services a trend. In this research, we propose our design of WSRush, a platform that can be used to simplify the development of Web service applications. The platform simplifies the development cycle of Web service applications. The WSRush platform consists of three layers: the core engine and service provider interface, the Web interface, and the Web service utilities. With the platform, developers can reuse or inherit from the provided software modules to prevent re-inventing the wheel. Two types of services: the script-based ones and the static ones are supported by WSRush. Furthermore, developers can even inherit from the core of WSRush and build their own platforms.

Keywords: Webservice, SOA, Platform

1 Introduction

The emerging of cloud computing issues causes wide adoption of Web service technologies, which in turn has demonstrated high impacts on software development cycles. Protocols such as SOAP¹ and Restful² are widely used in Web service applications and they do help develop clear separations among functionalities, data models, and user interfaces. Furthermore, a Web service application is by nature platform independent. Despite of its elegance, Web service technologies are still considered difficult for inexperienced programmers. As a teacher offering programming courses, the researcher has tried several methods to make students understand the concept of Web service, however, only few of them can really get familiar with the technology and even fewer of them can utilize it in real world software projects. We believe that it will be beneficial to help students and inexperienced programmers get acquainted with them as soon as possible, considering the importance of Web

service technologies. Therefore, the researchers built 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. During the development of Web service applications, usually, developers will face 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 the marshalling and unmarshalling of input/output data 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.

But what are the core concepts of Service-Oriented-Architecture (SOA)? According to Endrei et al., "Service-oriented architecture presents an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services." [1] Therefore, the goal of the proposed platform is to help users focus on wrapping application functionality as services rather than dealing with the above issues. Of course, there are already tools such as NetBeans, Eclipse, and Visual Studio aimed at helping developers handle the above issues. However, the diversities of these tools cause additional burdens for developers, not to mention the vendor-lock-in problem.

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

*Corresponding Author: Yung-Hui Chen; E-mail: cyh@mail.lhu.edu.tw
DOI: 10.3966/160792642018091905033

¹ <https://en.wikipedia.org/wiki/SOAP>

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

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 themselves. Instead, developers simply inherit from a provided software module (in the current stage, the platform is implemented in Java, so developers have to extend 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. Due to the popularity of the JSON-format, this characteristic should not cause limitations.

(3) The platform provides 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 jQuery³, Spring Framework⁴, and Jersey⁵, 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 programmers and students manage the Web service technology. Besides, experienced developers will also benefit from the platform with its well integrated set of technologies since they can focus on the business logic part. In this research, we will at first give a detailed introduction to the functionality and architecture of the platform.

2 Related Works

The importance of the service oriented architecture (SOA) can never be overestimated. The definition of SOA is explained in detail in [2]. Web service standards are the central idea of SOA. In [3], Web services are defined as the integration technology preferred by organizations implementing service-oriented architectures. The adoption of SOA is extremely wide. Trkman and Kovačić discusses the phases of SOA adoption [4]. The research of Luthria and Rabhi [5] 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 [6].” The work of Paik, Rabhi, Benatallah, and Davis designed a dedicated virtual teaching and learning space for students to learn SOA [7]. Spillner implemented SPACEflight, which is a versatile live demonstrator and teaching system for advanced service-oriented technologies [8]. Furthermore, it is proved that using Web services in introductory programming classes can enhance students’ learning performance [9]. In [10], a framework for measuring student learning gains and engagement in a Computer Science 1/Information Systems 1 programming course was described. The results of the research showed that adopting Web services is effective in student learning gains. In [11], how and why teaching modern web services is an important part of information systems curriculum, and how they can be introduced at introductory levels were discussed. According to the research result, Knowing the workings of Web APIs is quickly becoming a vital skill of programmers, tech entrepreneurs, and managers of any kind. The work of Jakimoski highlighted that in education systems, SOA can be adopted to reduce the complexity of integrating systems, but there is limited academic research work on the domain [12]. Frameworks such as Spring Framework⁶ 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. Note that this does not mean that WSRush lacks important features and imposes huge limitations on applications. WSRush is itself built based on Spring Framework, so it in fact can achieve everything Spring Framework can achieve. Our point is, for circumstances in which developers don’t need advanced functionality such as dependency injection, interceptors, and controllers, adopting WSRush will be a lean solution. A more comparative framework is JAX-WS⁷. The framework is also a lightweight framework and covers only the Web service part. In fact, the current implementation of WSRush is built based on JAX-WS. The issue of JAX-WS is that it exposes too many details and offers too many degree

³ <https://en.wikipedia.org/wiki/JQuery>

⁴ https://en.wikipedia.org/wiki/Spring_Framework

⁵ https://en.wikipedia.org/wiki/Project_Jersey

⁶ <https://projects.spring.io/spring-framework/>

⁷ <https://jax-ws.java.net/>

of freedom to ordinary developers. For example, JAX-WS allows developers to specify output formats via the `@Produce` annotation while in most cases the JSON format is assumed. Frameworks such as Zend Framework⁸ and Ruby on Rails⁹ take a different approach. They focus more on the adoption of the MVC design pattern and they provide some very advanced model construction tools. They are usually regarded to as full-stack frameworks that are intended to be used alone while WSRush is designed to allow easy integration with other technologies. Other frameworks, such as DeployR¹⁰ and Shiny¹¹ are special purpose frameworks. They are designed for integrating the R technology into Web applications while WSRush is a general purpose framework. Hayes et al. proposed a platform-as-a-service method for building Web services but only focused on healthcare [13].

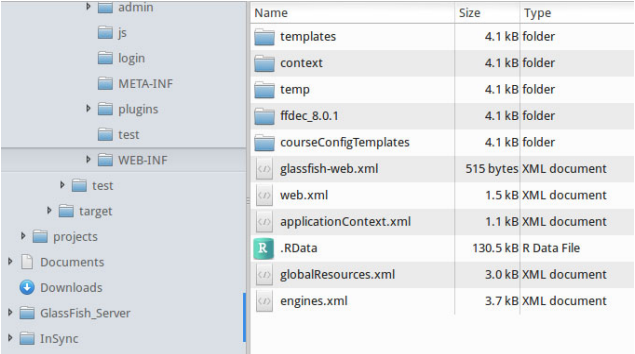
3 WSRush Platform

The WSRush platform is a framework and tool set to make develop Web service based applications simpler. It has a core library which can be easily extended to address different environments and protocols, a framework which can be used for implementing business logic for Web services, and some tools to simplify the design and deploy tasks. The 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 and HTTPS processing 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.

The current version of 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 7.0 or higher and a Java application server

supporting Servlet 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:



Name	Size	Type
templates	4.1 kB	folder
context	4.1 kB	folder
temp	4.1 kB	folder
ffdec_8.0.1	4.1 kB	folder
courseConfigTemplates	4.1 kB	folder
glassfish-web.xml	515 bytes	XML document
web.xml	1.5 kB	XML document
applicationContext.xml	1.1 kB	XML document
.RData	130.5 kB	R Data File
globalResources.xml	3.0 kB	XML document
engines.xml	3.7 kB	XML document

Figure 1. the directory structure of WSRush

To deploy a Web service, one has to at first implement the Service interface, and then register the Web service in the `engines.xml` configuration file. The interface is shown in Figure 2. 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)
    public boolean isAuthenticatedSessionRequired();
}
```

Figure 2. the Service interface

The code snippets below demonstrates the implementation of a simple echo Web service:

```
public class EchoService extends AbstractService{
    @Override
    public ExecutionResult execute(ServiceRequestParameter parameter)
        throws Exception {
        return this.createExecutionResult(null, true,
            parameter.getParameter("value"));
    }

    @Override
    public boolean isAuthenticatedSessionRequired() {
        return true;
    }
}
```

⁸ <https://framework.zend.com/>

⁹ <http://rubyonrails.org/>

¹⁰ <https://msdn.microsoft.com/en-us/microsoft-r/deployr-about>

¹¹ <https://shiny.rstudio.com/>

As shown in the code snippets, developers has to extend from *AbstractService*, which handles the fundamental operations such as parameter marshaling and protocol decoding/encoding of a Web service, and then override the *execute* and *isAuthenticated Session Required* methods. In the *execute* method, developers implement the business logic. The *parameter* argument wraps the parameters in HTTP packets. The invocation of the *createExecution* method will create the return value adhering to the format defined by 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 Simple Unsafe Engine Impl, the Simple Unsafe Scriptable Engine Impl, 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 support. Hence, services registered with the type of engine is open to all requests provided that the network settings allows it. The *Simple Unsafe Scriptable EngineImpl* is similar with the *Simple Unsafe Engine Impl* with the difference that the type of engine supports script languages. 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 provides security mechanisms. That is, user name and password can be set 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 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, success Callback, 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", "Echo", {"value": "Hello!"},
      function (data) {
        alert(data);
        console.log(data);
      }
    );
  });
</script>
```

Figure 3. a JavaScript example

4 Design

WSRush is composed of three layers: the core engine and service provider interface, the Web interface, and the Web service utilities. The main considerations are extensibility and separation of concerns. The core engine and service provider interface defines most core functionalities. The Web interface layer deals with I/O between WSRush and clients via the HTTP protocol. The Web service utilities layer defines utility classes that can be used by other layers.

4.1 The Core Engine and Service Provider Interface Layer

The core engine and service provider interface layer provides the definition and implementation of core components. These core components form the basis of the WSRush platform and make the platform extensible and flexible. At this layer, no communication protocol is assumed. The design focuses on the management of services. To achieve this, two central concepts are *Engine* and *Service*. A *Service* is an implementation of a set of application logics. In most cases, if a developer is using a concrete implementation of WSRush, she/he only focuses on writing services. On the other hand, an *Engine* is responsible for providing an execution environment for services managed by it. Therefore, for developers who are extending WSRush or implementing their own version of WSRush, they have to deal with the engine part. The most fundamental methods of engines are defined in the *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

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.

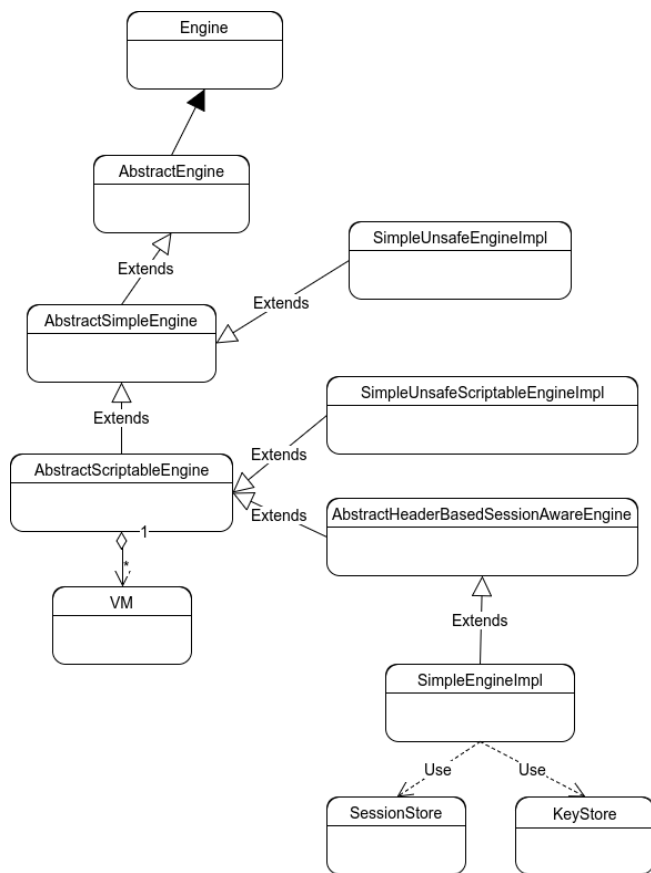


Figure 4. the class diagram of engine classes

Some additional classes deserve explaining are VM, *DaemonTask*, and *Context*. VM implements the concept of a virtual computing machine. Implementations of *AbstractScriptableEngine* delegate their computation tasks to VMs. A VM wraps one or more scripting interpreters and provides a supporting infrastructure for these interpreters. Currently, JavaScript and Beanshell scripting languages are supported. With the design of VMs, developers can dynamically change the implementation of their services and perform hot deployment, which will further shorten the development cycle. The definition of VM has the

following methods:

- (1) *getVMType*: return the scripting language supported
 - (2) *getRootFolder*: return the root folder (the working directory) of the VM
 - (3) *executeScriptAsService*: execute the service
- Currently, *DefaultJavaScriptVMImpl* and *DefaultBeanShellVMImpl* are provided to support JavaScript and BeanShell respectively.

A *DaemonTask* is a background task that will be automatically managed by WSRush. Unlike traditional *java.lang.Thread*, which may left in system after the termination of the main program and lock some resources, a *DaemonTask* will automatically be terminated once the main program is finished. To implement a *DaemonTask*, one simply implements the interface *ilab.apprush.core.daemon.DaemonTask* or extends from the abstract class *ilab.apprush.core.daemon.AbstractDaemonTask*. Furthermore, unlike traditional *java.lang.Thread*, a *DaemonTask* can be shutdown programatically. To start a *DaemonTask*, one has to invoke *GlobalContext.requestDaemonExecution*. Besides, a convenient function, *GlobalContext.terminateDaemons* is provided to shutdown all *DaemonTask*.

A *Context* encapsulates a set of environmental parameters. *Services* are executed by WSRush, and therefore, they have to acquire environment information such as working directory from WSRush. Three types of *Contexts* are defined: *GlobalContext*, *EngineContext*, and *ServiceContext*. A running *Service* has reference to its own *ServiceContext*. From the context, it can obtain the directory structure relative to the root path (the deployed path) of the *Service*. Furthermore, it can acquire the corresponding *EngineContext* and *GlobalContext* from the *ServiceContext*. As their names suggest, *EngineContext* provides engine-wide information while *GlobalContext* provides global information. Additionally, by invoking the *executeService* method defined in all types of *Contexts*, a *Service* can even request the execution of other services, which will make code-reuse easier.

4.2 The Web Interface Layer

The Web interface layer handles HTTP and HTTPS requests. The current implementation of this layer assumes the adoption of the Java Servlet¹² technology. The popularity of the technology makes the assumption reasonable. The core class in this layer is the *ilab.apprush.core.services.system.DispatcherService* class, which is a Restful endpoint implemented with Jersey. Jersey is the Java implementation of the Restful protocol and allows users to create Restful style Web services by simply annotating a Java class. In WSRush, Jersey is adopted for implementing the front controller of *Services*. That is, *DispatcherService* is responsible

¹² <http://www.oracle.com/technetwork/java/index-jsp-135475.html>

for accepting Restful requests and wrap/unwrap parameters in incoming and outgoing packets. For convenience, *DispatcherService* accepts both GET and POST requests. Note that since WSRush is packaged as a software library, developers who needs detailed control can simply extends *DispatcherService*. Original request objects will be wrapped into *WrappedHttpRequest* objects to provide identical access methods for the two types of requests. To allow flexible configurations, this layer also assumes the adoption of the spring framework¹³. A sample configuration file is listed below:

```
<beans>
  <import resource="globalResources.xml"/>
  <import resource="engines.xml"/>
  <bean id="springConfigurator" class
    =
    "ilab.apprush.core.configurator.SpringConfigurator"
  >
    <property name="engineList">
      <list>
        <ref bean="simpleEngine1"/>
        <ref bean="simpleScriptableEngine1"/>
      </list>
    </property>
  </bean>
</beans>
```

In the configuration file, two engines, the *simpleEngine1* and the *simpleScriptableEngine1*, are loaded. By modifying the list, developers can decide the engines to be loaded into their WSRush instances. Furthermore, two additional configurations, the *globalResources.xml* and the *engines.xml*, are included. The former is used for configuring global resources such as connection pools while the latter is for configuring engines. An example of *globalResources.xml* is shown below:

```
<beans>
<bean id="defaulttdb" class="com.mchange.v2.c3p0.ComboPooledDataSource" destroy-method="close">
  <property name="driverClass"
    value="com.mysql.jdbc.Driver" />
  <property name="jdbcUrl"
    value="jdbc:mysql://localhost/nhu_ccs?characterEncoding=UTF-8"
  />
  <property name="initialPoolSize" value="1" />
  <property name="minPoolSize" value="1" />
  <property name="maxPoolSize" value="5" />
  <property name="maxIdleTime" value="600" />
  <property name="user" value="xxx" />
  <property name="password" value="xxx" />
</bean>
</beans>
```

The file declares a connection pool named as *defaulttdb* with some pooling configurations. The pool allows 5 concurrent database connections at max and will keep at least 1 connection stay in the pool.

Additionally, a *SpringConfigurator* class is defined to incorporate configurations loaded via Spring

Framework configuration files. Developers can refer to the injected instance of *SpringConfigurator* to access the configured objects. The figure below demonstrates the service dispatching flow:

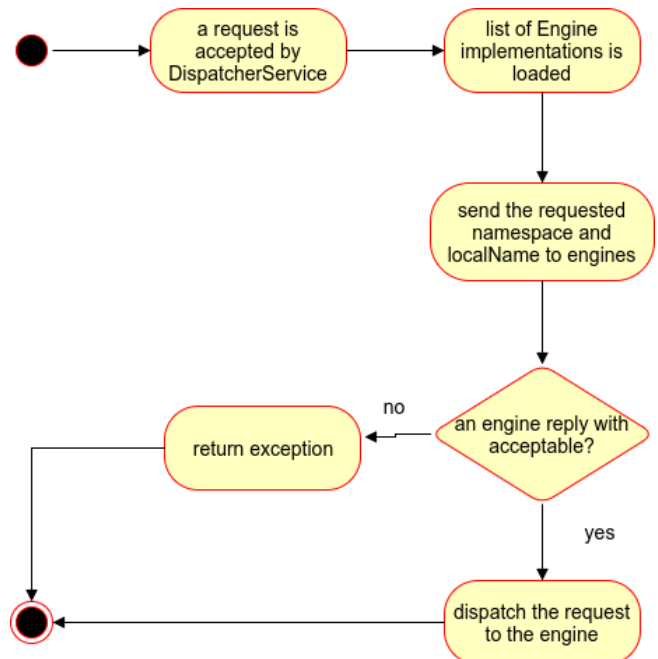


Figure 5. the activity diagram

4.3 The Web Service Utilities Layer

For developers that do not want to extend or modify the core of WSRush, they can simply deploy the packaged version of WSRush. By default, WSRush is most suitable for Web applications and a Web service utilities layer is implemented to simplify the adoption. 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. With the dynamic configuration file, it will be easier to maintain a WSRush instance. 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

¹³ <https://projects.spring.io/spring-framework/>

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"/>
        <property name="group" value="user"/>
      </bean>
      <bean class="ilab.apprush.core.security.User">
        <property name="userId" value="user2"/>
        <property name="password" value="password2"/>
        <property name="group" value="user"/>
      </bean>
      <bean class="ilab.apprush.core.security.User">
        <property name="userId" value="admin"/>
        <property name="password" value="admin"/>
        <property name="group" value="admin"/>
      </bean>
    </list>
  </property>
</bean>
```

Figure 6. a sample configuration file for Keystore

within two months. Moreover, some Web based tools have been provided for administrating a WSRush instance. To enter the administration page, users have to log into the following url:

http://host:port/WSRush/admin

Then, the user will be requested to enter the administration username/password. After a successful login, the user will be redirected to an online file manager:

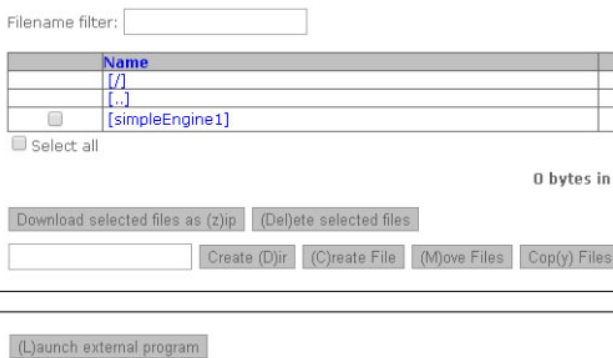


Figure 7. the online file manager of WSRush

Note the file manager is extended from an open source project named as jsp File Browser¹⁴. With the file manager, users can edit script-based services online. The index page of the file manager lists the directory structure of script-based engines. By navigating into the corresponding folder, users can create/modify/delete/show script-based services for specific engines. The online editor is shown below:

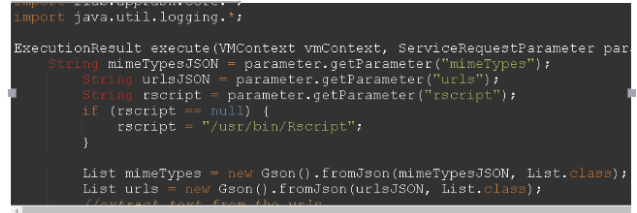


Figure 8. the online script-based service editor

4.4 The Composition of Web Services

In many cases, developers have to reuse or compose existing Web services to achieve their tasks. The current implementation of WSRush does not support automatic composition. However, manual composition is supported. In ilab.apprush.core.context.Context, a method:

ExecutionResult executeService (String namespace, String localName, ServiceRequestParameter parameter) is defined. Developers simply invoke the function to interact with other Web services registered in the same WSRush instance. Note that WSRush itself keeps a very lean design principle, so it can be integrated with other orchestration or choreography technologies such as BPEL.

5 Case Study

In this section, a case study is given to demonstrate the adoption of WSRush. The research team adopts WSRush in their own MOST project: “Cloud and Crowd Supported Math Learning in Computer Science.” In the project, we develop a method as well as an e-learning system based on cloud technologies and crowd intelligence to enhance students’ learning performance of math in computer science. As mentioned, we use WSRush to implement Web services to augment the OLAT learning management system. With WSRush, we can prototype the needed services quickly by using the beanshell and javascript scripting languages. Furthermore, team members don’t need to study full-stack Web service frameworks such as JAX-WS in advance. These characteristics result in faster development. First, for version control, we created a brached version of WSRush. Then, to load WSRush related objects, we add the default WSRush listener via the standard web.xml configuration file:

```
<listener>
<listener-class>ilab.apprush.core.listener.DefaultStartupListener</listener-class>
</listener>
```

To prevent the need of redeploy the *Services*, script-based *Services* were preferred. On the other hand, there were still *Services* that will not be affected by experiments results, so static *Services* were used in such cases for performance. Additionally, we added two entries to globalResources.xml:

¹⁴ <http://www.vonloesch.de>

```
<bean id="globalParameters" class="java.util.HashMap">
  <constructor-arg>
    <map key-type="java.lang.String" value-type="java.lang.Stri
      <entry key="audioUploadPath" value="/var/APPRush/audios
    </map>
  </constructor-arg>
</bean>
<bean id="properties" class="ilab.wsrush.mathws.Properties">
  <property name="properties">
    <map>
      <entry key="olatResourceFolder" value="/tomcat/temp/ola
    </map>
  </property>
</bean>
<bean id="olatdb" class="com.mchange.v2.c3p0.ComboPooledDataSource"
  <property name="driverClass" value="com.mysql.jdbc.Driver" />
  <property name="jdbcUrl" value="jdbc:mysql://localhost:3306/olat
  <property name="initialPoolSize" value="1" />
  <property name="minPoolSize" value="1" />
  <property name="maxPoolSize" value="5" />
  <property name="maxIdleTime" value="600" />
  <property name="user" value="olat" />
  <property name="password" value="olat" />
</bean>
```

The former entry, the *properties* bean was used for specifying olat specific environment variables while the latter entry was used for registering a database connection pool to the olat database. Finally, we changed its Web application name to MathWS and deploy the resulting war file to a tomcat server.

We totally implemented 7 Web services. Among them, two are script-based and four are static. The following code snippets demonstrate a script-based service:

```
import ilab.apprush.core.scripding.*;
import ilab.apprush.core.*;
import imsofa.nhu.rjbridge.RJ;
import java.io.*;

ExecutionResult execute(VMContext vmContext, ServiceRequestParameter
String K1=parameter.getParameter("K1");
String K2=parameter.getParameter("K2");
String K3=parameter.getParameter("K3");
String rscript=parameter.getParameter("rscript");

if(rscript==null){
  rscript="/usr/bin/Rscript";
}

RJ rj=new RJ(rscript);
rj.appendRCode("library(rpart)");

File webINF=new File(vmContext.getGlobalContext().getRealPath("/
java.util.logging.Logger.getLogger(this.getClass().getName()).in
String path=webINF.getAbsolutePath();
rj.appendRCode("setwd("+path+")");
rj.appendRCode("load('RData')");
rj.appendRCode("d<-as.data.frame(list(K1n="+K1+", K2n="+K2+", K3
rj.appendRCode("pc-predict(fit, d)");
rj.appendRCode("pL<-p[1]");
rj.appendRCode("pH<-p[3]");
rj.setReturnedObjects(new String[]{"pL", "pH"});
rj.run();
return vmContext.getExecutionResultFactory().createExecutionResu
}
```

As shown in the code snippets, to implement a script-based service, developers have to complete two functions: *execute* and *isAuthenticatedSessionRequired*. The former defines the business logic of the service while the latter controls whether an authenticated session is required for accessing the service. In the *execute* function, *VMContext* plays a similar role with *ServiceContext* and *Service Request Parameter* is a wrapper of parameters passed by the program that invokes the service.

On the other hand, implementing a static service takes a very similar approach:

```
public class EmotionSelectService extends AbstractService {

@Override
public ExecutionResult execute(ServiceRequestParameter parameter)
final String course = parameter.getParameter("course");
final String user=parameter.getParameter("user");
final String emotion=parameter.getParameter("emotion");
final Date date=new Date();

SQLExecutor.executeWithConnection(this.context.getGlobalContext
@Override
public void execute(Connection conn) throws Exception {
PreparedStatement pstmt=conn.prepareStatement("insert
pstmt.setTimestamp(1, new Timestamp(date.getTime()));
pstmt.setString(2, course);
pstmt.setString(3, user);
pstmt.setString(4, emotion);
pstmt.executeUpdate();
}
});
return this.createExecutionResult(UUID.randomUUID().toString()
}
}
```

In most cases, developers start by extending the *AbstractService* and override two function: *execute* and *isAuthenticatedSessionRequired*. With the help of WSRush, students joining the project created all the needed Web services within two months. Empirically, most students are not good at server-side programming, not to mention Web service programming. For comparison, a course for JAX-WS on udey takes 12.5 hours¹⁵, which will be mapped to a 3-5 weeks course unit in universities.

6 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) 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.
- (2) Further extend WSRush to support other usecases. For example, the research teach already has a plan to adopt WSRush to build a wrapper of Cordova, which is a very popular APP development framework.

¹⁵ <https://www.udemy.com/java-web-services/>

References

- [1] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, M. Luo, T. Newling, *Patterns: Service-oriented Architecture and Web Services*, IBM Corporation, 2014.
- [2] T. Erl, *Service-Oriented Architecture (SOA) Concepts, Technology and Design*, Prentice Hall, 2005.
- [3] T. Erl, *Service-oriented Architecture: A Field Guide to Integrating XML and Web Services*, Prentice Hall, 2004.
- [4] P. Trkman, A. Kovačić, A. Popović, *SOA Adoption Phases, Business & Information Systems Engineering*, Vol.3, No. 4, pp. 211-220, July, 2011.
- [5] H. Luthria, F. Rabhi, 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*, Vol. 4, No.1, pp. 39-56, April, 2009.
- [6] N. Lopez, R. Casallas, J. Villalobos, *Challenges in Creating Environments for SOA Learning*, SSDSOA'07: ICSE Workshops 2007, Minneapolis, MN, 2007, pp. 9-9.
- [7] H. Y. Paik, F. A. Rabhi, B. Benattallah, J. Davis, Service Learning and Teaching Foundry: A Virtual SOA/BPM Learning and Teaching Community, *Business Process Management Workshop: BPM 2010 International Workshops and Education Track*, Hoboken, NJ, 2010, pp. 790-805.
- [8] J. Spillner, SPACEflight: A Versatile Live Demonstrator and Teaching System for Advanced Service-oriented Technologies, *Microwave and Telecommunication Technology (CriMiCo)*, Sevastopol, Ukraine, 2011, pp. 455-456.
- [9] B. Hosack, B. Lim, W. P. Vogt, 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*, Vol. 23, No. 4, pp. 373-384, December, 2012.
- [10] B. Lim, B. Hosack, P. Vogt, Student Assessment of Learning Gains in an Introductory Computing Course Integrated with Web Services Technology, *Engineering and Technology (ICCIET'2014)*, Phuket, Thailand, 2014, pp. 18-20.
- [11] T. Olsen, K. Moser, Teaching Web APIs in Introductory and Programming Classes: Why and How, *SIGED: IAIM Conference*, Boston, MA, 2013.
- [12] K. Jakimoski, Challenges of Interoperability and Integration in Education Information Systems, *International Journal of Database Theory and Application*, Vol. 9, No. 2, pp. 33-46, June, 2016.
- [13] G. Hayes, H. Khazaei, K. El-Khatib, C. McGregor, J. M. Eklund, Design and Analytical Model of a Platform-as-a-service Cloud for Healthcare, *Journal of Internet Technology*, Vol. 16, No. 1, pp. 139-149, January, 2015.

Biographies



Chun-Hsiung Tseng received his B.S. in computer science from National ChengChi University, and received M.S. and Ph.D. in computer science from National Taiwan University. He is a faculty member of Department of Communications Engineering, YuanZe University. His research interests include big data analysis, crowd intelligence, e-learning systems, and Web information extraction.



Yung-Hui Chen is an associate professor in Department of Computer Information and Network of LungHwa University of Science and Technology. He received the B.S. degree in Computer Science Information Engineering from Tamkang University in 1997, and the M.S. and Ph.D. in Information Engineering from the TamKang University.



Yan-Ru Jiang received her bachelor degree from department of information management in Nanhua university. She is specialized in programming and is a member of the IMSofa lab.



Jia-Rou Lin is currently studying in Nanhua university and is majored in Information Management. She is specialized in programming and is a member of the IMSofa lab.

