ServiceEnabler - Accessing Web Sites as Web Services

Andreas Hubmer (0525780)
Technical University of Vienna

andreas.hubmer@student.tuwien.ac.at

We describe a system that is capable of turning web sites into web services from the outside. It is controlled by simple configuration files that describe how a web application is used by humans. This high-level description is used to define a web service and its operations. On invocation they simulate a browser and the user interacting with a web page and retrieving information. Conformance to standards like WSDL and SOAP ensures that every client can use the ServiceEnabler system.

Contents

1 Introduction

The World Wide Web contains millions of web sites that provide a massive amount of information. It can be seen as the hugest database on earth. But compared to relational databases the Internet is missing structure. It was designed to be read by humans and so HTML, the language of the web, is still a mixture of content and style. Strict separation of content and style is possible with XML and XSLT, but still rarely used.

Web services are the current way of sharing information and services among information systems. They provide structured access and a defined behaviour. But there exist many web applications that will not be updated any more and that provide their information only through HTML pages.

Such legacy web applications may be old but still provide interesting information. To programmatically access such web applications it is necessary to simulate a web browser. Simulation can consist of sending requests, parsing HTML, sending forms and maybe even running Java Script. This is a complicated task and different for every web page.

To assist in accessing legacy web applications we developed ServiceEnabler. ServiceEnabler is a wrapper around web applications that provides structured access to them. It converts web applications into web services. Simple configuration files determine what parts of a web application and how they are exposed as a web service. ServiceEnabler takes the burden of simulating a web browser and performs the necessary requests to get the appropriate pages. It parses the pages, retrieves the information that is asked for and provides the results via a web service. A client application only has to know how to call a web service and does not need to interact with any HTML pages.

2 Requirements

The following sections describe the required capabilities of the ServiceEnabler system.

2.1 Configuration

The ServiceEnabler system shall be configured by simple XML files. A configuration file defines the name of a service and the methods it provides. A method is again identified by a unique name within the service. It consists of a list of actions that are later applied sequentially to a web page. A method may require parameters and provide results that are both defined in the configuration file.

An XML schema definition should document the structure of such a configuration file.

2.2 Runtime Engine

The runtime engine has to provide the core functionality of executing the methods defined in the configuration. It is able to interact with web pages and retrieve information out of web pages (single values and also groups of data). Because web sites do not always obey the HTML specification the runtime engine has to properly deal with invalid HTML pages. It should not deny working with invalid pages but try to be as fault-tolerant as web browsers are.

2.3 Web Service

The ServiceEnabler system needs a web service interface that the client can use to access the services of the runtime engine. This web service shall support the SOAP protocol and provide a WSDL that describes itself. It should be comprehensible how a specific configuration leads to a web service interface. The web service allows to execute methods of the runtime engine and returns the provided results.

2.4 Session Management

Many web sites require a login before any action can be taken. This means that whatever the ServiceEnabler system shall do on such a web site, first it has to perform the login (e.g.: entering identifier and password, clicking the login button). The login sequence is always the same and is normally valid for a period of at least some minutes. The ServiceEnabler system shall be able to reuse a session in order that it saves time when a login is required a second time while the first login is still valid.

2.5 Caching

Some results of a specific service can be valid for a longer time and therefore the ServiceEnabler system shall provide caching of results. It shall be possible to define the caching behaviour in the configuration of each service.

Going further: An automatic update of outdated results by a background process would provide cached results at every time.

2.6 Validation

A service may become invalid because the underlying web page has changed. The page can be (re)moved and the page design can be modified. A mechanism is needed that checks the validity of a page and marks a service as invalid in case of a validation error.

3 Usage

To use the ServiceEnabler system first the desired services have to be defined. Then the ServiceEnabler system can be run and the underlying web sites can be accessed as web services.

3.1 Defining a service

A service corresponds to a web site and contains several methods than can be applied to this web site. The definition of a service also states what the resulting web service looks like. Each service is configured in a separate XML file and all the configuration files of the different services should be put in the same directory. A service is composed of methods that are defined as a sequence of actions.

Appendix A presents the structure of a service definition as XML schema definition. This section provides a textual description of the schema definition and some additional constraints that are not included in the schema definition. They are not included in the schema definition because they can not (easily) be expressed as a schema definition.

Whenever an interaction (a click, entering text, ...) with a web page is necessary the interactive element on the page has to be identified somehow. This is achieved by XPath expressions. An example for identifying the title of a web page is the XPath expression "/html/head/title".

Many interactions need additionally an argument (e.g.: the text to be entered). Sometimes this is a constant value and sometimes the client shall be able to set it. If it is a constant value then this constant value just has to be defined. Otherwise a name of this variable argument has to be defined. The name shall help a client to find out what argument he has to pass.

- `service`: A service needs a unique name, can have a login method and contains a set of methods. The name is used to build the URL at which it can be contacted. If a login method exists it is normally executed before any invocation of any of the normal methods and its resulting HTML page is used by them.
- `login`: The login method defines a starting point for the methods of a service. All the other methods (except those defining an own "startUrl") use the login method as a starting point and operate on the last page of the login method. The login method can be simple and only define a start URL or it can be more complex and consist of several actions. A typical login method would navigate to the login page of a web site, enter user credentials and click the login button.

  For the login method a "startUrl" has to be specified. It indicates at what URL it is starting. Its list of actions defines what actions should be carried out to perform the login. Important is that none of the actions must expect parameters and none of them must provide results. The "validTime" attribute specifies how long the resulting HTML page of a login is valid and can be reused by subsequent calls to methods. It should correspond to the time that a session is valid on the used web site.

- `method`: A method can be called through the web service and defines the actions that constitute its behaviour. It needs a unique name among all the methods of the service. Before a method is actually performed, normally the login method is executed. This can be prevented by specifying a "startUrl" for a method. Then the login method is not executed beforehand and instead the method starts at this URL. The "validTime" attribute specifies how long the results of a method invocation with certain parameters are cached. This means that if a method is called a second time with the same parameters and the "validTime" has not passed then the cached results are returned without any of the actions being invoked.
- `click`: This action simulates a click onto an element of a web page. The element is identified by its XPath.
- `puttext`: The puttext action simulates entering some text on a web page. An XPath determines where the text should be entered. This has to be either an HTML input field (`<input>`) or an HTML text area (`<textarea>`). The text to be entered is either a constant value or a variable value.
- `select`: This action simulates the selection of an item in a list, of a radio button or of a check box. The selection element is identified by its XPath. The value can be constant or variable. If the selection element is a list (`<select>`) then the value has to be equal to the value (and not to the content) of one option (`<option>`) of the list. If the selection element is a radio button or a check box and the value equals to "true" (no matter whether upper or lower case) then the element is selected. Otherwise it is deselected.
- `getvalue`: The getvalue action defines a value that should be returned to the caller of the web service. An XPath denotes what part of an HTML page should be returned and the "name" attribute specifies the name of this result.
- `gettable`: This action can be used to retrieve a table of data items of an HTML page. It consists of at least one column and each column is described by a name and an XPath that defines the elements of its column. If there are more columns than one than the number of elements of each column have to be equal. Otherwise it would be impossible to build a proper table.

- `verify`: The verify action can be used to check whether the current web page is the one you expect and that it has the structure you expect. It verifies that at the given XPath exactly one element can be found and that the text of this element meets the expectations.
- `repeat`: This action works like a loop and repeats a sequence of actions. The "count" attribute determines the number of repetitions.

3.2 WSDL Mapping

Each service that is defined within a configuration file is mapped to an independent WSDL service. For each service the WSDL is published at the URL "http://<HOST>:<PORT>/axis2/services/<SERVICE-NAME>?wsdl".

The methods of a service are mapped to SOAP operations. For each action with a variable argument within a method a parameter is added to the operation. That means that the argument name should help to find out the meaning of a variable.

Results of an operation are defined by the use of `getvalue` and `gettable` actions within a method. Each of these actions form another part of the result message.

3.3 Running ServiceEnabler

The ServiceEnabler system is provided as a "jar" archive and a set of libraries that are needed. The command to start it may look like this example: `java -jar path/to/se.jar`. If no arguments are given then the web service is bound to port 8080 and looks for service definitions in the directory "services" (relative to the working directory). The service is bound to all network interfaces and can be used by everyone that has TCP/IP access to the server machine. A different port and a different configuration directory can be defined by giving them as command line arguments.

Example: `java -jar path/to/se.jar 8081 my-services`

3.4 Accessing a service

A list of the deployed services is provided as HTML page and can be reached by pointing a web browser to the address "http://localhost:PORT". Each service has its own WSDL that describes the possible operations, parameters and arguments in detail. Any SOAP compatible client can be used to make use of the operations.

3.5 Further information

JavaScript is disabled by default because it is most times not necessary and requires a lot of computing power. Cookies are supported. Cached data is stored in a hidden sub-directory of the home directory, called ".serviceen-abler".

4 Technical Description

This section explains the architecture of the ServiceEnabler system and provides information to understand the source code and be able to modify it. Figure 1 shows an overview of the underlying architecture. The components of the ServiceEnabler system are coloured in different shades of blue, the external actors in other colours

To differentiate between Java methods and methods in the language of the ServiceEnabler system (represented by the class `Method`), we will always talk of functions in this section when referring to Java methods.

4.1 Reading XML configurations

To read in the XML configuration files XStream [8] is used. XStream is a simple XML serialiser/deserialiser that does not provide validation. Therefore the XML documents are validated before to the service schema definition (see appendix A). And the serialisable objects validate themselves too.

The `XmlSerialization` class provides a function to load all XML files of a directory and deserialize them into service, method and action objects. Finally they are grouped into a service group.

4.2 Services, Methods, Actions and Results

Figure 2 shows the interesting parts of the modelling of services. A service group object is the container of the different services and additionally manages the caching system.

Each service has an independent cache to manage sessions and store method results. Further it provides the function `callMethod()` to call any of its methods. This is used by the message receiver that has to specify the name of the method and its parameters. Before actually calling the method this function has a look at the cache whether it is really necessary to execute the method, and executes the login method (if needed).

A method provides a run function that iterates its actions and executes them. It can specify that it does not need a login. That would mean that it uses its own start URL and the service would not have to call the login method before executing the method. The web service needs to know the parameters a method expects and the number and type of results. Therefore the method class includes functions that provide this information.

In an analogous manner every action has a function giving the names of all the variables it needs and a function giving descriptions of all its results. The important function the perform function that executes an action and applies the action onto a method context. Applying an action onto a method context means that its page may be changed, variables may be updated and new results may be added.

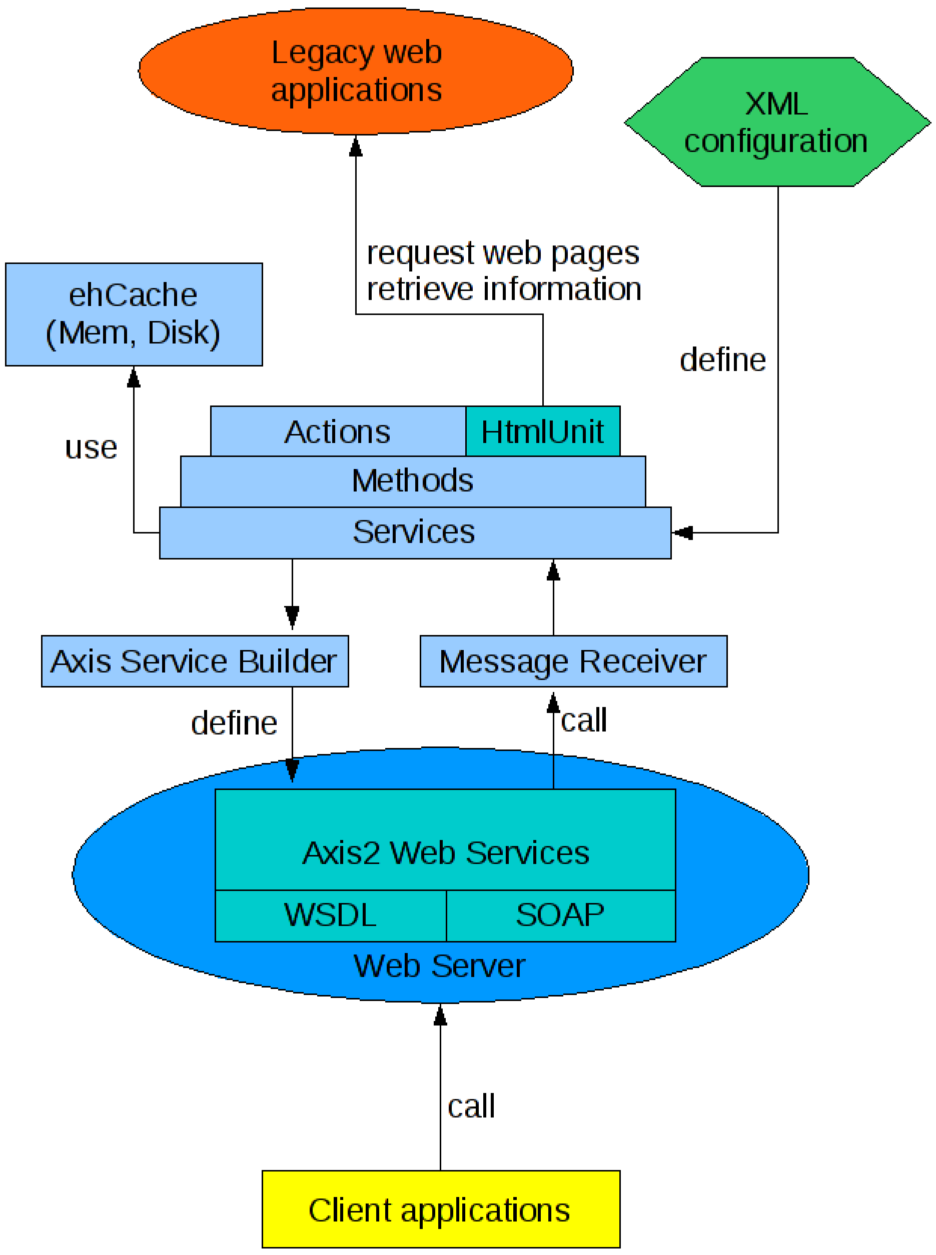

Figure 1: Architectural overview

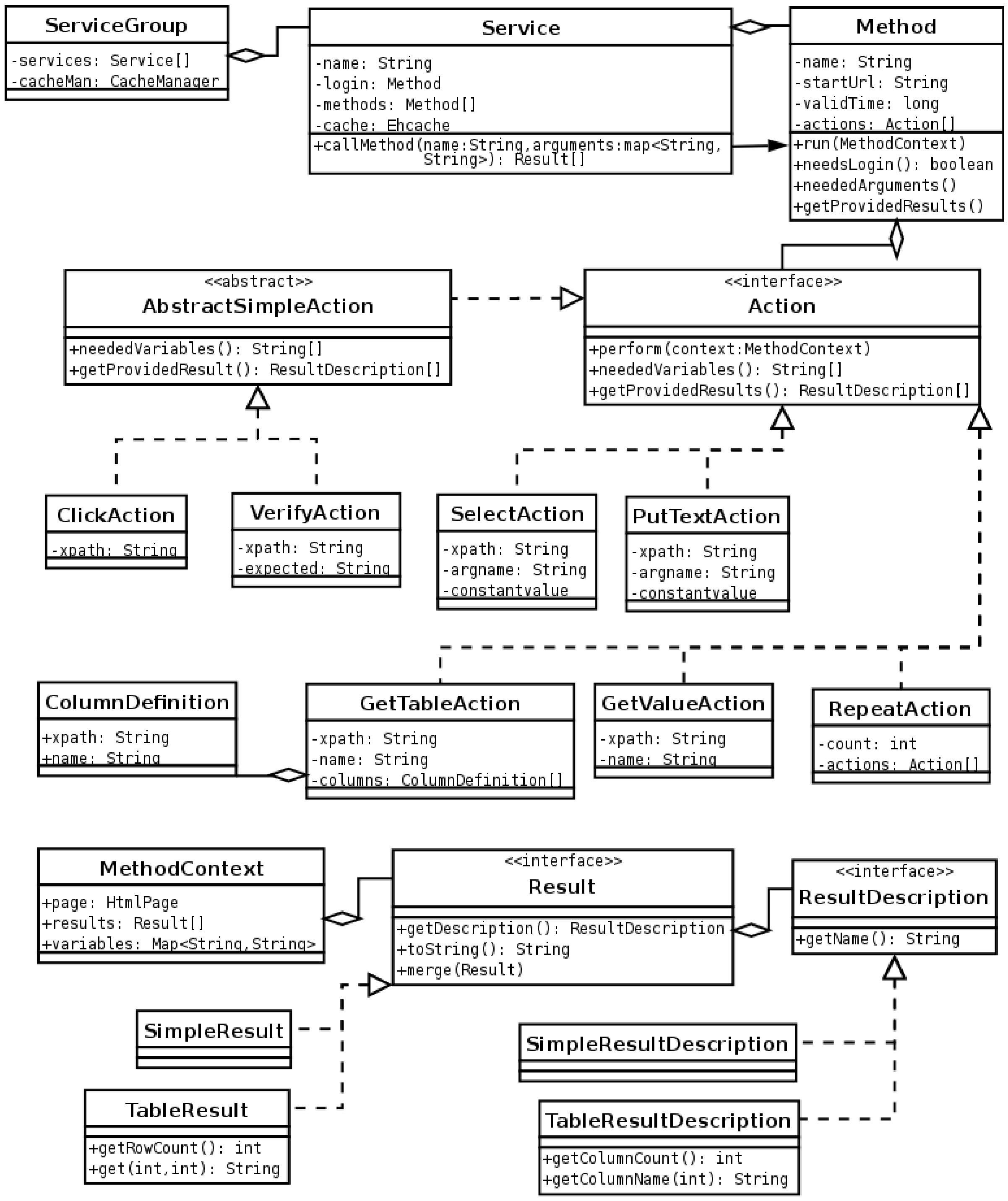

Figure 2: Class diagram of the service part (stripped-down)

Two types of results are currently supported: A simple result and a table result. The simple result is just a string, the table result a matrix of strings. They are defined by the actions `getvalue` and `gettable`. The attributes that are defined in the service configuration files are stored in a result description. Both of them have a name for identification which is used as tag name in the SOAP response. In addition for a table result the number of columns and their names are stored in a table result description.

4.3 HTMLUnit

The library HTMLUnit [6] is used to access the legacy web sites. In simple terms, it is a "browser for Java programs" and "models HTML documents and provides an API that allows you to invoke pages, fill out forms, click links, etc." Its basic class is the `WebClient`, which corresponds to a web browser program. It contains pages that are accessed by the actions using XPath. The result of an XPath query are one or more DOM nodes. They can be used to enter some text, to click onto them or to retrieve the text they contain. The retrieved text is normally the text content of a node. Only if the node is an attribute then its value is used or if the node is an HTML input element then its value is used.

HTMLUnit's support for JavaScript is disabled by default because most web sites can be used without and the use of JavaScript consumes needless computing power. Nevertheless it can be helpful, for instance in AJAX web applications. But then it should be tested because it does not work well with all scripts. Details are available at the homepage of HTMLUnit [6]

4.4 Web Service

To expose the services as web services Apache Axis2 [1] is used. On startup for each service an Axis service is built and all of them are combined to an Axis service group. For each method within a service an Axis operation is added to the Axis service. The argument and result types are defined as XML schema elements, which is a little bit complex for result tables (produced by `gettable` actions).

Finally the Axis service group is added to the Axis configuration and the whole thing is put into a simple HTTP server. It can also be used as a servlet and put into any other servlet container.

When an HTTP request arrives it is dispatched by the web server and passed through Axis to the `MessageReceiver` class. This class works as a gateway between the XML world of the SOAP protocol and the Java world of functions of the services and methods. It parses the method parameters and calls the method. Afterwards it creates an XML element out of the method results and puts it into the result message of the SOAP request.

4.5 Caching

Caching is used for two purposes: To provide session management and to provide caching of method results. Mostly a service contains a login method that is used to define the starting point of all its methods. If its result can be reused then that means that we can cache the last page of the login method and directly use it the next time. Caching of method results can be enabled by the user by specifying a "validTime" for a method. This means that for a given time a cached result can be returned instead of executing the actions of a method. Important is that the method parameters have to be the same.

We decided to use ehCache [4] for caching because it is a mature library and provides disk caching too. It supports to create several independent named caches that means several named pools of cached items. For each service a separate cache is used. It is assigned by the service group which manages the caches. Cached items are identified by a string. The result of a login method is always identified by "+login". Method results are identified by the name of the method and a concatenation of the values of the method arguments.

In the beginning we thought it would be possible to cache the web page objects of HTMLUnit directly. But they are mutable and it turned out to be impossible to create independent copies. Therefore the cached elements are the web response objects. They contain the HTTP response of the web server. The drawback of caching only the web response is that external files like JavaScript files or CSS files are not cached. This also means that in the case of HTML frames only the page that defines the frames is cached and not the frame pages that contain the relevant information.

4.6 Building the ServiceEnabler

Ant is used as a build system. The required libraries are listed in section 4.7 and can be found in the "lib" directory of the repository. The ant targets "compile" and "run" allow to easily build and run the ServiceEnabler. The target "jar" creates a JAR archive file that can be executed using the target "run-jar". The JAR file does not contain the required libraries and is therefore very small (ca. 80KB). This brings the advantage that in case of an update only this small archive has to be exchanged. That also means that for deployment the JAR file and and the "lib" directory have to be distributed.

4.7 Used Libraries

Library	Field of application
Axis2 1.4.1[1]	Providing the web services and the WSDLs
ehCache 1.6.0[4]	Caching of login pages and of results
HTMLUnit 2.3[6]	Accessing and manipulating web sites
JUnit 4[7]	Unit testing
Log4J 1.2.15[2]	Logging
XStream 1.3[8]	Reading in XML configuration files

5 Conclusions and Outlook

The ServiceEnabler system shows how it is possible to treat web sites as web services. It provides an infrastructure that only needs simple XML configuration files that describe the used web sites at a high level. The descriptions correspond to the actions a real user would carry out. The ServiceEnabler system performs these actions and allows to access web sites as interoperable web services.

A tool to help the user with the creation of the configurations would be an important improvement. At the moment the XML configuration files have to be written by hand. This tool should be a browser plugin that works like a macro recorder. The user should have to define a starting page and then for every user action a ServiceEnabler action is defined. If the user enters text or selects an item then he should be asked whether this is a constant value or a variable one. In the case of a variable value the user has to provide a descriptive name. To define results the user should be able to select parts of a web page like it is possible with Firebug [5] or Dapper [3].

Another improvement would be to provide a servlet interface. At the moment a simple HTTP server of the Axis project is used. For bigger installations the use of a high performance servlet container would provide a better throughput. This enhancement should be easily to add because Axis is prepared for this.

To improve stability and recognise errors type checking could be added. In the configuration for each input and output value a type could be defined and verified at runtime. If a wrong type was detected a fault should be thrown. This would help to detect invalid input of the client applications and unexpected result values of the web sites.

And finally - this is a more ambitious feature request - a template engine including scripting support would provide a lot of new opportunities. Instead of only distinguishing between constant values and variables each value could be the result of a calculation. Also, each result value could be transformed or merged with other values. The JavaScript engine that is already used by HTMLUnit could be reused to achieve this.

References

[1] Apache axis 2. http://ws.apache.org/axis2.

[2] Apache log4j. http://ehcache.sourceforge.net.

[3] Dapper. http://www.dapper.net/.

[4] ehcache. http://ehcache.sourceforge.net.

[5] Firebug. http://getfirebug.com/.

[6] Htmlunit. http://logging.apache.org/log4j/.

[7] Junit. http://www.junit.org/.

[8] Xstream. http://xstream.codehaus.org.

A XML Schema Definition for services

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- XML schema definition for a Service within
        the ServiceEnabler infrastructure.
       Author: Andreas Hubmer -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.tuwien.ac.at/serviceenabler/service"
  xmlns="http://www.tuwien.ac.at/serviceenabler/service"
  elementFormDefault="qualified">

<xs:element name="service">
<xs:complexType>
  <xs:sequence>
        <xs:element minOccurs="0" name="login" type="loginmethod"/>
        <xs:element maxOccurs="unbounded" name="method" type="method"/>
  </xs:sequence>
  <xs:attribute name="name" use="required" type="name"/>
</xs:complexType>
<xs:key name="methodname">
        <xs:selector xpath="method"/>
        <xs:field xpath="@name"/>
</xs:key>
</xs:element>

<xs:complexType name="method">
  <xs:sequence>
        <xs:element type="actions" name="actions"/>
  </xs:sequence>
  <xs:attribute name="name" use="required" type="name"/>
  <xs:attribute name="startUrl" use="optional" type="xs:anyURI"/>
  <xs:attribute name="validTime" type="xs:nonNegativeInteger" default="0"/>
</xs:complexType>

<xs:complexType name="loginmethod">
  <xs:sequence>
        <xs:element type="actions" name="actions"/>
  </xs:sequence>
  <xs:attribute name="startUrl" use="required" type="xs:anyURI"/>
  <xs:attribute name="validTime" type="xs:nonNegativeInteger" default="0"/>
</xs:complexType>

<xs:complexType name="actions">
  <xs:sequence>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="click" type="click"/>
          <xs:element name="puttext" type="puttext"/>
          <xs:element name="verify" type="verify"/>
          <xs:element name="gettable" type="gettable"/>
          <xs:element name="select" type="select"/>
          <xs:element name="getvalue" type="getvalue"/>
          <xs:element name="repeat" type="repeat"/>
        </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="gettable">
  <xs:sequence>
        <xs:element maxOccurs="unbounded" name="column" type="column"/>
  </xs:sequence>
  <xs:attribute name="name" use="required" type="name"/>
</xs:complexType>

<xs:complexType name="column">
  <xs:sequence>
        <xs:element name="name" type="name"/>
        <xs:element name="xpath" type="xpath"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="select">
  <xs:sequence>
        <xs:element name="xpath" type="xpath"/>
        <xs:element name="argname" type="name"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="getvalue">
  <xs:sequence>
        <xs:element name="xpath" type="xpath"/>
  </xs:sequence>
  <xs:attribute name="name" use="required" type="name"/>
</xs:complexType>

<xs:complexType name="puttext">
  <xs:sequence>
        <xs:element name="xpath" type="xpath"/>
        <xs:element name="argname" type="name"/>
        <xs:element minOccurs="0" name="constantvalue" type="constantvalue"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="click">
  <xs:sequence>
    <xs:element name="xpath" type="xpath" />
  </xs:sequence>
</xs:complexType>

<xs:complexType name="verify">
  <xs:sequence>
        <xs:element name="xpath" type="xpath"/>
        <xs:element name="expected" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="repeat">
    <xs:complexContent>
      <xs:extension base="actions">
        <xs:attribute name="count" use="required" type="xs:positiveInteger"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>

  <xs:simpleType name="name">
    <xs:restriction base="xs:NCName"/>
  </xs:simpleType>
  <xs:simpleType name="xpath">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>
  <xs:simpleType name="constantvalue">
    <xs:restriction base="xs:string"/>
  </xs:simpleType>

</xs:schema>
```