WordReplacer Development Manual

Kamaledin Ghiasi-Shirazi, Mahdi Mohseni, Majid Darvishan, and Reza Yousefzadeh

***Abstract*—In this article we study two simplified examples of the usage of the RSCM technology. In the first example, we introduce a manageable server-side networking library for SMPP protocol with the ability to manage the connection objects. In the next example, we use this SMPP networking library to develop a WordReplacer application which receives messages from a SMPP client and replaces the words in the messages according to some replacement rules. Finally, we show how the WordReplacer application can be managed with our CLI and web-based managers.**

I. A Networking Library with Manageable Connections for SMPP Protocol

In this example, we explain the process of producing a server-side SMPP networking library called SmppNetworking. Here, one important design goal is to have manageable networking connections. As mentioned earlier, the RSCM technology can only manage objects which are created at initial load of the application from the configuration file or by a CREATE reconfiguration command. However, a connection object is usually created when a client connects and is deleted after the network connection is closed. Thus, in the usual design the connection objects are created by an event and so cannot be managed by the RSCM technology. Solving this problem in the client side is easy: it is enough to permanently associate a connection object with each client-side network connection which continues to exist even when the network connection is closed. However, management of server-side connections is more subtle. For those network protocols where the clients cannot be identified after losing the network connection, like HTTP, the lifetime of a server-side connection object is restricted to the lifetime of the associated network connection. However, In the SMPP protocol clients send a bind packet immediately after the network connection is established. The bind packet contains a SystemId field which serves as the identity of the client. In our modified design, the SMPP server creates a connection object for each valid client that is listed in the configuration file (or created using CREATE command) and uses these objects when the associated clients actually connect. This way, the lifetime of the connection objects would be independent of whether the network connection is established or not.

The detailed design of the SmppNetworking library is as follows. For each valid client, we consider an instance of the SmppExternalClient class which contains the client's information like SystemId and Password. The SmppExternalClient class represents an ESME (External Short Messaging Entity) in the SMPP protocol. Each object of type SmppExternalClient contains a child of type SmppConnection whose lifetime is the same as its parent. The creation/deletion of SmppConnection objects, like any other managed object, is

performed by the reconfiguration commands and the establishment of the actual network connection has no effect on it. After a client connects, we create a temporary non-manageable object of type SmppConnection. The configuration parameters of this temporary connection are cloned from a manageable SmppConnection called the DefaultConnectionTemplate. According to the SMPP protocol, when a client connects to a server it should send a Bind packet which contains a SystemId field that identifies the associated SmppExternalClient object. After receiving the Bind packet, we copy the socket information of the temporary SmppConnection to the actual manageable SmppConnection that is associated with the client and delete the temporary SmppConnection.

Main managed classes of the SmppNetworking library are[1]:

1- SmppServer: The SmppServer is the root object which encompasses all other objects associate with a SMPP server. SystemId of the server and the SessionInitTimeout (which according to the SMPP protocol is the longest time where a client is allowed to delay between establishment of the connection and sending the Bind packet) are the configuration parameters of this object. In addition, to allow the users of the library to decide whether to restrict the permissible clients to those specified in the ExternalClientdsTable or not, the BindProcessPolicy configuration parameter is introduced. If the BindProcessPolicy is set equal to ContinueInitialConnection the packets are processed by the unmanageable SmppConnection object which was cloned from the DefaultConnectionTemplate. However, if the BindProcessPolicy is set to SwitchToKnownConnections then after receiving the bind packet, the temporary SmppConnection is deleted and the network connection is assigned to the appropriate manageable SmppConnection within the ExternalClientdsTable table. Figure 1 shows the XML Schema of the SmppServer.

2- SmppConnection: The SmppConnection is a managed class which represents a network connection. According to the SMPP protocol, each connection has the following configuration variables: EnquireLinkTimeout, InactivityTimeout, and ResponseTimeout. We also designate the actions which can be performed on a managed object by an element with minOccurs="0" and maxOccurs="0" in the XML Schema (a technique previously used by Menten[1]) . For the SmppConnection, we have provisioned the Disconnect action. Figure 2 shows the XML Schema of the SmppConnection managed class.

3- SmppExternalClient: The SmppExternalClient represents a valid client (or ESME in SMPP terminology). The SMPP protocol has provisioned the SystemId, SystemType, Password, and PermittedBindTypes configuration parameters for each valid client. It must be mentioned that by overriding the IsPassword method of the SimpleManagedObject class in the SmppExternalClient class we have reqested the subagent to store the Password configuration

[1] In accompanying codes and documents the names of the classes may be preceded by a "PA_" prefix.

parameter in encrypted format. In addition, each SmppExternalClient includes a child of type SmppConnection which allows the associated connection to be manageable even when the client is disconnected. This manageable SmppConnection object can be used to reconfigure the configuration parameters of a connection and monitor its state, even when the actual networking connection is closed. The proposed trick prevents the monitoring variables associated with a client to be reset after an unwanted disconnection. The schema of this class is depicted in Figure 3.

4- SmppExternalClientsTable: The SmppExternalClientsTable is a tabular managed object which stores the list of the ExternalClient objects. The Schema of this class is depicted in Figure 4.

II. A Sample WordReplacer Application

In this section, we use the RSCM technology to develop a sample application called WordReplacer. This application contains a SMPP sever, which is an instance of the SmppServer class of the previous section, along with a list of words and their replacements. After binding, clients can send messages using SubmitSm packet of the SMPP protocol. When a message contains a word that is in the words list, the word is replaced by its associated replacement word and a new message is composed and returned back to the sender. To show how the RSCM technology allows a modular design, we use the SmppNetworking library of the previous section. From the above definition of the WordReplacer application, we can identify the following classes:

1- WordReplacer class which is the main class of the application and represents the whole WordReplacer application. This class has two children: one of type WR_Server (which inherits from the SmppServer) and another child of type WordReplacementRulesList where its definition follows. In addition, it is possible to run the ReplaceWord action on the WordReplacer. The XML schema of the WordReplacer is shown in Figure 5.

2- WR_ExternalClient which inherits from the SmppExternalClient and has an additional configuration parameter called WordReplacementLicensePerMinute. Note how a general SMPP configuration parameter, like SystemId, is specified in the SmppNetworking library while this application-specific configuration parameter is specified in the application-level class WR_ExternalClient. The XML schema of the WR_ExternalClient class is shown in Figure 6

3- WordReplacementRulesList which contains the list of the replacement rules. Figure 7 shows the XML schema of this class.

4- WordReplacementRule which represents a rule. It contains two configuration parameters: OriginalWord and NewWord. Each rule has a monitoring variable which counts the number of times that this rule has been applied. The XML Schema of this class is depicted in Figure 8.

In addition to the above classes, there are some other classes which inherit from the classes of the SmppNetworking

library but have no additional configuration parameters. These are application-specific classes which implement or override some of the functionalities of their parent classes. These classes are:

1- WR_Server which inherits from the SmppServer class.
2- WR_Connection which inherits from the SmppConnection class.
3- WR_ExternalClientsTable which inherits from the SmppExternalClientsTable class.

All the XML Schemas mentioned up to now are associated with some class. However, there should be an XML Schema which designates the format of the XML configuration file as a whole. This XML Schema is depicted in Figure 9. The C++ codes of these classes along with the associated XSD files are available at http://profsite.um.ac.ir/~ghiasi/publications/RSCM/index.html However, since the code depends on some proprietary libraries of PeykAsa Company (including the SmppNetworking and RSCM) it cannot be compiled.

III. Managing the WordReplacer Application by a CLI

We have developed a CLI with a syntax which is compatible with the ITU-T Z300 standard. This CLI is completely general and can be used to manage any application which is developed by the RSCM technology. In this section we report our experiment in managing the WordReplacer application with this CLI. Figure 10 shows a sample run of the CLI for creating a new managed object. Figure 11 shows another sample run for deleting managed objects from the WordReplacer application. To view a complete list of the images and videos of how this CLI is used for managing the WordReplacer application visit http://profsite.um.ac.ir/~ghiasi/publications/RSCM/index.html

IV. Managing WordReplacer Application by a Web-Based Manager

PeykAsa Company has developed a web-based management system based on the 3GPP 32.603 standard which uses the CORBA technology for the manager-agent communication. We added the RSCM management protocol to the agent of this web-based management system and used the modified system for web-based management of the telecommunication applications developed by the RSCM technology. The user interface of the web-based manager is automatically generated from a database which in turn is filled in automatically using the XSD files of the application.

Since the RSCM technology organizes the managed objects in a tree structure, it seems natural to show a tree view of the managed objects in the user interface. However, the tree representation of a tabular managed object with many children is not readable for human users. In addition, implementing a web interface to add a new tree-structured managed object is a complicated task. Since tabular objects can have only one type of child, a natural representation is to display the subtree of a tabular managed object as a table with each child in a row and each configuration variable in a column. We map the tree structure of the child class to a linear list by adding the name of the parent objects to the beginning of the name of each configuration variable, using dot to join the names. Figure 12 shows the tree representation of a tabular managed object as

displayed by our CLI. The tabular representation of the same object in the web-based UI is shown in Figure 13. Note how the configuration variables of the SmppExternalClient class are linearized in the web-based representation.

REFERENCES

[1] L. E. Menten, "Experiences in the application of XML for device management," *IEEE Commun. Mag.*, vol. 42, pp. 92-100, 2004.

```
  <xs:complexType name="SmppServer">
    <xs:sequence>
      <xs:element name="Port" minOccurs="1">
      <xs:simpleType>
      <xs:restriction base="xs:unsignedShort">
        <xs:minInclusive value="1024" />
      </xs:restriction>
      </xs:simpleType>
      </xs:element>
      <xs:element name="SystemId" type="xs:string" minOccurs="1" />
      <xs:element name="SessionInitTimeout" type="xs:unsignedInt" minOccurs="1" />
      <xs:element name="BindProcessPolicy" minOccurs="1">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="ContinueInitialConnection" />
            <xs:enumeration value="SwitchToKnownConnections" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="DefaultConnectionTemplate" type="SmppConnection" minOccurs="1"
/>
      <xs:element name="ExternalClientsTable" type="SmppExternalClientsTable"
minOccurs="1" >
      <xs:key name="ExternalClientSystemIdKey">
          <xs:selector xpath=".//ExternalClient" />
          <xs:field xpath="SystemId" />
      </xs:key>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

Figure 1: XML Schema for SmppServer

```
    <xs:complexType name="SmppConnection">
        <xs:sequence>
            <xs:element name="EnquireLinkTimeout" type="xs:integer"/>
            <xs:element name="InactivityTimeout" type="xs:integer"/>
            <xs:element name="ResponseTimeout" type="xs:integer"/>
            <xs:element minOccurs="0" maxOccurs="0" name="ActionList">
                <xs:complexType>
                    <xs:all>
                        <xs:element name="Disconnect">
                            <xs:complexType>
                                <xs:all>
                                </xs:all>
                            </xs:complexType>
                        </xs:element>
                    </xs:all>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
```

Figure 2: XML Schema for SmppConnection

```
    <xs:complexType name="SmppExternalClient">
        <xs:sequence>
          <xs:element name="SystemId" type="xs:string" />
          <xs:element name="SystemType" type="xs:string" />
          <xs:element name="Password" type="xs:string" />
    <xs:element name="PermittedBindTypes" default="TRX">
      <xs:simpleType>
      <xs:restriction base="xs:string">
            <xs:enumeration value="TX" />
      <xs:enumeration value="RX" />
      <xs:enumeration value="TRX" />
            </xs:restriction>
    </xs:simpleType>
    </xs:element>
    <xs:element name="Connection" type="SmppConnection" />
        </xs:sequence>
  </xs:complexType>
```

Figure 3: The XML Schema of the SmppExternalClient class

```
  <xs:complexType name="SmppExternalClientsTable">
    <xs:sequence>
      <xs:element name="ExternalClient" type="SmppExternalClient" minOccurs="0"
maxOccurs="unbounded">
      </xs:element>
    </xs:sequence>
    <xs:attribute name="key" type="xs:string" use="required" />
  </xs:complexType>
```

Figure 4: The XML Schema of the SmppExternalClientsTable class

```
  <xs:complexType name="WordReplacer">
    <xs:sequence>
      <xs:element name="WordReplacerServer" type="SmppServer" />
      <xs:element name="WordReplacementRulesList" type="WordReplacementRulesList">
        <xs:key name="OriginalWord">
          <xs:selector xpath=".//WordReplacementRule" />
          <xs:field xpath="OriginalWord" />
        </xs:key>
      </xs:element>
      <xs:element minOccurs="0" maxOccurs="0" name="ActionList">
        <xs:complexType>
          <xs:all>
            <xs:element name="ReplaceWord">
              <xs:complexType>
                <xs:all>
                  <xs:element name="Word" type="xs:string" />
                </xs:all>
              </xs:complexType>
            </xs:element>
          </xs:all>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
```

Figure 5: The XML Schema for WordReplacer class.

```
<xs:complexType name="WR_ExternalClient">
  <xs:complexContent>
    <xs:extension base="SmppExternalClient">
      <xs:sequence>
        <xs:element minOccurs="1" name="WordReplacementLicensePerMinute"
type="xs:unsignedInt" />
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figure 6: The XML Schema for WR_ExternalClient

```
    <xs:complexType name="WordReplacementRulesList">
        <xs:sequence>
    <xs:element name="WordReplacementRule" type="WordReplacementRule" minOccurs="0"
maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="key" type="xs:string" use="required" />
    </xs:complexType>
```

Figure 7: The XML Schema for WordReplacementRulesList

```
    <xs:complexType name="WordReplacementRule">
        <xs:sequence>
            <xs:element name="OriginalWord" type="xs:string" />
            <xs:element name="NewWord" type="xs:string" />
        </xs:sequence>
    </xs:complexType>
```

Figure 8: The XML Schema for WordReplacementRule class.

```xml
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <xs:element name="PA_Component">
    <xs:annotation>
      <xs:appinfo>
        <managedElementType value="WordReplacer"/>
      </xs:appinfo>
    </xs:annotation>
    <xs:complexType>
      <xs:all>
        <xs:element name="WordReplacer" type = "WordReplacer"/>
      </xs:all>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 9: The XML Schema of the whole WordReplacer program.

```
WR>$help: sig;
       "sig" command prints the signature for adding a new managed object to a parent tabular node.
       Syntax:"sig: <RDN>;" where <RDN> is the relative distinguished name of the parent node (which is a tabular node).
WR>$help: add;
       "add"  command is used to add a child node to another node. The parent node should be a tabular Managed Object.
       Syntax:"add: <RDN>: <MOC>: <GATR>=<VAL>(,<GATR>=<VAL>)*;" where:
       <RDN>  is the relative distinguished name of the parent node
       <MOC>  is the managed object class of the new node
       <GATR> is a generalized attribute of the new node
       <VAL>  is a numeric or string value
       Definition: A generalized attribute of a node is either an attribute of that node or an attribute of a desendent node
              In the latter case, the name of the generalized attribute consists of the sequence of intermediate nodes
              and the attribute name separated by comma.
       Example: Assume that object T1 is tabular and can contain objects of class C1.
              Objects of class C1 contain and attribute c1 and two objects named A and B.
              In addition, A contains two attributes a1 and a2 and B contains attributes b1 and b2. Then the command
                     "add: T1: C1: c1=7, A.a1=a1, A.a2=2, B.b1=8, B.b2=9;"
              adds a new node of type C, with the specified values for attributes of itself and its children, to T1.
WR>$cn: WordReplacer, WordReplacementRulesList;
WR>WordReplacer,WordReplacementRulesList>$ls;
       WordReplacementRule="acheive"
       WordReplacementRule="accross"
       WordReplacementRule="appearence"
       WordReplacementRule="begining"
       WordReplacementRule="beleive"
WR>WordReplacer,WordReplacementRulesList>$sig;
       MOC (Managed Object Class) of children is: WordReplacementRule.
       The following table lists the attributes of this class.

       Name                                                                                |   Type        |Opt/Req   |Default Value
       -----------------------------------------------------------------------------------+---------------+----------+---------------
       OriginalWord                                                                       |string         |   R      |
       NewWord                                                                            |string         |   R      |
WR>WordReplacer,WordReplacementRulesList>$add: : WordReplacementRule: OriginalWord="concious", NewWord="conscious";
       Add command succeeded.
WR>WordReplacer,WordReplacementRulesList>$ls;
       WordReplacementRule="acheive"
       WordReplacementRule="accross"
       WordReplacementRule="appearence"
       WordReplacementRule="begining"
       WordReplacementRule="beleive"
       WordReplacementRule="concious"
WR>WordReplacer,WordReplacementRulesList>$lsatr: WordReplacementRule="concious";
       OriginalWord : concious
       NewWord : conscious
```

Figure 10: A sample run of the text-based manager used for adding managed objects to the WordReplacer application.

```
WR>$help: del;
        "del" command is used to delete a child node from a tabular node.
        Syntax: "del: <RDN>;" where <RDN> is the relative distinguished name of the node which is to be deleted. Note that <RDN> cannot be empty
        Example:
                del:M1,M2,M3="id";
WR>$cn: WordReplacer , WordReplacementRulesList;
WR>WordReplacer,WordReplacementRulesList>$ls;
        WordReplacementRule="acheive"
        WordReplacementRule="accross"
        WordReplacementRule="appearence"
        WordReplacementRule="begining"
        WordReplacementRule="beleive"
        WordReplacementRule="buisness"
WR>WordReplacer,WordReplacementRulesList>$del: WordReplacementRule="accross";
        Delete command succeeded.
WR>WordReplacer,WordReplacementRulesList>$ls;
        WordReplacementRule="acheive"
        WordReplacementRule="appearence"
        WordReplacementRule="begining"
        WordReplacementRule="beleive"
        WordReplacementRule="buisness"
WR>WordReplacer,WordReplacementRulesList>$ls: .., WordReplacerServer, ExternalClientsTable;
        ExternalClient="WRClient1"
        ExternalClient="WRClient2"
WR>WordReplacer,WordReplacementRulesList>$del: .., WordReplacerServer, ExternalClientsTable, ExternalClient="WRClient2";
        Delete command succeeded.
WR>WordReplacer,WordReplacementRulesList>$ls: .., WordReplacerServer, ExternalClientsTable;
        ExternalClient="WRClient1"
WR>WordReplacer,WordReplacementRulesList>$
```

Figure 11: A sample run of the text-based manager used for deleting managed objects from the WordReplacer application.

```
--ExternalClientsTable
  |--ExternalClient="WRClient1"
  |   |  SystemId=WRClient1
  |   |  SystemType=Telecom
  |   |  Password=*****
  |   |  PermittedBindTypes=TRX
  |   |  WordReplacementLicensePerMinute=5
  |   |--Connection
  |         EnquireLinkTimeout=60
  |         InactivityTimeout=600
  |         ResponseTimeout=30
  |--ExternalClient="WRClient2"
      |  SystemId=WRClient2
      |  SystemType=Telecom
      |  Password=*****
      |  PermittedBindTypes=TRX
      |  WordReplacementLicensePerMinute=500
      |--Connection
           EnquireLinkTimeout=60
           InactivityTimeout=600
           ResponseTimeout=30
```

Figure 12: The tree representation of a tabular managed object used by the CLI.

SystemId	SystemType	Password	PermittedBindTypes	WordReplacementLicensePerMinute	Connection.EnquireLinkTimeout	Connection.InactivityTimeout	Connection.ResponseTimeout
WRClient1	Telecom	*****	TRX	5	60	600	30
WRClient2	Telecom	*****	TRX	500	60	600	30

1 of 1 1-2 of 2

Add Delete Update Select Node Close

Figure 13: The tabular representation of the tabular managed object of Figure 12 in the web-based manager.