Managed Gateway Plugin

Adding support for managed gateways

Introduction

sipXconfig is an extensible management system for complete Enterprise communications solutions based on SIP. It is capable of managing both the server side infrastructure as well as all the required devices, such as phones and gateways and additional applications.

If you want to add support for new type of phones or FXS gateways you should check section on Managed Phones Plugin. This page describes the steps required to add support for FXO gateways.

Three easy steps are described below:

1. Registering a new gateway type with sipXconfig
2. Defining the gateway's features using an XML file
3. Implementing profile (configuration file) generation

After those 3 steps are completed, sipXconfig users will be able to select new gateway type, configure the gateway using sipXconfig provided UI and generate gateway configuration files.

Registering a new gateway type

Gateway class

To add support for a newly released gateway from Acme Corp. we start by defining a class the gateway model class. Instances of this class will represent physical gateways known to sipXconfig. A newly added class has to extend the existing *org.sipfoundry.sipxconfig.gateway.Gateway* class.

```
package org.sipfoundry.sipxconfig.gateway.acme
public class AcmeGateway extends Gateway {

  private PhoneDefaults m_defaults;

  protected void defaultSettings() {{panel}
      super.defaultSettings();
      // TODO: add code that sets common settings for all gateways
  }

  public void generateProfiles() {
      // TODO: add code that generates configuration files
  }

  public void removeProfiles() {
      // TODO: add code that cleans generated configuration files
  }

  public void setDefaults(PhoneDefaults defaults) {
      m_defaults = defaults;
  }
}
```

Couple of comments:

- m_defaults will be magically injected in your gateway class - this is the instance of the PhoneDefaults class and despite its name it contains whole bunch of quite itneresting data about the system - you can get access to SIP Proxy address, TFTP server directory etc.
- generateProfiles and removeProfiles are called by sipXconfig when system needs to generate (or respectively clean) configuration for this gateway - the usual implementation of generateProfiles will use some kind of templating system (for example Velocity) to inject Gateway settings into configuration file template, removeProfiles should clean all the files generated by generateProfiles
- defaultSettings is a function that is called always before the instance of the gateway object is used - it's a change to set the default values for some common settings that cannot be hardcoded since they depend on specific installation - usually all settings the require IP address name of the SIP proxy or SIP domain name get their default value here

Registering gateway class

To register gateway class we need also to define gateway model and settings model.

SettingModel (see smAcme) is needed in order to locate and load .xml setting file that describes parameters for the gateways. The path is relative to sipXconfig/neoconf/etc directory in the source tree. Setting files are installed {prefix}/etc/sipxpbx.

Gateway class definition (see gwAcme below) needs to set "settingModel" property to model bean id.

Gateway model (see gmAcme definition below) is used by sipXconfig when the list of all available models is displayed. It has 3 paramters specifying respectively gateway class, gateway internal model number, and user readable label.

```
<bean id="smAcme" class="org.sipfoundry.sipxconfig.setting.SettingModel">
<property name="resource" value="acme/acme-gateway.xml"/>
<property name="modelFilesContext" ref="modelFilesContext"/>
</bean>

<bean id="gwAcme" class="org.sipfoundry.sipxconfig.gateway.acme.AcmeGateway" singleton="false"
parent="gwGeneric">
<property name="settingModel">
<ref local="smAcme"/>
</property>
</bean>

<bean id="gmAcme" class="org.sipfoundry.sipxconfig.phone.PhoneModel">
<constructor-arg><value>gwAcme</value></constructor-arg> <<<<---- needs to refere to gatway class bean id
<constructor-arg><value>1000</value></constructor-arg>
<constructor-arg><value>Acme 1000</value></constructor-arg>
</bean>
```

You also have to register gateway model in the list of available gateway models

```
<bean id="gatewayContextImpl" class="org.sipfoundry.sipxconfig.gateway.GatewayContextImpl">
<...>
<property name="availableGatewayModels">
<list>
<ref local="gmGeneric"/>
<...>
<ref local="gmAcme"/> <<<<----- needs to refer to gateway model
</list>
</property>
</bean>
```

Now it is a good time to check if the code is working:

Run all the tests in the *neoconf module* as follows (Example build and test environment for sipXconfig on Gentoo Linux):

```
ant default style test-all
```

Some of the tests will fail since the setting file is missing.

Defining the gateway model

Of course just letting users select a new gateway is not very exciting. We would also like to configure some settings of this gateway with the sipXconfig UI. The nice thing is that we do not actually have to write any GUI code or learn HTML to accomplish this. All we need to do is to define all the parameters of the new gateway that we want to configure through the sipXconfig UI in the gateway model XML file. The format used for the model file to define gateway properties is exactly the same as the format used to define phone properties when adding support for a new phone.

Let's start with just a couple of paramaters. Real gateways have closer to 300 parameters - I am just showing 3 parameters here in 2 groups "System" and "Voice Engine".

```
<?xml version="1.0"?>
<!DOCTYPE model
PUBLIC "-//SIPFoundry//sipXconfig//Model specification 1.0//EN"
"http://www.sipfoundry.org/sipXconfig/dtd/setting_1_0.dtd">
<model>
<group name="SYSTEM">
<label>System</label>
<setting name="DNSPriServerIP">
<label>Primary DNS Server</label>
<type>
<ipaddr required="yes"/>
</type>
<value/>
</setting>
<setting name="EnableSyslog" advanced="yes">
<label>Syslog</label>
<type>
<boolean/>
</type>
<value>0</value>
</setting>
</group>
<group name="Voice_Engine">
<label>Voice Engine</label>
<profileName>Voice Engine</profileName>
<setting name="IdlePCMPattern">
<type>
<integer/>
</type>
<value>255</value>
</setting>
</group>
</model>
```

It's good to use an XML aware editor with DTD validation to do that, because sipXconfig provides formal DTD - settings.dtd file (on line and in neoconf/etc directory).
There is also a description of the settings format on the Settings File Format page.

The settings format allows for group settings. Each group is rendered by sipXconfig on a separate screen. The settings format also provides for declaring settings as *advanced* - in which case they are only shown if the user presses the "Show Advanced Settings" link. I can also define setting types, which helps sipXconfig choose how to present my setting and how to validate user input. I also can define default values, labels and short descriptions that are shown on the screen.

The nice thing about the settings format is that I can start with a very simple file and gradually add more settings, define or adjust settings type and add quick help. I can do this even without recompiling sipXConfig because all model files are kept in my system's */etc/sipxpbx* directory.

Thanks to the regular structure of the settings file in many cases it is possible to generate a substatial part of the model file from some kind of formal documentation.

I run the test and, if everything is OK, I run sipXconfig. Now when I click on the newly added gateway, I should be able to modify all the settings I defined in my model file.

Defining the settings allows sipXconfig not only to display them in the UI, but also persist them in the database, and back them up with all other managed configurations.

Implementing gateway profile generation

Profile generation is the most gateway specific part of the process. Whenever the user presses the *Send Profiles* button on the *Gateways* page, sipXconfig calls the *generateProfiles* function in the AcmeGateway class. Depending on the configuration protocol supported by the gateway, implementing *generateProfiles* can be really simple or may require substantial time.

sipXconfig natively supports devices that allow for downloading configuration through TFTP. You can generate configuration files using Velocity, DOM4J or just by outputting text to the file.

Gateway.getTftpRoot gives you the name of the directory to which files should be generated to be visible through TFTP. *Gateway.getSettings* returns an easily traversable settings collection. The easiest way to retrieve all settings names and values is writing your own SettingsVisitor.

Please post questions to the sipx-dev mailing list.