SipXecs replication mechanism

At a glance

First, it is important to understand what "replication" means and why it exists. Basically by replication we mean writing some data managed by sipXconfig (kept in the database) to a medium and at a location accessible by some entity. For instance grabbing a list of users with some of their properties from the database and writing them to an XML file to be used by the Imbot service. Or configuring some parameters of a service and writing them to an XML or properties service configuration file at a location accessible by the service. If necessary, a restart of the service can be triggered so that the new configuration to take effect.

While not complete, this article should give you a head start in understanding sipXconfig's replication mechanism. Hopefully it will grow bigger wiki style. Meanwhile, you can always discover things by looking at the code. Good luck!

Replication of service configurations

All configuration for each sipXecs service is created by sipXconfig and *replicated* to the system where the service runs through sipXsupervisor.

In sipXconfig, the managed services configurations are kept in the database for presentation purposes. Each service keeps its configuration in XML files found at specific locations. A client may alter a service configuration via sipXconfig's UI or REST services. Obviously the configuration of each service must be kept in synch with sipXconfig's database. Thus a configuration change via sipXconfig will trigger a replication of the service's configuration. In particular cases, a service restart must be triggered in order for the new configuration to take effect.

Triggering replications

Replications are triggered in two ways: in bulk, via one of the org.sipfoundry.sipxconfig.service.ServiceConfigurator.replicateServiceConfig() overloaded methods or manually with the help of org.sipfoundry.sipxconfig.admin.commserver.SipxReplicationContext. In the first case all the configuration files of the service are replicated, in the second specific file replications are triggered. Ideally all files should be declared as configuration files that belong to an entity (most probably a service) and get replicated by the `ServiceConfigurator`.

The places where replications are triggered are scattered around sipXconfig's code. In the article SipXconfig replication triggers you will find more details about each service replication rules.

Datasets replications

In addition to service configurations there is another type of replications: replication of datasets. (More details below.)

Other replications

Not only sipXecs services configurations and data sets need to be replicated. In addition to those, other entities use configuration files that sipXconfig must provide via the replication mechanism. Some examples are: dial plans, alarms configuration, etc.

Configuring replications

Service configs

In sipXconfig, services are registered as Spring beans in service.beans.xml. Here, amongst other properties, a list of "service configurations" is kept for each service.

```
  <bean id="sipxProxyService" class="org.sipfoundry.sipxconfig.service.SipxProxyService" parent="sipxService"
    scope="prototype">
    <property name="processName" value="SIPXProxy" />
    <property name="modelName" value="sipxproxy.xml" />
    <property name="modelDir" value="sipxproxy" />
    <property name="editable" value="true" />
    <property name="configurations">
      <list>
        <ref bean="sipxProxyConfiguration" />
        <ref bean="peerIdentitiesConfiguration" />
      </list>
    </property>
    <property name="bundles">
      <set>
        <ref bean="primarySipRouterBundle" />
        <ref bean="redundantSipRouterBundle" />
      </set>
    </property>
  </bean>
```

Each "configuration" is registered separately:

```
  <bean id="sipxProxyConfiguration" class="org.sipfoundry.sipxconfig.service.SipxProxyConfiguration" scope="
prototype"
    parent="serviceConfigurationFile">
    <property name="template" value="sipxproxy/sipXproxy-config.vm" />
    <property name="name" value="sipXproxy-config" />
    <property name="sipxServiceManager" ref="sipxServiceManager" />
  </bean>

  <bean id="peerIdentitiesConfiguration" class="org.sipfoundry.sipxconfig.service.PeerIdentitiesConfiguration"
scope="prototype"
    parent="defaultConfigurationFile">
    <property name="name" value="peeridentities.xml"/>
    <property name="tlsPeerManager" ref="tlsPeerManager" />
    <property name="restartRequired" value="true" />
  </bean>
```

Datasets

None of the datasets is configured as a service configuration. That means that their replication will be triggered manually in places where it is needed.

org.sipfoundry.sipxconfig.admin.commserver.imdb.DataSet holds an enumeration of datasets. Dataset beans are registered in Spring in commserver.beans.xml. Here, the beans hold the configuration of the business beans that do the replication. These beans are implementations of org.sipfoundry.sipxconfig.admin.commserver.imdb.DataSetGenerator abstract class.

The other replications are configured similarly with the service configuration.

Lazy vs eager replication

Some operations may involve modifying a large chunk of entities, each of this modification triggering a replication. For instance, think about importing users from an LDAP server. Each save of a new user should be replicated in various configuration files. So, instead of doing a replication for each operation, the concept of lazy replication was introduced.

org.sipfoundry.sipxconfig.admin.commserver.LazySipxReplicationContextImpl is the implementation of the SipxReplicationContext that takes care of the lazy replications. Replication jobs are kept in an ArrayList of `ReplicationTasks`. A worker thread sleeps while waiting for something to be done and when there is, it is notified, sleeps for a bit and then it triggers the queued replications.

In sipXconfig the vast majority of replications are lazy, with a few exceptions (dial plans, certificates, the deploy of conference bridges on some application events like the deletion of conferences).

Writing XML files

All configuration files extend the abstract class org.sipfoundry.sipxconfig.admin.AbstractConfigurationFile. org.sipfoundry.sipxconfig.admin.TemplateConfigurationFile is a common base class for all configuration files generators that use Velocity templating engine.

The service configuration files are a particular case of the `TemplateConfigurationFile` mentioned above. The configuration bean is registered in Spring as a child of `serviceConfigurationFile` bean. The class itself extends `org.sipfoundry.sipxconfig.admin.TemplateConfigurationFile`. Spring injects the `sipxServiceManager` into the bean. In the above example `sipxProxyConfiguration` is a `serviceConfigurationFile`. It uses sipXproxy-config.vm as the Velocity template. Of course other parameters can be configured on a case by case basis.

Another common way to write XML files is to use org.dom4j.Document. In the above example `peerIdentitiesConfiguration` is configured as a child of `defaultConfigurationFile`. The class itself extends org.sipfoundry.sipxconfig.admin.dialplan.config.XmlFile and it must override `XmlFile.getDocument()` method and return a dom4j document.

Other types of configuration files can be viewed by checking out the type hierarchy of `org.sipfoundry.sipxconfig.admin.AbstractConfigurationFile`.

isReplicable(Location location) and isRestartRequired()

The two are methods of the org.sipfoundry.sipxconfig.admin.ConfigurationFile interface.
From the javadoc, the method `isReplicable(Location location)` verifies if this configuration file can be replicated on the given location. The `isRestartRequired()` method checks if a service need to be restarted when this configuration file is changed.

`isReplicable(Location location)` is implemented in the abstract class `AbstractConfigurationFile`. It will always return true. Implementations of the `AbstractConfigurationFile` must override this method in order to provide aditional checks to make sure the file is replicable at requested location. For instance one might want to see if the service is installed at the specific location in order to replicate its configuration there.

`isRestartRequired()` is initialized with *true* in `AbstractConfigurationFile`. Its value can be overridden by implementing the method in the configuration class or by setting the configuration bean property `restartRequired`.

```
<bean id="presenceRoutingConfiguration" class="org.sipfoundry.sipxconfig.service.PresenceRoutingConfiguration"
  parent="defaultConfigurationFile">
  <property name="name" value="presencerouting-prefs.xml"/>
  <property name="coreContext" ref="coreContext" />
  <property name="restartRequired" value="false" />
</bean>
```

Proposed changes for future versions

See the thread here: http://thread.gmane.org/gmane.comp.voip.sipx.devel/21130