LDAPPC Next Steps

LDAPPC Next Steps

Potential future enhancements to Grouper's provisioner, ldappc, are described here in relation to provisioning work done at Memphis. Feel free to comment.

Wish List

- provision Active Directory, perhaps Exchange
- real-time provisioning
- pluggable attributes (regex, script, etc)
- non-ldap SPML targets
- --dry-run, i.e. show what would be done before doing
- monitoring of long-running daemon-ish process
- better logging to aid troubleshooting
- auditing

and also

- integration with grouper loader
- integration with grouper notifications
- more advanced provisioning selection criteria including events ?
- provision privileges downstream ?

SPML

The Service Provisioning Markup Language (SPML) 2.0 specification encompasses most if not all of the issues that an organization may confront when provisioning groups and memberships. While not specific to Grouper, SPML provides a standard language useful for discussion.

In SPML terms, ldappc resembles both a Requesting Authority (RA) and a Provisioning Service Provider (PSP) : a software component that requests, listens for, processes, and returns the results of provisioning requests. A provisioned LDAP directory service would be a Provisioning Server Target (PST). A provisioned group or membership is a Provisioning Service Object (PSO), unique to a target and identified with a PSO Identifier (PSO-ID).

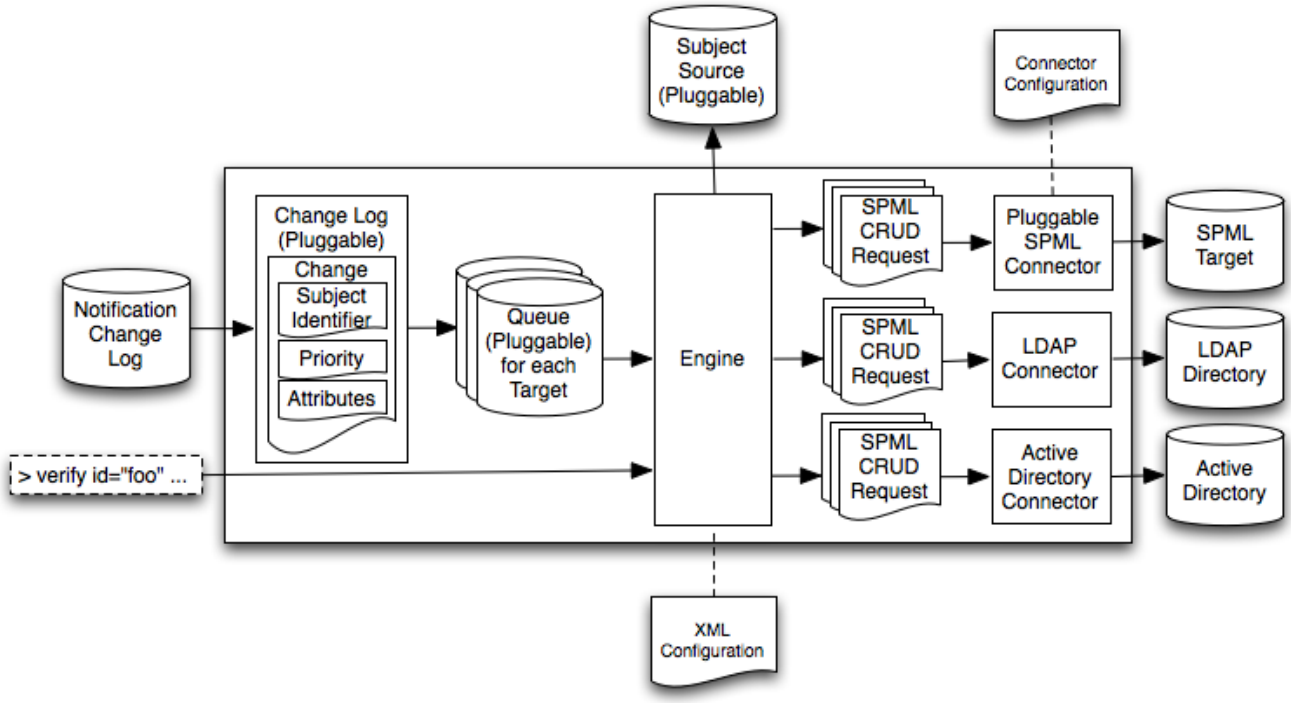
Of interest to Grouper, SPML 2.0 describes References between provisioned objects (groups and their members and vice versa). The SPML specification makes note of concerns regarding scalability (e.g. groups with many members) and referential integrity (e.g. member and memberOf consistency).

Memphis-Style-Provisioning Overview

The provisioning Engine calculates how Subjects, found in Sources, should be provisioned based on an XML configuration. The Engine can be run as a daemon or from the command line.

As a daemon, a thread periodically queries a (pluggable) change log and stores each change, which consists of an identifier, priority, attributes, etc., in (pluggable) queues, one for each target.

A worker thread, one for each target, processes its queue as a FIFO. The worker thread asks the Engine to calculate the resultant provisioning for each change, and then applies the provisioning, as an SPML CRUD request, to its Target via a pluggable Connector. By associating each Target with its own queue and worker thread, changes are processed in order, even across restarts of the Engine when a Target is temporarily unavailable (for maintenance, network downtime, etc).



Provisioning Parameters : Subject Identifier [Attributes] [Priority] [Targets]

To provision an object, the engine requires only a subject identifier, but will also accept attributes, priority, and targets.

The priority can either be passed from the registry changelog or calculated by the engine, and is used to alter the ordering of changes queued for processing by a connector for a target.

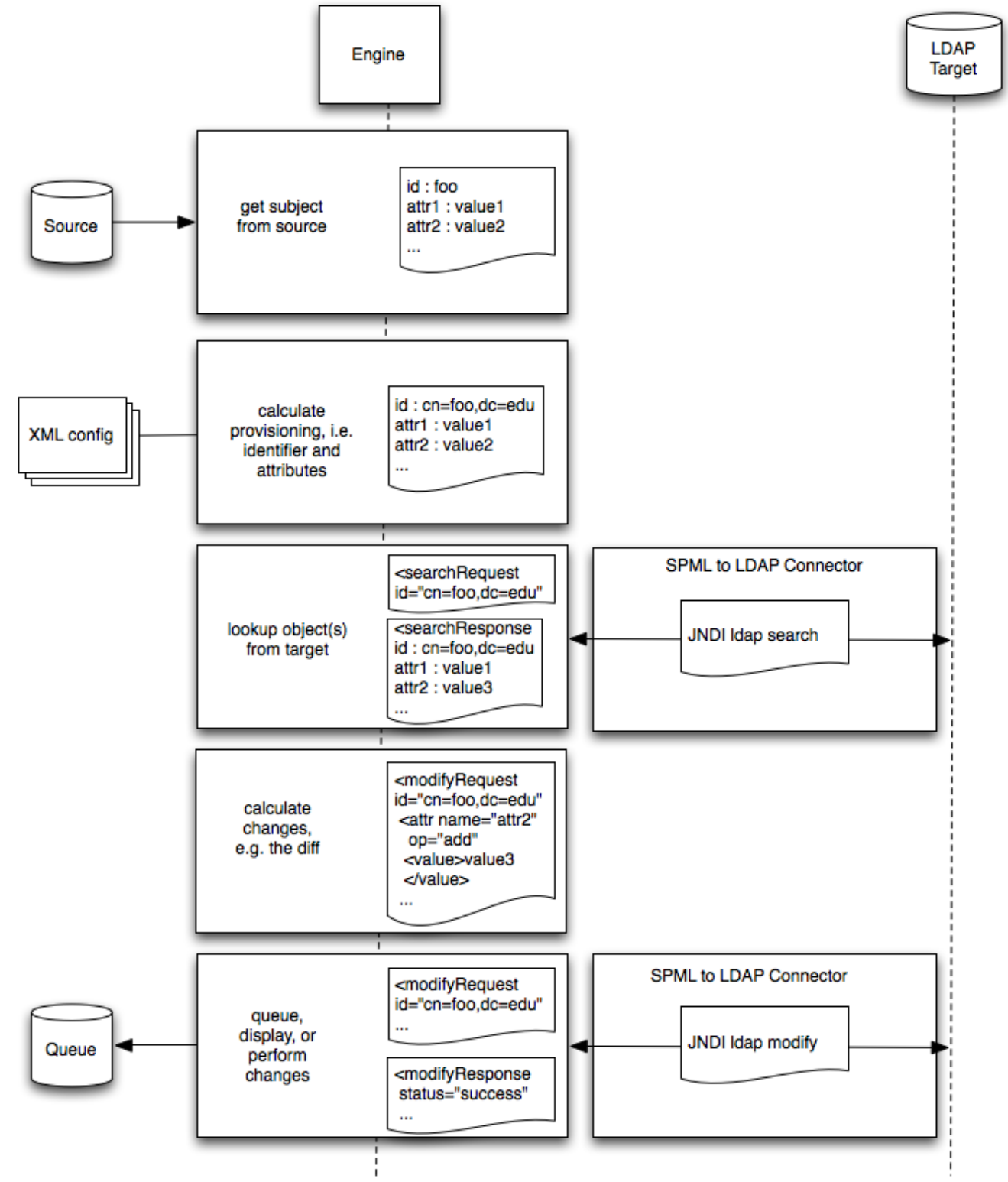
The presence of subject attributes indicates that only those attributes should be provisioned, otherwise all subject attributes are eligible for provisioning.

Sometimes it is desireable to provision specific targets only, by default all targets are eligible for provisioning.

Example : Attribute Value Modification

The following example describes a modification of a provisioned attribute value.

- Given a subject identifier, the engine queries sources for subject attributes.
- The engine rewrites subject attributes and identifiers suitable for provisioning to targets based on an XML configuration.
- The engine transmits SPML requests and responses to connectors, which translate SPML specific to their target, e.g. JNDI.



SPML to LDAP Example : AddRequest

Converting from SPML to LDAP is straightforward, the following example code handles an SPML add request :

```
public class LDAPConnector {

public AddResponse add(AddRequest spmlAddRequest) {

 Attributes jndiAttributes = new BasicAttributes();

 for(Attribute spmlAttribute : spmlAddRequest.getAttributes() {
  Attribute jndiAttribute = new BasicAttribute(spmlAttribute.getName());
  for(String value : spmlAttribute.getAttributeValues()) {
    jndiAttribute.add(value);
  }
  jndiAttributes.add(attribute);
 }

 ldap.createSubcontext(spmlAddRequest.getIdentifierString(), jndiAttributes);
```

Provisioning Configuration : Sharing Configuration Across Targets

Memphis considers our XML provisioning configuration file to represent/reflect our business logic. By versioning our configuration, we have some notion of change history.

We often use the same provisioning logic for multiple LDAP directory targets, such as for parallel deployments of production and test directory services. It is useful, therefore to share an XML representation of provisioning logic amongst several target directores :

```
<Target id="ad, ad-test" >
 ...provisioning logic...

<TargetConnection id="ad
 URL="ldaps://ad.memphis.edu"
 search_base="dc=uom,dc=memphis,dc=edu"
 ...
</Target>

<TargetConnection id="ad-test
 URL="ldaps://ad-test.memphis.edu"
 search_base="dc=test,dc=memphis,dc=edu"
 ...
</Target>
```

Currently, ldappc requires multiple ldappc.xml configuration files in order to provision multiple LDAP directories, which rules out sharing provisioning logic between targets unless the ability to include other xml configuration files is supported (e.g. via <xi:include />).

Graceful Provisioning

The ability to disable (SPML suspend) or enable (SPML resume) an object is part of Memphis' graceful de-provisioning. For example, before deleting a person from Active Directory, the object is disabled (via userAccountControl or expired via accountExpires) so that it appears to have been deleted. After a configurable delay (e.g. 30 days) has elapsed, the disabled object is then deleted. By disabling an object before deleting, it is possible to roll-back and preserve attributes of a provisioned object that are outside of the provisioning process (e.g. the Active Directory SID).

Enabling, disabling, and re-enabling may need to be performed immediately, after a configurable delay, or on a specified date. An LDAP directory target should also be able to return whether or not an object is enable or disabled (the SPML 'active' operation).

In addition to enabling and disabling, Memphis' provisioner includes the concept of no_provisioning, which simply means that a subject in a source should not be provisioned. By doing so, we have control over which subjects in a source system we provision, and provide some ability to recover from an inadvertent group delete. Using our custom web UI to Grouper, when someone deletes a group they actually set a no_provisioning attribute. By doing so, if the person inadvertently deleted the group from Grouper, the deleted group can be restored by simply removing the no_provisioning attribute.

```
<ProvisioningControl>
 <Switch>
  <Case registryAttribute="role" anyValue="Student, Employee, Faculty">ENABLED</Case>
  <Case scriptlet="activeEntitlement" args="urn:mace:...primaryAD">ENABLED</Case>
  <Case scriptlet="expiredEntitlement" args="urn:mace:...:primaryAD">DISABLED</Case>
  <Case registryId="role" anyValue="RecentStudent, RecentEmployee, RecentSponsored">DISABLED</Case>
  <Default>NO_PROVISIONING</Default>
 </Switch>
</ProvisioningControl>
```

The above example will (1) provision Students, Employees, and Faculty, or any account with a "primaryAD" entitlement, (2) disable expired "primaryAD" entitlements and recent students, employees, or other sponsored accounts, and (3) by default does not provision.

Identifier Mapping

Memphis' provisioner calculates object identifiers (PSO-IDs) for target LDAP directories. For example, the following calculates an ActiveDirectory DN :

```
<IdentifierMapping identifierType="urn:oasis:names:tc:SPML:1:0#DN" >
 <Value>CN=<RegistryAttribute id="REGISTRY.uid" />,OU=UoMPeople,DC=uom,DC=memphis,DC=edu</Value>
</IdentifierMapping>
```

In order to delete a provisioned object based on a subject identifier, either the target identifier must be calculated solely based on the subject identifier or the target provisioned object must contain the subject identifier as an indexed, searchable attribute.

Ldappc identifies subjects in a target LDAP directory via searches :

```
<source-subject-identifiers>
 <source-subject-identifier source="jdbc" subject-attribute="id">
  <ldap-search
    base="ou=people,dc=example,dc=edu"
    scope="onelevel_scope"
    filter="(&(examplePersonId=\{0\})(objectclass=examplePerson)))" />
 </source-subject-identifier>
</source-subject-identifiers>
```

Perhaps finding subjects in a target LDAP directory both via calculated identifiers and LDAP searches is desireable. Using searches may be more flexible but less performant than calculating identifiers.

Pluggable Provisioned Attributes

Ldappc is capable of provisioning attributes mapped from Grouper to a target LDAP directory :

```
<group-attribute-mapping ldap-object-class="">
 <group-attribute-map
    group-attribute="aci"
    ldap-attribute="aci"
    ldap-attribute-empty-value="" />
</group-attribute-mapping>
```

At Memphis, we compute rather than store the value of many provisioned attributes, in such a way that our provisioner's configuration represents our business logic.

For example, to provision a mail attribute as uid@memphis.edu :

```
<ProvisionedAttribute id="mail" >
 <Value><RegistryAttribute id="uid">@memphis.edu</Value>
</ProvisionedAttribute>
```

For more complex attribute provisioning Memphis once used BeanShell, although we've discontinued such us for performance reasons. The following would also provision the mail attribute as uid@memphis.edu, where getAttribute() is a compiled BeanShell method :

```
<ProvisionedAttribute id="mail" >
 return getAttribute("uid") + @memphis.edu
</ProvisionedAttribute>
```

For provisioned attributes that are too complex or too ugly to represent in XML, such as our ActiveDirectory homeMDB attribute, a provisioned attribute may make use of a complied class :

```
<ProvisionedAttribute id="homeMDB" implementation="edu.memphis.ActiveDirectory.homeMDB" />
```

We have found that simple switch-case statements are useful and handle most circumstances. For example, the following provisions the displayName attribute as either SELFSELECTED.name, ERP.name, or as last resort REGISTRY.uid :

```
<ProvisionedAttribute id="displayName" >
 <Switch>
  <Case registryAttribute="SELFSELECTED.name" />
  <Case registryAttribute="ERP.name" />
  <Default><RegistryAttribute="REGISTRY.uid" /></Default>
 </Switch>
</ProvisionedAttribute>
```

Some calculated provisioned attributes may be shared across multiple target LDAP directories, e.g. a displayName attribute which is calculated based on first, middle, and last name. For this reason, it is useful for a provisioned attribute XML representation be referenced for multiple targets.

Expanding ldappc to support pluggable attributes seems reasonable, given Memphis' experience.

Verify and Repair

At Memphis, we rely on the ability of our provisioning system to produce an SPML representation of changes that would be made (verify) in addition to actually making those changes (repair). The verify mode of operation is useful to determine how out of sync a target directory is, and to test code or configuration changes.

For example, here is an SPML 1.0 representation of the replacement of an ActiveDirectory attribute :

```
<spml:modifyRequest xmlns:spml='urn:oasis:names:tc:SPML:1:0' xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core'>
  <spml:identifier type='urn:oasis:names:tc:SPML:1:0#DN'>
    <spml:id>CN=tzeller,OU=UoMPeople,DC=uom,DC=memphis,DC=edu</spml:id>
  </spml:identifier>
  <spml:modifications>
    <dsml:modification name='textEncodedORAddress' operation='replace'>
      <dsml:value>X400:c=US;a= ;p=UM;o=Exchange;s=Zeller;g=tzeller;</dsml:value>
    </dsml:modification>
  </spml:modifications>
</spml:modifyRequest>
```

Ldappc does not currently speak SPML, however, it may be beneficial for ldappc to produce SPML or LDIF dependent upon a site's needs. SPML is useful for non-LDAP targets (e.g. Memphis' ERP portal, Luminis) and integration with a site's existing identitiy management infrastructure.

Incremental Provisioning

Currently, ldappc is designed to provision groups and memberships en masse. For example, given a Grouper stem, ldappc will provision all groups subordinate to that stem, including deleting groups from the target LDAP directory that are not children of the given stem.

Ldappc's current method of operation might be described as 'batch'. In order to provision in real-time, a single group or membership or some subset of groups and memberships will need to be provisioned.

It is desireable for ldappc to provision incrementally in real-time as a daemon attached to a notification queue and to provision in batch triggered at intervals by a scheduler or at the whim of a system administrator.

Notifications

Memphis' change queue contains sequential notifications which consist of three elements : (1) a subject identifier, (2) optional subject attributes, and (3) a priority. The Subject to be provisioned is returned from a Source using the subject identifier. The presence of optional subject attributes determines whether the provisioning occurs greedily or non-greedily, as discussed below. Finally, the priority allows the provisioning system to process time sensitive operations (e.g. password changes) independent of the length of the queue.

Greedy Attribute Provisioning

Provisioning systems at Memphis have alternated between being largely greedy and non-greedy, however it has become clear that a provisioning system needs to be both. A greedy system provisions all attributes of an object, while non-greedy provisions a subset of attributes, and even a subset of an attribute's values.

One particular provisioning system at Memphis was non-greedy and made LDAP modifications on a per-attribute basis. For example, if a password was changed in the source registry then the non-greedy system would perform an LDAP modification on the password attribute. Over time however, LDAP modifications would sometimes fail, and changes would need to be re-played chronologically, often at a system administrator's expense. When resubmittal of non-greedy changes became difficult, out-of-band scripts were used to repair all attributes (greedily).

As a response to the gradual divergence that occurred over time with the non-greedy provisioner, the next system at Memphis provisioned greedily. If a password was changed in the source person registry, then all of the attributes of the person were subject to provisioining. The greedy system maintained target LDAP directories much closer in sync with the source registry, since provisioning failures were automatically corrected upon the next attempted provisioning. However, when group memberships were introduced to this greedy system, performance suffered when provisioning large groups in real-time. For example, with greedy provisioning of the "All Students" group, all members were queried from our source database and all members were slurped from the target LDAP directory whenever a member was added or removed.

Greedy and non-greedy provisioning are appropriate at different times, and the provisioning system should ideally provide both.

Deletes

Deleting a provisioned object from a target LDAP directory upon a Subject's removal from a Source is an interesting situation. For ldappc to delete provisionings for a Subject that can not be found in a Source, the identifier of the Subject must be enough to uniquely identify any provisioned objects.

As the SPML 2.0 specification relates, deleting a provisioned object from a target LDAP directory may or may not involve deleting references to the deleted object.

Referential Integrity

Some LDAP directories maintain referential integrity between group and membership objects (ActiveDirectory) while others do not (RedHat, vanilla OpenLDAP). The provisioner may need to update both the member and hasMember attributes of group's and members. This provisioning behavior should be configurable.

Person or Group Centric Provisioning

Ldappc currently provisions group-centrically : it updates a group and optionally it's members. It may also be desireable to provision person-centrically : provision a membership and optionally any referenced groups. In other words, the notification change queue might consist of group or member identifiers.

(A)synchronous Operation

Ldappc is a synchronous provisioner, which seems simpler and easier to implement than asynchronous provisioning. In synchronous provisioning, the provider (ldappc) waits for the target LDAP directory to complete a request before sending the next request. In asynchronous provisioning, the provider (ldappc) sends requests and then queries the target LDAP directory for the status of those requests. Memphis had an asynchronous system which provisioned a Novell directory because at that time Novell did not provide an LDAP interface. The primary drawback of the asynchronous system was the difficulty for a systems administrator to diagnose problems or determine status because of the number of moving parts.

Bulk Updates

Memphis' provisioners do not support bulk updates, whether they be delete or modify operations. Typically bulk updates involving a large number of objects are the realm of a system administrator's ldap modification tools, and are performed out-of-band of the provisioning system. Potentially, deleting a Grouper stem and all of it's subordinate groups may be considered a bulk delete.

Caching

Ldappc supports caching subjects and groups during provisioning to reduce LDAP or Source lookups. During incremental provisioning, when attached to a notification queue, such caching will likely need to be disabled or scoped.