

The following document summarizes the discussion to date on modeling relationships in SPML 2.0. It is intended to record common understanding, to promote discussion and to be a contribution artifact for the generation of the 2.0 specification. It is in no way binding or concrete.

Objectives for Relationships in SPML 2.0

1. To provide within the protocol explicit support for expressing relationships between Provisioning Service Objects (PSOs). (NOTE: We will not explicitly support expressing relationships between any other protocol or model elements. This would have to be done at the attribute level in target schema.)
2. To provide extensible support for the mechanics of expressing relationships *without specifying definitive concrete models for specific relationships*. That is to say, we will not define what a role, group, or hierarchy actually means, although we should not do anything that prevents cooperation at this level via some form of optional binding or interface.
3. To allow the following types of relationships to be expressed:
 - a. **Containment** (i.e., hierarchy) at the point of PSO creation and as an optional interface. For example, an organizational unit must be created beneath an organization or an organizational unit. After creation, an organizational unit can be moved beneath another organization or organizational unit.
 - b. **Reference** (i.e., connections) at the point of PSO creation and as an optional interface. For example, a Group object may refer to Account objects as its members. Each Account may also refer to any Groups to which it belongs.

Statements about Relationships in SPML 2.0

1. **Containment and reference relationships are not concrete objects.** An instance of such a relationship is not a PSO and does not have an ID. Instead, an instance of a reference relationship is represented as an XML element or attribute that connects two PSOs, each of which has an ID.

*It is possible (but is not part of the specification) to express more complex relationships using the building blocks exposed by this simple model. For example, a relationship that is “stateful” and persistent could be represented as a PSO that refers to two connected objects. One might also represent such a relationship as a PSO that is bound beneath one object (its *parent* by containment) and that refers to another object (its *referent* by a reference relationship).*

For example, a user’s connection to a RACF Group has state. The RACF

CONNECT command allows one to specify whether the user has the AUDITOR, GRPACC, OPERATIONS, or SPECIAL privileges. The RACF CONNECT command also allows one to specify a REVOKE date or RESUME date (as well as immediate revocation or resumption) on a user's connection to the group.

In order to model this in SPML 2.0, one would have to define RACF Group Membership as a type of PSO. Each instance of RACF Group Membership would have Boolean attributes (defaulting to false) for AUDITOR, GRPACC, OPERATIONS, and SPECIAL. RESUME and REVOKE could be represented as date-valued attributes.

Each RacfGroupMembership object could be *bound* beneath a user object and *refer* to a group. That way, if the user is deleted, the user's group memberships also disappear. (If groups were more volatile than users, it might be better to have each RacfGroupMembership object bound beneath a group and refer to a user. For that matter, it is equally valid to represent each RacfGroupMembership as an independent object that refers to a group and also refers to a user.)

2. **A PSO can participate in only a single containment relationship.** That is to say, any object has at most one parent.
3. **A PSO can participate in any number of reference relationships.** For example, an Account can refer to any number of Groups.
4. **A PSO can participate in multiple types of reference relationships.** For example, an Account might refer to Groups by means of a "memberOf" relationship and to Roles by means of a "hasRole" relationship.
5. **Any type of reference relationship is optional and multi-valued.** For example, an Account might refer to no Group by means of a "memberOf" relationship.
6. **An optional interface defines Containment operations.** Containment operations include:
 - a. **createChild(parentID, ...)**
// creates an object beneath a specified parent.
 - b. **setParent(childID, parentID)**
// moves an object beneath a specified parent.
 - c. **getParent(childID)**
// returns the parent of a specified object.
 - d. **listChildren(parentID)**
// returns the ids of objects beneath a specified parent.
// could have a 'oneLevel' or 'allLevels' parameter.

7. **Each Target advertises Containment Rules.** In much the same way as a Target exposes its capabilities, a target may also expose containment rules. These rules would specify what types of objects each type of object may contain. For example, suppose that an `OrganizationalUnit` can be bound only beneath an `Organization` or an `OrganizationalUnit`. There are many ways to do this, but one can imagine XML as simple as the following:


```

<Target id=" " >
  ...
  <ObjectType name='Organization'>
    <MayContainObjectType name='OrganizationalUnit'/>
  </ObjectType>
  <ObjectType name='OrganizationalUnit'>
    <MayContainObjectType name='OrganizationalUnit'/>
  </ObjectType>
  ...
</Target>

```
8. **An optional interface defines Connection operations** (for reference relationships). Connection operations include:
 - a. **connect (fromID, toID, connectionType)**
// creates an object beneath a specified parent.
 - b. **disconnect(fromID, toID, connectionType)**
// moves an object beneath a specified parent.
// 'toID' and 'connectionType' could be optional.
 - c. **listConnected(fromID, connectionType, objectType)**
// returns the ids of objects beneath a specified parent.
// could have a 'oneLevel' or 'allLevels' parameter.

Other Issues (Resolved):

1. **Should the core 'create' operation specify containment? (No)**
We talked at one point about adding a 'parentID' argument to the 'create' operation. If no parent were specified, the created object would be bound directly beneath the target.

It is asymmetric (and in that sense bad) to have a core operation specify containment on initial object creation when no other core operation allows one to modify or to query containment. If we decide to make containment part of the core, then we should also add 'getParent', 'setParent' and 'listChildren' operations to the core interface.

Reviewers decided that it is better (although not great) to define a new 'createChild' operation (that is largely redundant with the core 'create' operation) for an optional Containment interface.
2. **Must we enforce cardinality for connection types? (No)**
Do we need to support required relationships? Do we need to support

reference relationships with a specific number of objects? If so, then it seems we must allow reference relationship types to be defined, so that the definer may specify the cardinality of the relationship (see Issue#3 below).

For example, we might allow the cardinality of a relationship type to be specified as one of the following:

0+ // an optional relationship to any number of referents
1? // an optional relationship to exactly one referent
1 // a required relationship to exactly one referent
1+ // a required relationship to at least one referent
N // a required relationship to exactly N referents
N? // an optional relationship to exactly N referents

Reviewers decided not to enforce cardinality. The benefit does not justify complicating both the protocol specification and its implementation.

3. **Must connection (reference relationship) types be defined? (No)**

If not, how do we know the direction of a relationship? If not, how do we know for what types of objects each connection type is valid?

It may be possible for the target to define (in a manner similar to the way that target capabilities define which optional interfaces are supported for each type of object) which types of connections to which types of objects are valid for each object type. One might imagine XML like the following:

```
<Target id=" " >
...
  <ObjectType name='Account' schemaRef='<XPATHorQNAME>'>
    <ConnectionType name='memberOf' toObjectType='Group' />
    <ConnectionType name='hasRole' toObjectType='Role' />
  </ObjectType>
  <ObjectType name='Group' schemaRef='<XPATHorQNAME>' />
...
</Target>
```

Reviewers decided NOT to define connection types, noting that there was a danger of getting carried away with defining, validating, and enforcing meta-data for reference relationship types (especially when that data might be updated by other actors who do not enforce the meta-data).