06 - associating and extending models

Associating

The association of two models means that one model contains a reference(id) of another model the association can be optional or mandatory. The association of two models only require the modification of the models APIs and the meta's. We will use the last sample from "05 - sql alchemy support" chapter, so in order to associate one entity with another entity we need a new entity. Lets say that to the **User** model we want to associate a **UserType** model, first we will need to create and API and implementation for the user type as we did for the user.

sample_plugin.api.user_type

```
'''
Created on Apr 9, 2012

@package: simple plugin sample
@copyright: 2011 Sourcefabric o.p.s.
@license: http://www.gnu.org/licenses/gpl-3.0.txt
@author: Gabriel Nistor

The API descriptions for user type sample.
'''

from ally.api.config import query, service
from ally.api.criteria import AsLike
from sample_plugin.api import modelSample
from sql_alchemy.api.entity import Entity, QEntity, IEntityService

# --------------------------------------------------------------------

@modelSample
class UserType(Entity):
    '''
    The user type model.
    '''
    Name = str

# --------------------------------------------------------------------

@query
class QUserType(QEntity):
    '''
    The user type model query object.
    '''
    name = AsLike

# --------------------------------------------------------------------

@service((Entity, UserType), (QEntity, QUserType))
class IUserTypeService(IEntityService):
    '''
    The user type service.
    '''
```

This is the user type API, is actually the user API adjusted to represent the user type, and now the meta.

sample_plugin.meta.user_type

```
'''
Created on Apr 9, 2012

@package: simple plugin sample
@copyright: 2011 Sourcefabric o.p.s.
@license: http://www.gnu.org/licenses/gpl-3.0.txt
@author: Gabriel Nistor

Mapping for the user type model.
'''

from ally.support.sqlalchemy.mapper import mapperModel
from sample_plugin.api.user_type import UserType
from sample_plugin.meta import meta
from sqlalchemy.schema import Table, Column
from sqlalchemy.types import String, Integer

# --------------------------------------------------------------------

table = Table('sample_user_type', meta,
              Column('id', Integer, primary_key=True, key='Id'),
              Column('name', String(20), nullable=False, unique=True, key='Name'))

# map User Type entity to defined table (above)
UserType = mapperModel(UserType, table)
```

We have defined the **sample_user_type** table almost like the **sample_user** table except that we had declared the name as a unique column, we don't want multiple types with the same name. Last but not least we have the implementation.

sample_plugin.impl.user_type

```
'''
Created on Apr 9, 2012

@package: simple plugin sample
@copyright: 2011 Sourcefabric o.p.s.
@license: http://www.gnu.org/licenses/gpl-3.0.txt
@author: Gabriel Nistor

Simple implementation for the user type APIs.
'''

from sample_plugin.api.user_type import IUserTypeService, QUserType
from sample_plugin.meta.user_type import UserType
from sql_alchemy.impl.entity import EntityServiceAlchemy

# --------------------------------------------------------------------

class UserTypeService(EntityServiceAlchemy, IUserTypeService):
    '''
    Implementation for @see: IUserTypeService
    '''

    def __init__(self):
        EntityServiceAlchemy.__init__(self, UserType, QUserType)
```

After the user type modules have been defined you just need to start the application and because of the AOP configurations we have made the user type will appear automatically as one of the REST services that can be accessed via http://localhost/resources/Sample/UserType. We will have of course an empty list here to begin with, so lets insert a user type.

method:	**POST**
Accept:	**xml**

Content-Type:	**xml**

http://localhost/resources/Sample/UserType

```
<UserType>
    <Name>Administrator</Name>
</UserType>
```

RESPONSE:

```
<?xml version="1.0" encoding="UTF-8"?>
<UserType href="http://localhost/resources/Sample/UserType/1">
    <Id>1</Id>
</UserType>
```

If you try to make the **POST** again you will receive as a response,

```
<?xml version="1.0" encoding="UTF-8"?>
<error>
    <message>Already an entry with this value</message>
    <code>404</code>
</error>
```

this is because we declared the name as unique and the binded validation automatically checks if the provided name is not present in the database. The sample until this point can be found here.
Ok so now we have the **User** model and the **UserType** model we just have to see how we can specify to the user the user type, first we need to change the user API.

sample_plugin.api.user

```
from ally.api.config import service, query
from ally.api.criteria import AsLike
from sample_plugin.api import modelSample
from sample_plugin.api.user_type import UserType
from sql_alchemy.api.entity import Entity, QEntity, IEntityService

# --------------------------------------------------------------------

@modelSample
class User(Entity):
    '''
    The user model.
    '''
    Name = str
    Type = UserType

...
```

We added to the **User** model a new attribute called **Type**, we assign as a value the model class we want to associate with, in this case the **UserType**, the ally framework knows now that **Type** is actually a reference to a **UserType** object. The actual value that is contained in **Type** is the model id value of the **UserType**, basically the **Type** will not contain an entire **UserType** object it will contain just an id of a **UserType**. Now we need to modify the meta in order to contain also the type.

sample_plugin.meta.user

```
from ally.support.sqlalchemy.mapper import mapperModel
from sample_plugin.api.user import User
from sample_plugin.meta import meta
from sqlalchemy.schema import Table, Column, ForeignKey
from sqlalchemy.types import String, Integer
from sample_plugin.meta.user_type import UserType

# --------------------------------------------------------------------

table = Table('sample_user', meta,
              Column('id', Integer, primary_key=True, key='Id'),
              Column('name', String(20), nullable=False, key='Name'),
              Column('fk_user_type', ForeignKey(UserType.Id, ondelete='RESTRICT'), nullable=False,
key='Type'))

# map User entity to defined table (above)
User = mapperModel(User, table)
```

We added a new column to the table that is a foreign key to the user type table, you notice that when we define relations with other models we always need to use the meta class, in this case the **UserType** mapped in the module **sample_plugin.meta.user_type**. Because the logic in the services is not modified by the newly added information we don't need to modify anything in the service APIs or implementations.
In order to test this, before we start the application we need to delete the **sample.db** file in the distribution, this will force the creation of the new **sample_user** table that contains now also the user type foreign key, also to get a better error message that will also tell which attribute is the problem change the configuration **explain_detailed_error** to **true** in the "application.properties" file. Now lets insert a user, keep in mind that our database is empty.

method:	**POST**
Accept:	**xml**
Content-Type:	**xml**

http://localhost/resources/Sample/User

```
<User>
    <Name>Jhon Doe</Name>
</User>
```

RESPONSE:

```
<?xml version="1.0" encoding="UTF-8"?>
<error>
    <code>404</code>
    <User>
        <Type>Expected a value</Type>
    </User>
</error>
```

So we get an error of **Invalid resource** because the **User.Type** is not specified, that is because when we defined the table we set the **nullable** flag to **false** for the **Type** column. Since our database is empty lets insert a user type.

method:	**POST**
Accept:	**xml**
Content-Type:	**xml**

http://localhost/resources/Sample/UserType

```
<UserType>
    <Name>root</Name>
</UserType>
```

RESPONSE:

```
<?xml version="1.0" encoding="UTF-8"?>
<UserType href="http://localhost/resources/Sample/UserType/1">
    <Id>1</Id>
</UserType>
```

Now that we have user type of id **1** lets try to insert the user having this user type.

method:	**POST**
Accept:	**xml**
Content-Type:	**xml**

http://localhost/resources/Sample/User

```
<User>
    <Name>Jhon Doe</Name>
    <Type>2</Type>
</User>
```

RESPONSE:

```
<?xml version="1.0" encoding="UTF-8"?>
<error>
    <code>404</code>
    <User>
        <Type>Unknown foreign id</Type>
    </User>
</error>
```

I had intentionally set the type as **2** because there is no user type in the database with that id and as you see the binded validations will deliver a message telling us that the id we had specified is invalid. Lets to this again but with a valid id.

method:	**POST**
Accept:	**xml**
Content-Type:	**xml**

http://localhost/resources/Sample/User

```
<User>
    <Name>Jhon Doe</Name>
    <Type>1</Type>
</User>
```

RESPONSE:

```
<?xml version="1.0" encoding="UTF-8"?>
<User href="http://localhost/resources/Sample/User/1">
    <Id>1</Id>
</User>
```

Now we have successfully inserted a user in the database that also has a type, so now if you access http://localhost/resources/Sample/User/1

http://localhost/resources/Sample/User/1

```
<?xml version="1.0" encoding="UTF-8"?>
<User>
    <Type href="http://localhost/resources/Sample/UserType/1">
        <Id>1</Id>
    </Type>
    <Id>1</Id>
    <Name>Jhon Doe</Name>
</User>
```

, you have the new user model with a user type reference. The sample code can be found here.

Extending

The extending is when a service provides models based on another model id, even if the provided models are not associated with the other model. The extending requires only the modification of the service's APIs and implementations.

sample_plugin.api.user

```
from ally.api.config import service, query, call
from ally.api.criteria import AsLike
from ally.api.type import Iter
from sample_plugin.api import modelSample
from sample_plugin.api.user_type import UserType
from sql_alchemy.api.entity import Entity, QEntity, IEntityService

...

# --------------------------------------------------------------------

@service((Entity, User), (QEntity, QUser))
class IUserService(IEntityService):
    '''
    The user service.
    '''

    @call
    def getUsersByType(self, typeId:UserType.Id, offset:int=None, limit:int=None,
q:QUser=None)->Iter(User):
        '''
        Provides the users that have the specified type id.
        '''
```

We added a service method that will deliver all the users that have the specified type id, also the service method will allow the specification of offset, limit and user query.

sample_plugin.impl.user

```
from sample_plugin.api.user import IUserService, QUser
from sample_plugin.meta.user import User
from sql_alchemy.impl.entity import EntityServiceAlchemy

# ----------------------------------------------------------------------

class UserService(EntityServiceAlchemy, IUserService):
    '''
    Implementation for @see: IUserService
    '''

    def __init__(self):
        EntityServiceAlchemy.__init__(self, User, QUser)

    def getUsersByType(self, typeId, offset=None, limit=None, q=None):
        '''
        @see: IUserService.getUsersByType
        '''
        return self._getAll(User.Type == typeId, q, offset, limit)
```

The implementation is very easy because it makes use of the **_getAll** method inherited from **EntitySupportAlchemy** that allows for an easy get of models from database. So now we have a service method that provides user models based on a user type, if we access http://localhost/resources/Sample/UserType/1 we get:

http://localhost/resources/Sample/UserType/1

```
<?xml version="1.0" encoding="UTF-8"?>
<UserType>
    <Id>1</Id>
    <Name>root</Name>
    <User href="http://localhost/resources/Sample/UserType/1/User/"/>
</UserType>
```

Now, beside the **UserType** model data we also have a new reference for the **User** models that belong to this **UserType**, this reference will call our new service method. The idea is that we are able to add information on existing models like **UserType** from a different service than the main main user type service. The sample is available here.