Sakai Entity Model

Feb 15, 2006

Please direct questions or comments about this document to:
Glenn R. Golden, ggolden@umich.edu

DRAFT

Introduction

This documents the Sakai framework's Entity model, how it looks in Sakai 2.1, and how it has changed since it was called the "resource" model in Sakai 2.0.

Entity

An Entity is data modeled by a Sakai application. The granularity of entities are set by the applications, and usually represent the different "real world" domain things that the application models.

Examples of entities include:

- chat message
- announcement channel
- resource or collection in content hosting
- site
- user

Entities are composed of many different units of data; these units are not usually entities. Examples of things that are not entities are:

- id
- created-by user
- last-modified-by time

Most applications surface their entity model by providing a Java interface (which will extend Entity), and a manager interface (which will extend EntityProducer) to provide access to the application's entities.

Sakai applications that produce entities are expected to take part in some common entity related features of Sakai, including:

- import and export
- entity sensitive security
- automatic entity creation / cleanup as sites come and go

- public entity display in the site browser
- entity references within Sakai (such as for attachments)

and more.

The Sakai Entity Model is designed to let an application “play the entity game” in Sakai. By declaring a manager in your application to be an “EntityProducer”, and by registering the manager component with the Sakai EntityManager at the component’s init() time, your application will be called on to take part in entity activities.

EntityManager

The Sakai EntityManager is an API used to:
- register entity producing applications
- get a list of all entity producers
- create Reference and ReferenceList objects for generic entity processing

This is a manager (also known as a service) available by discovery or injection using the Sakai Component manager, and also available by static cover.

EntityProducer

EntityProducer is an API that any entity producing application will satisfy to handle generic Entity related requests. These fall into these categories:
- Entity Reference processing
- Security processing
- Archive, merge and import
- Site lifecycle changed for related entities
- Public entity access

An application’s manager API that is an EntityProducer extends the EntityProducer interface, like this:

```
public interface AliasService
     extends EntityProducer
```

In the implementation component of the manager, it registers with the EntityManager in the init() method to declare that it’s an available entity producing application:

```
// register as an entity producer
m_entityManager.registerEntityProducer(this);
```

Entity

Entity is the API that all entity objects need to satisfy to be part of the Entity model in Sakai. It assures that the entity has an id, url, reference, properties, and a way to produce XML for export.

The Java class that the application defines that is the entity extend the Entity API, like this:

```
    public interface Alias
        extends Entity, Comparable
```

Entity Identification and Access

Every entity has an ID - a unique, never to be changed value, which you get from getId(). This id can be used with the entity's EntityProducer/manager/service to access the entity.

Every entity has a reference - a string that encodes the entity's id and the producer, each producer taking a different branch of the reference space. For instance, ContentHosting uses "/content/..." references, Alias uses "/alias/..." references. References are good for internal access to an entity, within Sakai code. Attachments are stored by reference, for instance.

We extend this idea to say that there can be multiple references to the same entity - each one handled by a different producer. The 'raw' or 'native' reference is when you ask the entities producer for the reference (you get "/content/..." for example from ContentHosting). But other references are possible. For Metaobj, you can get a reference to the XML document holding form data that lives in ContentHosting that looks like "/metaobj/content/...".

Every entity has a URL - which is simply the Access URL root pre-pended to the reference. As such, there are alternate URLs to match each alternate reference. This is good when you want to encode a link to access the entity from outside of Sakai.

Changes

This used to be called the “Resource Model” in Sakai 2.0. The term “resource” has been a source of confusion, since it’s used in content hosting and the “resources” tool. The term “entity” has grown to replace “resource” in design work recently, so that’s why the rename of the model.

Reference.java used to have a number of things going against it. It is an API but is a class rather than an interface. It has hard coded knowledge of all the possible applications that play the entity game – which means that it's a single point concentration that has to be edited to get a new application into Sakai. These problems have been cleared up. It is now a clean interface; to get a Reference you can no longer use code like:

```
	Reference ref = new Reference(str);
```

Instead, you must call the EntityManager, either through injection or cover, like this:

```
	Reference ref = EntityManager.newReference(str);
```

The implementation no longer has hard-coded knowledge of any Sakai applications. Instead if uses the list of registered EntityProducers, and asks them to parse reference strings and come up with entity related information from a Reference.

The ReferenceVector has been removed from the API. This is just a List, but implemented with a special reference specific implementation. When creating them, use the EntityManager:

```
	List attachments = m_entityManager.newReferenceList();
```
or
```
	List attachments =
	EntityManager.newReferenceList(listToCopy);
```

Instead of creating them from a `new` or a clone().

Many more service APIs extend EntityProducer; all that had hard coded knowledge in the Reference.java implementation. Those that don't want to participate in archive/merge or import indicate so with the:
```
	boolean willArchiveMerge();
```
and
```
	boolean willImport();
```

methods (returning false). These all register in their init() method. Most have the EntityManager injected. Parts of the code that need the manager that are not service components use the cover.

Locations in the Sakai Source Code

The Entity Model lives in the legacy-service/service module. It will be promoted sometime soon to Common. For now, it still lives in the package:

```
	package org.sakaiproject.service.legacy.resource;
```

Entity, EntityManager, Reference, and EntityProducer are in here. Edit, AttachmentContainer (and Edit), ResourceProperties (and Edit) are still here.

The implementation of the EntityManager is in the legacy/component module, in the package:

```
    package org.sakaiproject.component.legacy.entity;
```

<end>