KEEPER: Knowledge Engineering Environment for Provenance and Entity Registration

A reference implementation

Kenneth Baclawski – Maximo Gurmendez

Abstract:

This document describes a reference implementation of the KEEPER system, which acts as a model repository and gate keeping web service. This system provides a way register items such as models, documents and specifications in a flexible manner, by allowing each registration authority to work with its own set of procedures. The first part of this paper outlines the use cases, and basic concepts involved in the system. The second part specifies the design and implementation of the system.

Motivation

Many registration authorities require a formal procedure to handle models or documents before they become a standard. In order for a model to become an official standard (according to a registration authority), the following aspects need to be considered:

- The procedures that specify the process by which an item becomes a standard.
- A repository that stores the latest version of the item serving as a single point of entry for every relevant actor to query the items.
- The roles that the actors play in the accreditation process.
- The provenance of an item registered on the system.
- The accessibility of models in the repository.

The KEEPER system allows any organization to become a registration authority upon the authorization of an accreditor (a singular entity that makes this decision). Representatives of a registration authority called registrars, stewards and submitters are able to flexibly define and follow though the different procedures that will regulate the process of an item from its inception to the moment it becomes a standard. KEEPER is based on the ISO 11179, Metadata Repository specification [MDR].

Main Concepts

As suggested in the MDR specification, each actor in the KEEPER system is either a registrar, steward, submitter or read only user. Table 1 shows the role each actor plays in the system.

Actor Role	Description
Registrar	Directly represents a registration authority. Makes higher level decisions regarding the items in the registry.
Steward	Represents a registration authority through a particular registrar. Makes the lower level decisions regarding accreditation process.
Submitter	Is able to submit items to the registry as allowed by a steward.
Read Only User	Allows to access the models in the system in a read only manner.

As it can be appreciated in the table above, users form a hierarchy from the registrar to the read only user. Any user has a single "supervisor" which is the user that allowed the user into the system. For example when a registrar X allows a steward Y into the system, the supervisor of X is Y. In this manner each user responds to the user that allowed it into the system

The other main concept in the system is the actual item being subject to the registration process. We call this a "kept item" or an "administered item". Any kept item that is submitted to the registry goes through the registration process as required for each registration authority. If there is no prescribed process, then the kept item goes through a default process.

Whenever an item is submitted to the system by a submitter, a process may request the stewards or registrars to take action regarding its accreditation. The actual registrars or stewards that will make these decisions will be the supervisors of the submitter as established by the hierarchy.

Actors can be created in the system in two different ways: by direct creation or by request. For example a new submitter may be created directly be the steward or a submitter can submit the request to a particular steward and this will decide to approve it or not.

Use Cases

Use cases were defined by Prof. Baclawski's Software Engineering students, organized into different categories, and then formalized as an instance of Use Case Description Ontology [KB2010]. The following list shows the main use cases that the KEEPER system supports:

- Accredit Registration Authority
- Steward Registration
- Submitter Registration
- Read Only User Registration

- Upload Process Definition (or procedures)
- Register an Item
- Query an Item
- Update an Item
- Complete information as required by a process definition
- Update contact information
- Query process definitions
- Query contact information

A detailed description of these use cases can be found in [KBMG2001]. For illustrative purposes, consider the following scenario:

1. Jack, a registrar, uploads a procedure that says: "stewards need to validate models before they becomes a standard"
2. Submitter John uploads new model (the initial status is "Pending").
3. The workflow engine notifies the steward Paul about the new model (according to process definition)
4. Paul, the steward, logs on to the system, and approves the model.
5. The workflow engine modifies the status of the model, now being "standard".

Notice in the above scenario, how the system acts as a coordinator between the different actors towards the standardization of the model, and how the actual procedures are treated as **data** by system, modifying its behavior.

Component Design

The system includes 4 main components as it can be seen in Figures 1 and 2.

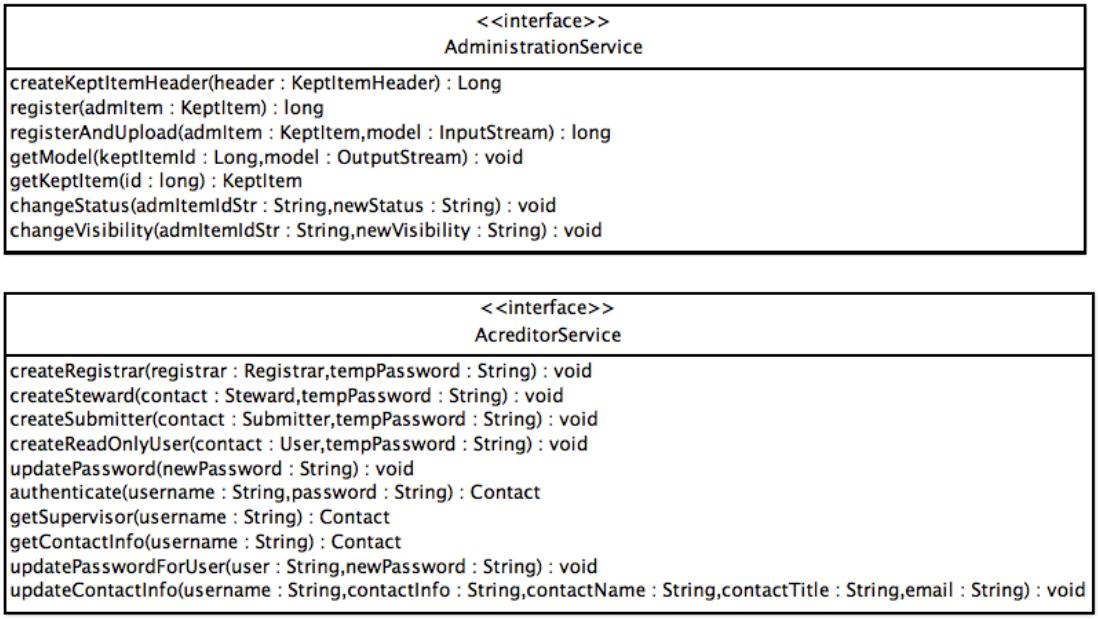

Figure 1

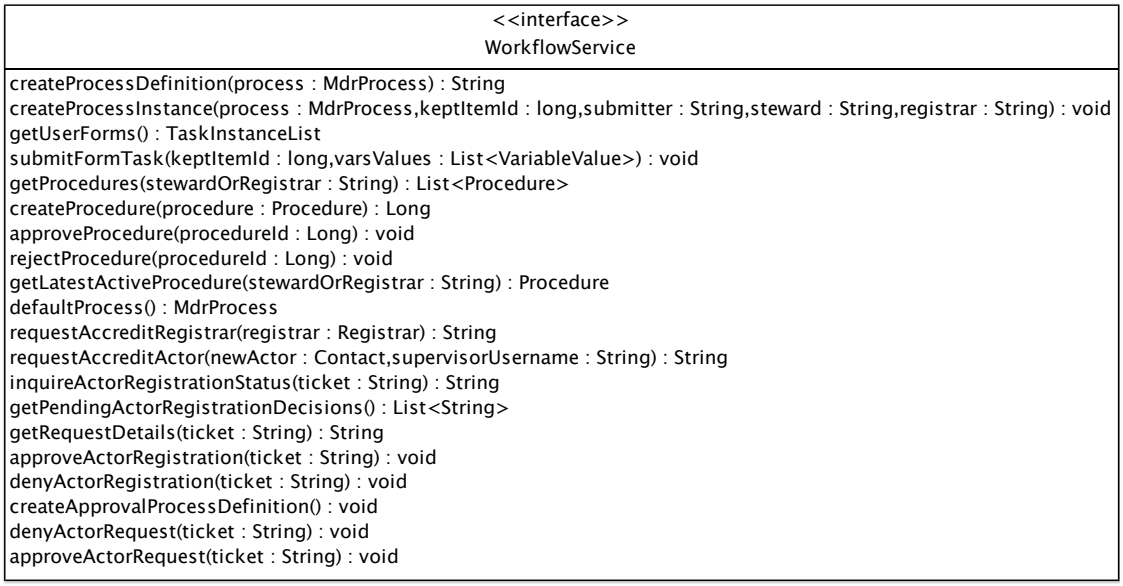

Figure 2

As it can be appreciated on these figures, the AdministrationService deals with the registration of an item in the system, while the WorkflowService deals with the different processes that actors undergo in the system. The AccreditorService refers to the creation of users in the system. These components interact within each other. For example, the WorkflowService must use the AcreditorService whenever it needs to create a user. The detailed descriptions of each of the methods are beyond the purpose of this document and the reader should consult the javadoc for these classes. These interfaces were implemented as classes that include both JEE annotations as

well as Web Services annotations. This enables the system to expose the interfaces both as EJBs and as a WSDL. Figure 3 shows the main classes that represent the users, kept items and procedures (process definitions).

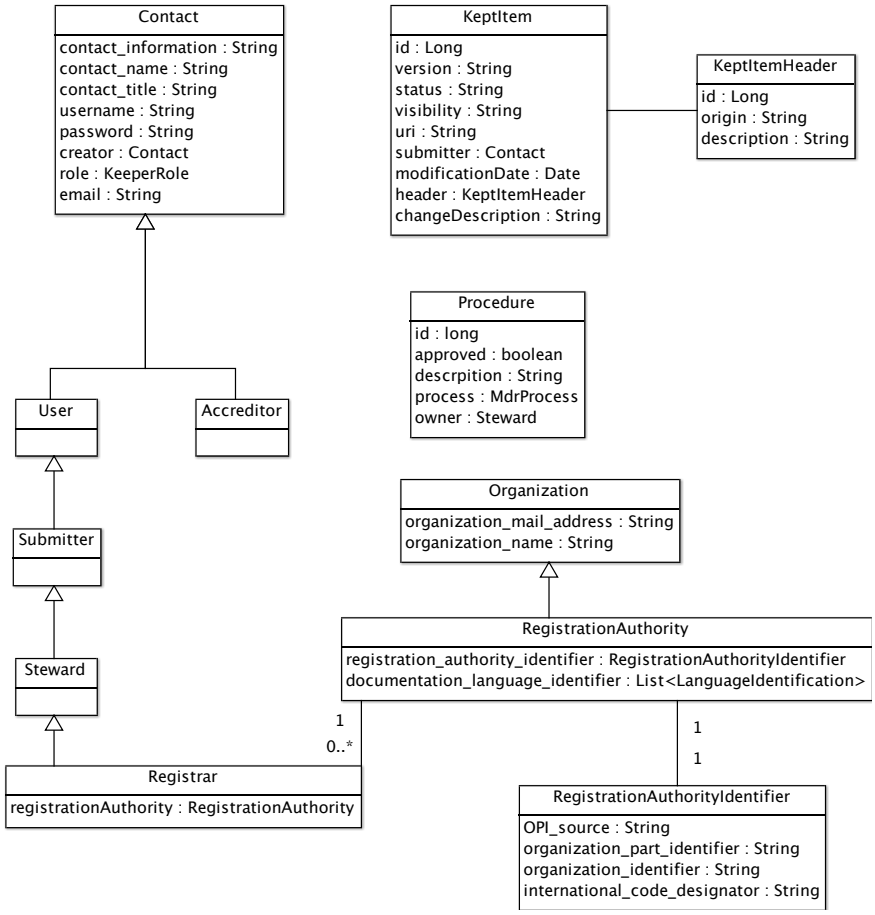

Figure 3

Note in Figure 3, that the Procedure class references an MDRProcess attribute. This attribute specifies the process that an item must undergo upon its registration. For example, an item may require the approval of both the steward and the registrar in order to become a standard. This results in a process definition that starts with the registration of the item and then the steward and registrar take turns in reviewing the item and deciding its validity. These users may need to input some information for the process to make a decision. This procedure is modeled according to the schema on Figure 4 that consists of states connected by transitions. These states can be tasks that users need to complete (such as the FormTask) or processes executed automatically by the system upon its execution (such as ChangeVisibilityState). Figure 5 shows a sample instantiation of this model (as an xml serialization, which is easier to read). Note how the process example includes the different states, transitions, assignees and conditions for the accreditation of the item.

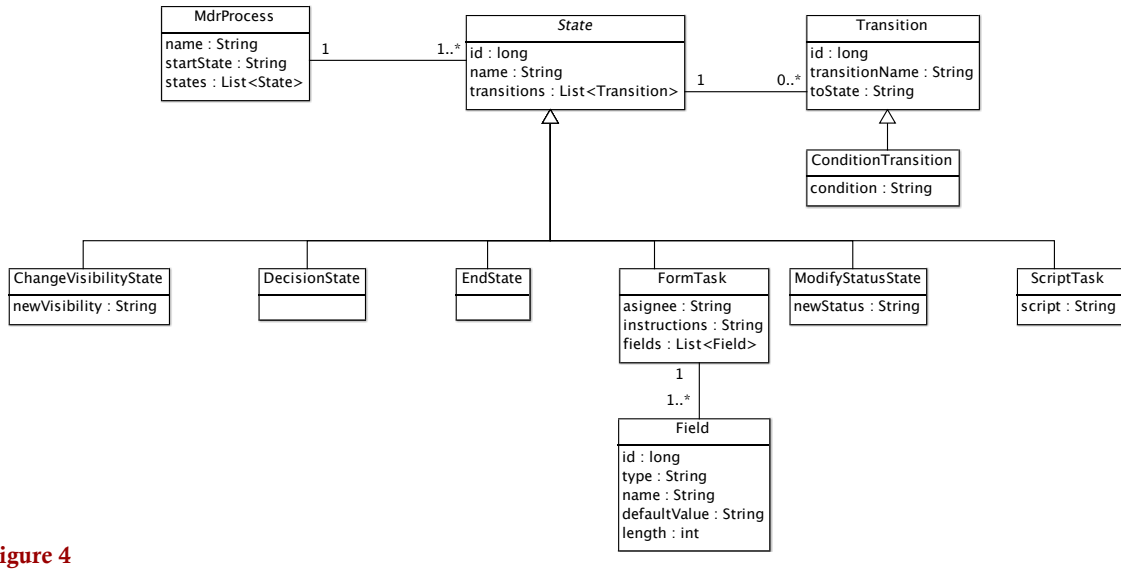

Figure 4

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<process>
        <name>process_requires_registrar_and_steward_aproval</name>
        <startState>Steward Approval</startState>
        <states xsi:type="formTask" >
                <name>Steward Approval</name>
                <instructions>
                   Dear Steward, Please approve this model by setting approveSteward=y.
               </instructions>
                <transitions>
                        <toState>Registrar Approval</toState>
                        <transitionName>toRegistrarAproval</transitionName>
                </transitions>
                <asignee>steward</asignee>
                <fields>
                        <length>10</length>
                        <name>approveSteward</name>
                        <type>java.lang.String</type>
                </fields>
        </states>

        <states xsi:type="formTask"
                <name>Registrar Approval</name>
                <instructions>
                Dear Registrar, Please approve this model by setting approveRegistrar=y.
                </instructions>
                <transitions>
                        <toState>Decide</toState>
                        <transitionName>toDecision</transitionName>
                </transitions>
                <asignee>registrar</asignee>
                <fields>
                        <length>10</length>
                        <name>approveRegistrar</name>
                        <type>java.lang.String</type>
                </fields>
        </states>
```

```
        <states xsi:type="decisionState" >
                <name>Decide</name>
                <transitions xsi:type="conditionTransition">
                        <toState>endState</toState>
                        <transitionName>toEnd</transitionName>
                        <condition>
                        #{approveRegistrar!='y' || approveRegistrar!='y'}
                        </condition>
                </transitions>
                <transitions xsi:type="conditionTransition">
                        <toState>mod</toState>
                        <transitionName>toMod</transitionName>
                        <condition>
                        #{approveRegistrar=='y' &amp;&amp; approveSteward=='y'}
                        </condition>
                </transitions>
        </states>
        <states xsi:type="modifyStatusState" >
                <name>mod</name>
                <transitions>
                        <toState>endState</toState>
                        <transitionName>toEnd</transitionName>
                </transitions>
                <newStatus>standard</newStatus>
        </states>
        <states xsi:type="endState">
                <name>endState</name>
        </states>
</process>
```

Figure 5

Registrars and stewards are able to upload procedures, such as that on Figure 5. Whenever a new item is submitted, the supervisor of the submitter (a steward) is used to determine which procedure to use (items always undergo the latest procedure uploaded by either a steward or a registrar). Registrars need to approve procedures of stewards. Suppose a steward has in place the procedure of Figure 5. Then the following set of calls (Figure 6) shows how the submitter would submit the item and the item would be checked by the steward and registrar to change the status from "pending" to "standard".

The Submitter submits the item:

```
// create a sample kept item instance
admItem = createKeptItem();

// initial status should be pending
assertEquals("pending",admItem.getStatus());

// register the item through the administration service
long idItem = adminService.register(admItem);
```

The Steward approves the item:

```
// checks its tasks
List<TaskInstance> tasks =  workflowService.getUserForms().getTaskInstances();
assertEquals(tasks.size(), 1); // there should only be one pending task for this user
TaskInstance task = tasks.get(0);
List<VariableValue> vars = new ArrayList<VariableValue>();
vars.add(new VariableValue("approveSteward", "y"));
workflowService.submitFormTask(task.getAdmItem(), vars);

// the item should still be pending as there is no change
admItem = adminService.getKeptItem(idItem);
assertEquals("pending",admItem.getStatus());
```

The Registrar approves the item:

```
tasks =  workflowService.getUserForms().getTaskInstances();
assertEquals(tasks.size(), 1); // there should only be one pending task for this user
task = tasks.get(0);
vars = new ArrayList<VariableValue>();
vars.add(new VariableValue("approveRegistrar", "y"));
workflowService.submitFormTask(task.getAdmItem(), vars);

// the item should have become a standard
admItem = adminService.getKeptItem(idItem);
assertEquals("standard",admItem.getStatus());
```

Figure 6

Many more examples such like those in Figure 6 can be found on the test case source files.

Implementation Details

The different interfaces were implemented as Enterprise Java Beans 3.0 by adding suitable annotations. Additional annotations were required to expose the methods as web services. The EJB annotations allow the beans to be deployed in any suitable JEE container, such as JBoss [JB] or Glassfish [GF]. However, this reference implementation uses the OpenEjb [OEJB], for two reasons: it does not require a container, and it contains a mechanism to automatically publish EJBs as WSDL1.1 [WSDL] compliant web services. Most methods require implicit authentication and in most cases JEE declarative security was used. Figure 7 shows a fragment of the Administration bean implementation. Note the @Stateless and @WebService annotations, the injection of the PersistenceContext used to persist and load objects from the database, and the SessionContext, which is used for programmatic security. The register method also uses declarative security by specifying the roles allowed for invoking the method. JEE beans do not need to know about the actual security protocol (secure HTTP, SSL, basic authentication, etc). This is potentially configured in the container without any need to modify the application source code.

```
@Stateless(name="Administration")
@WebService(portName = "AdministrationPort",
        serviceName = "AdministrationWebService",
         targetNamespace = "http://mdr.org/wsdl")
public class AdministrationServiceImpl implements AdministrationService {

        @PersistenceContext(unitName="mdr")
        EntityManager entityManager;

        @Resource
        SessionContext context;

        @Override   @RolesAllowed({"submitter","registrar","steward"})
        public long register(KeptItem admItem) throws RepositoryException {
        ...
        }
```

Figure 7

The model classes, such as those in Figures 3 and 4, are persisted in a database by annotating these classes with JPA annotations [JPA].

The Workflow bean uses the well-known workflow engine "Java Process Management Suite" [JBPM] to manage the different workflows required by the registration processes. JBMP allows one to create and instantiate process definitions. However, the power of JBMP exceeds that of KEEPER, reason for which a Façade of the process definition schema was created within KEEPER (this is, in fact, the class diagram on Figure 4). In the background, KEEPER transforms this simplified model into the schema required by JBPM.

Conclusions

As of this date, the KEEPER system is only a reference implementation and works mainly as a proof-of-concept thus is not fully ready for a production environment. More extensive testing needs to be performed, both for functional requirements and stress checks. Nevertheless its design is scalable, flexible, reusable and can be quickly deployed in a container.

One main area for improvement is its interface with other backend repository systems, capable of storing and versioning large models. Right now, the system stores all models locally or remotely through a URI. Additionally, KEEPER does not provide any kind of user interface. Applications that use KEEPER may require additional features, such as model browsing capabilities. Currently, KEEPER works as a single centralized repository. More work is required to provide interfaces for federation.

Regardless of its limitations KEEPER serves as a starting point to develop a more complete and integrated service, as requirements go through a process of refinement.

References

[MDR] International Standards Organization, Metadata Repository 11179. http://metadata-stds.org/11179/

[KB2010] Baclawski, K. Use Case Description Ontology, http://www.ccs.neu.edu/home/kenb/ontologies/

[KBMG2010] Baclawski,Gurmendez 2010, Ontolog Forum, Use Cases for the OOR.

http://www.ccs.neu.edu/home/kenb/ontologies/oor-usecase.xml

[JEE] Java Enterprise Edition Technology, Oracle, http://java.sun.com/javaee/technologies/javaee5.jsp

[WSDL] Web Services Description Language (WSDL) 1.1, 2001, http://www.w3.org/TR/wsdl

[EJB3] Enterprise Java Beans v3.0, Oracle, http://java.sun.com/products/ejb/

[OEJB] OpenEjb, Apache Foundation, http://openejb.apache.org/3.0/index.html

[JB] JBoss, Reg Hat, http://www.jboss.com/products/platforms/application/

[GF] Glassfish, Java Net, https://glassfish.dev.java.net/

[JPA] Java Persistence API, Oracle, http://java.sun.com/developer/technicalArticles/J2EE/jpa/

[JBPM] Java Business Process Management Suite, Red Hat, http://www.jboss.org/jbpm/