A conceptual framework to apply Sakai with contract and its practical implementation

Alejandro Sartorio
sartorio@cifasis-conicet.edu.ar

Guillermo Rodriguez

1-3 July 2008
Paris, France

http://www.mesadearena.edu.ar
Overview

Topics

• How to assist a new kind of requirement of e-learning under the perspective of Dynamic Hypermedia Device (DHD).

• Technological aspect about the evolution of Sakai towards Sakai with Contract.

• Our proposal of implementation of contracts to Sakai.
Technological aspect about the evolution of Sakai towards Sakai with Contract

Levels of evolution

Components to study (Theory)
- Complex relationship (Complex System)
- Common relationships (e-learning)
- Sensors (Application context-aware)
- Piece of Software (DHD)
DHD's requirement

Requirement definitions

- Reflect in the system's architecture the different level of changes (in run time) which are produced on the relationship under DH domain.

- Support evolution through the dynamic reconfiguration, without the services interruptions and minimizing its impact in the global system.

- Obtain Hypermedial / Relation adaptation with context-aware aspects.
DHD's requirement

Requirement definitions

- **Superposition** captured through morphisms and universal constructions (colimits).

- **Separation** between computation and coordination captured through functors that map systems to coordination interfaces.
Our interpretation of contract:

Contract is an abstraction for the modelling of interconnections. Whereas design-by-contract (B. Meyer) allows compile-time object-oriented interactions, we propose a notion of “composition contract” which allows run-time integration of services.

In general terms, a coordination contract is a connection that is established between a group of objects (i.e. participants). Through the contract, rules and constraints are superposed on the behavior of the participants, which determines a specific form of interaction. From a static point of view, a contract defines what in UML is known as an association class.
Our interpretation of contract to Sakai

The characteristics of contracts:

- A contract is a connection among objects (participants).
- A contract possesses rules and constraints on the behaviour of the participants.
- It is similar to association class in UML.
- A contract is a black-box.
- It is not accessible from other objects.
- It may be dynamically changed.
- It has been implemented with Java
Contract with context-aware behaviour


sensed and processed information, through e-learning system, to characterize a user or environment
Integration Model of Sakai with Contract
Using pattern to implement a coordination contract in Sakai (I)

The main requirements of the pattern can be divided into two categories:

1. General architecture requirements
   1.1 the ability to coordinate the behaviour of software components.
   1.2 minimising the number of required changes to the original components
   1.3 add and delete contracts in a “plug and play” mode.

2. “low-level” design requirements.
   2.1 Satisfy the semantics of contracts
   2.2 Optimise performance
Using pattern to implement a coordination contract in Sakai (II)
Use case to implement contract in Sakai (I)

Original Sakai Code

```java
import org.sakaiproject.discussion.api.DiscussionMessage;
import org.sakaiproject.discussion.api.DiscussionService;
public class DiscussionService extends BaseDiscussionService{
    public MessageEdit editMessage(MessageChannel channel, String id){
        return (MessageEdit) super.editResource(channel, id);
    }
}
```

- **Step 1**: Take a original Sakai code file #0
- **Step 2**: generation a file #1
- **Step 3**: generation a file #2
Use case to implement contract in Sakai (II)

1. Coordination contract and context-aware frameworks is imported

```java
package org.sakaiproject.discussion.impl;
    import java.util.*;
    import cde.runtime.*;
    import obab.ca.*; // Framework context-aware
public abstract class DiscussionService extends BaseMessageService implements DiscussionService, ContextObserver, EntityTransferrer, ForoInterfac
```
Use case to implement contract in Sakai (III)

2. Added method by the Tool SwC (Sakai with Contract) to the identification of classes which will be intercepted through the contract.

```java
public long _getNumber() {
    new ComponentOperationEvent(this, "getNumber").fireEvent();
    return number;
}

public messageEdit editMessage(MessageChannel channel, String id) {
    new ComponentOperationEvent(this, "Edit").fireEvent();
    return (MessageEdit) super.editResource(channel, id);
}
```
Use case to implement contract in Sakai (IV)

3. Method which implement the caller of client object to Proxy.

```java
protected CrdIProxy _proxy;
private static Class _classId= Sakai.Discussion.class;
public static Class getClassId() {return _classId;}
public CrdIProxy GetProxy() { return _proxy; }
public void SetProxy( Object p ) {
    if ( p instanceof CrdIProxy && p instanceof DiscussionInterface) _proxy = (CrdIProxy)p; }
public void SetProxy(CrdIProxy p) { _proxy = p; }
AccountInterface GetProxy_Account() {
    if ( _proxy == null ) return null;
    return (DiscussionInterface) _proxy.GetProxy(_classId);
}
```
1. Portion of code where is imported the components of the different framework, the abstract classes are inherited from the connectors and proxys. The class of real object "MethodServiceComponent" is represented through the subject attribute.

```java
package org.sakaiproject; import java.util.*;
  import cde.runtime.*;
  import obab.ca.*;

public abstract class IDiscussionPartner extends CrdContractPartner implements CrdIPProxy, DiscussiontInterface {
  protected Discussion subject;
}
```
Use case to implement contract in Sakai (VI)

2. Definition of the abstract method for the connection of the contract (implemented as “connectorMethodService” in the above pattern) with the services.

```java
public void SetProxy(Object p) {subject.SetProxy(p);} 
protected Object GetSubject_Object() { return subject; } 
public void ResetProxy() { subject.SetProxy(null); }
```
Use case to implement contract in Sakai (VII)

3. Method which permit the access to the methods that defines the services (e.i. methods belonging to MethodServicesComponent class)

protected Discussion GetSubjectDiscussion()  
{return (Discussion) subject;}

protected IDiscussionPartner getNextPartner_Discussion(){  
return (IDiscussionPartner)GetNextPartner(Discussion.GetClassId());}
4. Implementation for defect of methods defined in the interfaces of the services. Through the method "GetSubjectDiscussion()" which was detailed above, can be reached the methods which was made by de SwC tools belonging the first files.

```java
public void messageEdit (double amount,Customer c) throws DiscussionException {
    IDiscussionPartner next = GetNextPartner_Discussion()
    if (next != null) next.editMessage(amount,c);
    else GetSubjectDiscussion()._editMessage(amount,c);
}
```
5. Implementation of the conditionals of the contracts rules which are saved a XML files.

```java
public CrdPartnerRules messageEdit_rules(String text, Student c) throws DiscussionException, CrdExFailure {
    return new CrdPartnerRules (this);
}
```
Conclusion

The possibility of use coordination theory of contract as a way to implement some types of DHD's requirement used in:

- Design.
- Representation of DHD's relationship.

We invited you to see our next exposition called:

Towards a Dynamic Hypermedial Device (DHD)
Dra. Patricia San Martín – CIFASIS
Lic. Marina Burré - IUNA
Argentina