Integration of a Centralized and Highly Distributed Information System:  
A Case Study

Denis Kranjčec, Tanja Milićević  
University Computing Centre (Srce), Zagreb  
Marohničeva 5, HR-10000 Zagreb, Croatia  
denis.kranjcec@srce.hr, tanja.milicevic@srce.hr

Abstract. Integration of two systems is a complex procedure, and even so when one of the systems is highly distributed.  
With layering of applications, loosely coupling and using knowledge of eight fallacies of distributed computing we achieve a solution that is robust and stable.

Keywords. System integration, Distributed systems, Enterprise integration patterns, Loosely coupling.

1. Introduction

Information System for Higher Education (ISVU) [6] is an information system intended for institutions of higher education in the Republic of Croatia. It is financed by the Ministry of Science, Education and Sports (MSES). The implementation covers a complete evidence of syllabuses, the educational program fulfillment, course scheduling and all the students' activities, starting from the entrance examination, education and examinations to the graduation thesis and issuing of graduation certificate (diploma) and diploma supplement.

AAI@EduHr [1] was created as a combined project of Srce and CARNet (Croatian Academic and Research Network), financed by the MSES. AAI@EduHr is an authentication and authorization infrastructure of Science and Higher Education System in the Republic of Croatia. More on the project can be read on official internet page for the project.

AAI@EduHR is mandatory for all academic institutions whereas ISVU isn’t. The usage of both systems is free for every academic institution in the Republic of Croatia.

ISVU contains extensive information about students and employees. Electronic identity (EI) is one of that information. In order to have relevant information in EI in both ISVU and AAI@EduHR it was decided to integrate the two systems in a way that EI is created, updated and deleted by ISVU and propagated to AAI@EduHr.

ISVU is centralized system while AAI@EduHr is a highly distributed system with 217 directories; our goal in this paper is to give a case study of a possible solution to integration of those two systems.

It is important to emphasize that in this stage of integration the authentication of ISVU modules is not done by an EI from AAI@EduHr but with EI from ISVU directory (ActiveDirectory). Authentication of ISVU by EI from AAI@EduHR is an issue for some later project.

2. Background

ISVU is network oriented modular system for data processing and higher education institution’s (HEI’s) services and departments intercommunication. It consists of several modules, some of them are web modules and others are desktop modules. The system is based on relational database management system (DBMS). Although ISVU is centralized, every academic organization is owner of the data entered in the system. Through ISVU they can only work with data for their organization. All users, student included, use system on a daily basis.

AAI@EduHr system is realized in the usage of the distributed system of directories based on the Lightweight Directory Access Protocol (LDAP) technology. Every institution, that is a part of high education in the Republic of Croatia and is included into AAI@EduHr system, have their LDAP directory. That directory is stored at the institution and maintained by institution. The idea is that all data about students and employees in LDAP directories is maintained through ISVU and that no users ever use LDAP directory directly. When enrolling to the faculty the EI for
a student is automatically created and propagated to the LDAP directory of that institution.

Data exchange between these two systems is done in one direction, from ISVU to LDAP directory. Before integration, institution had accessed the directory through application for maintaining directory content (AOSI) and managed the EI. After the integration the EI is managed (created/deleted/updated) through ISVU.

AOSI is composed of two parts: the server part and the client part. AOSI server part is a web service (AOSI WS) whose purpose is to make access to directory easier through several procedures:
- Simple Object Access Protocol (SOAP) [10]
- Extensible Markup Language (XML) [2] is used for describing the data

AOSI client part is one of the possible clients and is intended for organizations that don’t have their own system for maintaining EI. Now, after the integration, directories are accessed through ISVU application. To ensure loosely coupling, the data that is only in directory is recorded through ISVU interface (GUI) through option of accessing the directory directly. This option is separated from ISVU system and in case of difficulties doesn’t affect the functionality of ISVU. The information is accessed using query-by-example (QBE).

We use Federation Web Service (FWS) for locating the (URI) AOSI WS.

Figure 1. ISVU and AAI@EduHr integration

The AOSI WS exposes the operations need for working with the directory. Most important operations are: ldapAddAttribute, ldapDeleteAttribute, ldapModifyAttribute, ldapAddUser, ldapDeleteUser.

A complete Web Services Description Language (WSDL) [9] of AOSI WS can be found on http://www.aaiedu.hr/aosi/aosi.wsdl

The important part of our integration is that the operations for adding, deleting and modifying attributes are separated in AOSI WS. Because of that it is necessary to insure the proper sequence for calling those operations during an asynchronous integration.

According to Peter Deutsch and James Gosling there are eight fallacies in designing a distributed system [8]:
1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn’t change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous

When working with distributed systems that are different in structure (different database, different programming language, operating systems, data structure), one needs a platform that can ensure good integration and communication between systems.

On the other hand, integration has its own set of problems. Systems that are integrated are usually developed on different platforms, technologies and/or standards. The amount and data diversity is also an issue in integration together with ensuring a data quality. The integrated systems were predefined. There was no possibility to change the systems and we had to adjust the integration to the systems.

There are several types of integration:
- Synchronization of information/data in real time (or in relatively short time period) also referred as synchronized exchange of data. It is characterized by small amount of data that is shared together with frequent sharing.
- filling of data warehouse
- system migration

During the synchronization the problem of data quality occurs. The best way of enabling system integration is to ensure that all of the distributed systems can work independently.

Low coupling indicates the stability of integration through a stable interface (contract). It indicates that ISVU should not be concerned with internal implementation of the
AAI@EduHr. Having loose coupling we ensure that our integration is stable.

Asynchronous integration is shown to be the best way of integration because it allows loosely coupled solutions [4, 5]. The main difference between synchronous and asynchronous integration is that in asynchronous model the system that sends the request to the other system does not need to wait for the response before sending another request. In that way the systems that are being synchronized can function independent of each other, ensuring that if there is a problem with one of the systems it doesn’t affect the functionality of the other system.

3. Implementation

Implementation includes both synchronous and asynchronous exchange of data. The EI consists of many attributes of which some are not needed in ISVU and to eliminate the need to access the directories through AOSI WS we use synchronous exchange. In synchronous exchange the administrator of LDAP directories can through ISVU module access the directory. This data is never recorded in ISVU. As mentioned before it is necessary to ensure that all of the actions done in ISVU can be carried out regardless of accessibility of LDAP directories. This can be done by asynchronous exchange.

There are considerable differences between the data in ISVU and the data in AAI@EduHr system. As the data in ISVU are recorded in the same place they are created it is expected that the data in ISVU is correct and complete. On the other hand the quality of the data in directories depends on how the institutions rely on their own information systems. For every institution that wants to integrate both ISVU and AAI@EduHr it is first necessary to pair the two systems. This means that every EI for student or employee in ISVU must be paired with EI from AAI@EduHr. That process can partially be done automatically but the final checkup is done manually. The pairing is done initially and is needed to ensure that the systems are synchronized and prepared for integration.

Asynchronous integration is implemented in the following manner: the module for integration periodically, every five minutes, gathers information about changes in data that are made in ISVU and need to be propagated to AAI@EduHr. The data gathered in that interval is then formed into a request for each directory and for each AOSI WS operation. Change of any attribute relevant to AAI@EduHr is recorded in a custom table. The recording of data change is automated using triggers on DBMS level. Table structure is simple and consist of chronological changes in ISVU as shown in Fig. 2

```
CREATE TABLE _transDBI
  (idtrans SERIAL NOT NULL,
   attributeID INTEGER NOT NULL,
   attributeName VARCHAR(255),
   newValue VARCHAR(255),
   transtype CHAR(1) NOT NULL,
   oid CHAR(50) NOT NULL,
   doc CHAR(50) NOT NULL,
   timeTrans DATE
  )
```

Figure 2. Table structure

Module for integration retains the ID of the last successful transaction. Using that ID forth the module gathers the information for next transaction.

One of the problems in working with distributed systems is in the stability of data transfer through network. The problem is emphasized when the data is transferred over the Internet, then over the intranet. The simple approach to integration would be that each change, deletion or adding an attribute results in single network call. This would drastically increase the number of network calls and would affect the reliability of integration and its performance.

The first optimization was made in a way that all of the changes made to one directory and particular operation (deletion, change or adding of EI) are consolidated into one call for entire interval. AOSI WS separately exposes each operation on directory so it is not possible to consolidate all calls to one directory. Simulation showed that using this kind of optimization during peak in ISVU (enrollment to faculty) a number of network calls was reduced 50 times.

The second step in reducing the number of AOSI WS operation and network calls is that for each attribute, for particular EI, only one operation is called, regardless of how many changes was on that attribute in one interval. One simple example of such optimization is entering a wrong passport number, 12345678 instead of 12345679. Integration would function correctly even when the two operation of attribute change where done (modifyAttribute). Our solution was that for each attribute, for every EI, the starting value and final value of attribute in one interval is recorded.

Based on those starting and final value the call of at most one operation on that attribute can be
determined. For example, attribute ATTRIBUTE has a starting value VALUE1, then that value is deleted (ATTRIBUTE now has value NULL), afterwards a new value VALUE2 is added. Now the final value of ATTRIBUTE is VALUE2. Instead of two operations, delete and add(VALUE2), it is sufficient to call only one operation: modify(VALUE2). Table 1 shows the possible combinations of starting and final states.

Table 1. Possible combinations

<table>
<thead>
<tr>
<th>Starting value</th>
<th>Final value</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>not needed</td>
</tr>
<tr>
<td>VALUE1</td>
<td>VALUE2</td>
<td>modify(VALUE2)</td>
</tr>
<tr>
<td>VALUE1</td>
<td>NULL</td>
<td>delete</td>
</tr>
<tr>
<td>NULL</td>
<td>VALUE1</td>
<td>add(VALUE1)</td>
</tr>
<tr>
<td>VALUE1</td>
<td>VALUE1</td>
<td>not needed</td>
</tr>
</tbody>
</table>

During monitoring of attribute change the domain was implemented using java classes. Those classes describe entities like Person, EI, and Institution. In those java classes we implemented dirty checking of changed attributes and records for starting and final values. This functionality is hidden from developers inside application programming interface (API). The standard get and set methods are exposed, and a dirty checking is implemented using classes that each of the domain inherits.

4. Integration issues

During integration process several problems were addressed, like the problem of data conversion from relations to objects to XML and back.

Typical problem in integration of information systems are the transformations from different models of data representation like relational objects, XML objects. During those transformations the problems known as object relational mismatch and object xml mismatch occur. In this case study the problem of object relational mismatch was solved by using the Data Access Object (DAO) layer, inside witch the conversion between relations and object is done. To access the database the proprietary library was used. The transformation between the relations in DBMS and java class in our domain was implemented in that way.

Java Architecture for XML Binding (JAXB) [7] was used for transformation between the object and XML. The starting point was XML because it matches the parameters that the AOSI WS expects. Using JAXB we got a generated java classes. Those generated java classes are not class from our domain so it was necessary to implement the transformation between the domain and JAXB classes.

Fig. 3 shows a Unified Modeling Language (UML) sequence diagram process. First, all of the changes from relations in DBMS are collected. Based on that information the changes on java classes from domain are made. The next step is generating the request for each specific operation for specific directory. This was implemented in a way that all of electronic identities for specific directory are gathered from domain and then the JAXB classes are generated for each operation in AOSI WS. From those JAXB classes using Marshall method we get the XML that is used as a parameter in SOAP. The result from AOSI WS is also an XML that is in an analog way transformed into domain java classes. The transformation to the domain java classes is needed for easier processing of data for API users. All of those transformations are hidden from the developer and he can see only the domain (entities) and services.

The important issue in integration is a recovery from problems that occur during integration. Problems can occur in network, system and applicative level. For example ISVU DBMS is not available, FWS service is not available, some of the directories are not available or change in some of the attributes wasn’t successful. It is recorded if change of attribute in ISVU, that needs to be propagated to AAI@EduHr, was successful or not. In that way it is possible to monitor how many requests were not successful and they can be resend. AOSI WS gives back the status for each electronic identity. In this case study, all such not successful requests are logged and information is send by e-mail to the administrators. The administrators can repeat the failed request. It is possible that after failed request for change the next request was successful. Then it is not needed to repeat the failed request, because the data can become inconsistent.

In implementation of domain java classes and helper class for dirty checking we used class inheritance. In that way every class from domain, for example Person class, inherits class called DirtyCheckHelper. This inheritance undermines separation-of-concerns principle in object programming. Advanced implementation could be based on Java Dynamic Proxy or AOP (Aspect Oriented Programming) solution.
Due to large differences in data model in object model and model used in LDAP directories the implementation of dirty checking was very demanding and it was needed to do a large number of testing and refactoring to achieve required functionality. A typical problem was in multi-value attributes, like hrEduPersonUniqueNumber, whose values can be found in properties of many classes in object model. Because of the way the AOSI WS is implemented, change of any value in a multi-value attributes results in a sending of all values. Other type of problem was that in LDAP directories exists attributes that are based on a several properties from object model. For example the attribute cn (full name as one attribute) is based on two properties from the object model, givenName (property for given name) and sn (property for surname). This means that change of any of the properties, givenName or sn, results in a change of the attribute cn.

During development we actively tested system integration by simulating production using four institutions in ISVU and four LDAP directories. In that testing period we solved majority of bugs. In pilot implementation on three institutions only one major bug occur that compromise the change of password. The reason was in different behavior of Java Virtual Machine (JVM) on Windows and Linux platforms. That problem was successfully solved.

After addressing all of the problems and system integration testing we were able to achieve a stable and robust solution. The problems that now occur are manly related to the temporary unavailability of individual LDAP directory.

5. Conclusion and Future work

In this stage of implementation three institutions are involved. The number of electronic identities is 17800. With layering of applications, loosely coupling and using knowledge of eight fallacies of distributed computing we were able to achieve a robust and stable solution to this particular integration problem of two systems in which one is highly distributed. The solution ensured that regardless of any change in ISVU or AAI@EduHr, integration is not affected. In designing the emphasis was that implementation is made as inconspicuous as possible for ISVU users.

This solution can be implemented in similar system. Because few layers are tightly connected to the domain, whereas ideas and design are not, better implementation could be used. This is regarded to the domain java classes, DAO layer and layer for transformation from JAXB java classes to domain java classes.

All requirements of this project where successfully implemented. In a next phase we plan to implement support for the binary attributes. We also plan to investigate a usage of messaging system such as Enterprise Service Bus (ESB). The automation of error recovery is also something we need to address in future work.

6. References

[1] AAI@EduHr , http://www.aaiedu.hr/ [01/12/2007]
Figure 3. The UML sequence diagram