USING JINI™ AND JAVASPACESTM TO DEVELOP AN
ADMISSION WORKFLOW SYSTEM WITH TRANSACTIONAL ABILITY

by

Pei Lin Lee

has been approved

September 2002

APPROVED:

Dr. Timothy E. Lindquist

Dr. Bruce R. Millard

Prof. Billy G. Wood

Supervisory Committee

ACCEPTED:

Dr. Timothy E. Lindquist
Department Chair
ABSTRACT

This project develops a distributed school admission system with transactional ability by using Jini™ and JavaSpaces™ technologies. The unstable network environment can easily cause distributed systems to crash in the middle of an operation and generate inconsistent results. Therefore, a transaction control is needed for coordinating remote clients and prevents the system getting into an inconsistent state. In this project, the program implementation is focused on providing the transaction control for graduate school admission workflow. The school admission system attempts to demonstrate the interactions and decision processes among the departmental recommendation, the graduate college’s final approval, and the method in which applying candidates accessing information. With the coordination and transaction tools provided by Sun Microsystems’ Jini™ and JavaSpaces™ technologies, the system can effectively generate consistent admission results for the graduate school admission workflow.
# TABLE OF CONTENTS

## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAPTER 1: INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Problem</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Scope</td>
<td>2</td>
</tr>
<tr>
<td>1.4 Assumptions</td>
<td>2</td>
</tr>
<tr>
<td>1.5 Approach</td>
<td>2</td>
</tr>
<tr>
<td>1.6 Computer Software Support</td>
<td>4</td>
</tr>
<tr>
<td>1.7 Sequence of Presentation</td>
<td>4</td>
</tr>
</tbody>
</table>

## CHAPTER 2: REVIEW OF LITERTURE

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Jini Technology Overview</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Basic Components of Jini</td>
<td>6</td>
</tr>
<tr>
<td>2.3 Five Key Elements of Jini</td>
<td>7</td>
</tr>
<tr>
<td>2.3.1 Discovery</td>
<td>7</td>
</tr>
<tr>
<td>2.3.2 Lookup Service</td>
<td>8</td>
</tr>
<tr>
<td>2.3.3 Leasing</td>
<td>10</td>
</tr>
<tr>
<td>2.3.4 Remote Events</td>
<td>11</td>
</tr>
<tr>
<td>2.3.5 Jini Transactions</td>
<td>11</td>
</tr>
<tr>
<td>2.4 JavaSpaces Overview</td>
<td>14</td>
</tr>
<tr>
<td>2.5 JavaSpaces Operations</td>
<td>16</td>
</tr>
<tr>
<td>2.5.1 Write Operation</td>
<td>16</td>
</tr>
<tr>
<td>2.5.2 Read Operation</td>
<td>16</td>
</tr>
<tr>
<td>2.5.3 Take Operation</td>
<td>17</td>
</tr>
<tr>
<td>2.5.4 Notify Operation</td>
<td>17</td>
</tr>
<tr>
<td>2.6 JavaSpaces and Transactions</td>
<td>18</td>
</tr>
</tbody>
</table>

## CHAPTER 3: METHODOLOGIES

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1 System Overview</td>
<td>21</td>
</tr>
<tr>
<td>3.2 System Architecture</td>
<td>22</td>
</tr>
<tr>
<td>3.3 System Classes</td>
<td>23</td>
</tr>
<tr>
<td>3.3.1 Server Side Classes</td>
<td>25</td>
</tr>
<tr>
<td>3.3.2 Client Side Classes</td>
<td>27</td>
</tr>
<tr>
<td>3.4 Handling System Transactions</td>
<td>28</td>
</tr>
</tbody>
</table>

## CHAPTER 4: SYSTEM RESULTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 Platform Requirements</td>
<td>31</td>
</tr>
<tr>
<td>4.2 System Batch Files</td>
<td>31</td>
</tr>
<tr>
<td>4.3 System Results</td>
<td>32</td>
</tr>
<tr>
<td>4.4 Analysis of Transaction Results</td>
<td>41</td>
</tr>
</tbody>
</table>
CHAPTER 5: CONCLUSIONS .................................................................43
  5.1 Discussion of Problems Encountered ..........................................43
  5.2 Conclusions ........................................................................44
  5.3 Future Work .......................................................................44

REFERENCES ........................................................................46
LIST OF FIGURES

Figure 1.1 Operations under the admission transaction ...........................................3
Figure 2.1 Jini Networking System ...........................................................................5
Figure 2.2 Basic Components of Jini System ...............................................................7
Figure 2.3 Discovery ......................................................................................................7
Figure 2.4 The Flow Diagram of Jini Lookup .............................................................10
Figure 2.5 Remote Events ..........................................................................................11
Figure 2.6 Prepare Phase ............................................................................................13
Figure 2.7 Commit Phase ...........................................................................................14
Figure 2.8 JavaSpaces ...............................................................................................15
Figure 2.9 Write Operations and Transactions ........................................................18
Figure 2.10 Read Operations and Transactions .......................................................19
Figure 2.11 Take Operations and Transactions .........................................................20
Figure 3.1 Major Components of the School Admission System .............................23
Figure 3.2 System Class Diagram .............................................................................24
Figure 3.3 Space operations involve in the admission transaction ............................29
Figure 4.1 Student Login Prompt .............................................................................33
Figure 4.2 Student User Interface ............................................................................34
Figure 4.3 Set Student Data ......................................................................................35
Figure 4.4 View Admission Status and Result ..........................................................36
Figure 4.5 Department Login Prompt ......................................................................37
Figure 4.6 Department User Interface .....................................................................37
Figure 4.7 Get the List of Applying Candidates ........................................................38
Figure 4.8 View the Candidate’s Information ..................................................38
Figure 4.9 Set the Recommendation .................................................................39
Figure 4.10 Graduate College User Interface ....................................................39
Figure 4.11 Set Final Approval ................................................................. 40
Figure 4.12 Announce Admission .................................................................40
Figure 4.13 Before the transactions commits .................................................41
Figure 4.14 After the transactions commits ....................................................42
CHAPTER 1 : INTRODUCTION

1.1 Background

Transaction mechanism is a way of grouping a set of operations together and ensures that either all operations complete successfully, or none complete at all [4]. A transaction mechanism is especially needed in a distributed system because remote clients can crash before an operation has completed. This uncompleted operation can lead to inconsistencies and cause the system to operate inaccurately. For example, to transfer money from savings to checking accounts, the operation of subtracting money from the savings account and the operation of adding money to the checking account must be completed together, otherwise the transfer should be cancelled and revert to the original or previous state. Hence, a properly designed transaction mechanism can prevent the case where the money is subtracted from the savings account but never added to the checking account. Sun Microsystems, Inc. provides a Jini service called JavaSpaces that fully supports transaction mechanism [8]. The main purpose of this project is to use Jini™ and JavaSpaces™ to develop a distributed system with transactional ability to process the graduate school admission workflow.

1.2 Problem

The unstable network environment can often affect the performance of distributed systems and generate error data or inconsistent results. The system must employ some kind of transaction control against unpredictable network failures to ensure the system always produces reliable results. The goal of this project is to use Arizona State University graduate school scenario to build a transactional secure admission workflow.
system. The system should always generate consistent results from unstable network environment.

1.3 Scope

The implementation of this distributed admission workflow system is focused on providing transaction control by using Java programming language with Jini and JavaSpaces. The introduction of Jini Technology and JavaSpaces service is also covered in this project. However, other approaches such as using Oracle transactional database system or C programming language to implement the transaction mechanism for the distributed system are not covered in this project.

1.4 Assumptions

Granting graduate school admissions for the applying candidates requires going through a set of operations. These operations include receiving the recommendation from the department and the final approval by the graduate college before announcing the admission results. All other procedures such as submitting official transcripts, paying application fees, and providing immunization records should be done before entering to this admission evaluating system.

1.5 Approach

The major steps of developing the school admission system are:

1. Designing client side applications with graphical user interface for data input.

2. Developing server side applications to handle the graduate school admission workflow.
3. Managing operations that are involved in the admission transaction. These operations include the department reading in the student object and taking out the student admission (studentAdm) object from the JavaSpace before granting the recommendation. This granting is performed by writing a Yes/No decision at the recommendation attribute in the studentAdm object. Next, the graduate college should take out the student and studentAdm objects from the JavaSpace and grants the final Yes/No decision at the approval attribute with admission date and admitted program to approve student’s admission. In addition, the graduate college publishes each student admission result by pressing the announce admission button on system user interface and then trigger the system server to complete the admission transaction. All of the above operations should be completed under one transaction. Any operation failure will cause the transaction to abort. When the transaction aborts, the student and studentAdm objects should all be rolled back to the original state and no modifications should be applied to the data objects. Figure1.1 depicts these operations graphically.

![Figure 1.1 Operations under the admission transaction](image_url)
4. Final Test and debug the program to complete the admission workflow system.

5. Develop the result and conclusion for the project report.

1.6 Computer Software Support

The computer software for developing the school admission system include:

- Windows 2000 professional
- Jini Technology Starter Kit, Version 1.2 (Jini1_2)

1.7 Sequence of Presentation

This project’s background, scope, and approach are covered in Chapter 1. Chapter 2 reviews the current literature that discusses the Jini technology, Jini Transaction mechanism, and the JavaSpaces service. The methodology of developing the school admission workflow system is presented in Chapter 3. Chapter 4 demonstrates the result of the admission workflow system. The project conclusions and future work are discussed in Chapter 5.
CHAPTER 2: REVIEW OF LITERATURE

Two essential technologies used in this project are Jini™ and JavaSpaces™. The literature review on these two technologies is covered in the following sections.

2.1 Jini Technology Overview

Sun Microsystems introduced the Jini™ Technology in 1999 [4]. Jini provides the Java-based networking system that enables all kinds of hardware or software components to link together and form a services sharing community on the network [17]. In addition, Jini offers the simple “plug and play” model that allows services and devices to join and detach from the network with effortless configuration [2]. For instance, when a Jini-enabled Personal Digital Assistant (PDA) enters the Jini network, the PDA can automatically find the nearby printer services or disk storage services and use them without complicated driver installation or physical connection. The vision of Jini is to provide wide range of services on the network from computer peripherals to home appliances and enable service users dynamically downloading services from the network without knowing the service’s location and implementation [4]. Therefore, with the Jini technology, people can remotely control home appliances such as air conditioners or garden sprinkler systems from their Jini-enabled PDA. Figure 2.1 illustrates the overview of Jini networking system.
2.2 Basic Components of Jini

The basic components of Jini system are:

- **Service** is an entity that provides needed functionality and be accessed remotely across the network by a person, a device or another service [9]. Examples of services include printers, digital cameras, applications, and so on.

- **Client** is an entity who requests the use of services [4]. Any Jini service can also be the client of other Jini services.

- **Lookup service** provides a central registry for service providers. Remote clients use lookup services to locate and download their desired services [4]. The Sun’s implementation of Jini lookup services is called Reggie and it is located in the lib/reggie.jar and lib/reggie-dl.jar files from Sun’s Jini Technology Starter Kit [8].

- **Service Proxy** is a Java object that includes all the interfaces provided by the service provider. The service provider uses it to do Jini registration by publishing its proxy to the lookup service. The client can dynamically download this proxy from the network and use it to interact with the service provider [4]. A service proxy can be implemented in three ways:
  1. **Local Only Proxy**: this proxy implements all the functionality of the service and resides within the client’s machine or downloaded through the network. Once it successfully downloaded, this proxy object is executed entirely within the client’s Java Virtual Machine without further network connection [9].
  2. **RMI Stub as a Proxy**: this proxy is a Java remote object that has minimal code to communicate with the remote service. The network connection is required, when the client uses this kind of proxy to interact with the service provider [9].
3. **Smart Proxy**: this proxy is the combination of the first two proxies. It consists a set of local methods and one or more remote object references [9].

Figure 2.2 displays each component graphically.

![Figure 2.2: Basic Components of Jini System](Adapted from 4)

### 2.3 Five Key Elements of Jini

Jini technology is based on couple of key elements in order to provide a robust, self-healing, and reliable networking system. The five key elements are discovery, lookup, leasing, events, and transactions that are discussed one by one at the subsequent sections.

#### 2.3.1 Discovery

Discovery is a protocol that enables services and clients to find lookup services on the network [1]. See Figure 2.3.

![Figure 2.3: Discovery](Adapted from 13)
Jini Discovery is like a boot up process for services and clients. Once services and clients located the lookup services, the service can register itself in the network and the client can search and download its desired services. Jini supports three kinds of discovery protocols for different situations.

- **The Unicast Discovery Protocol** is the direct lookup; it is used when the client or the service provider already known the IP address and the port number of a particular Jini lookup service [11].

- **The Multicast Discovery Protocol** is used when the client or the service provider does not know the locations of Jini lookup services and uses the multicast discovery protocol to search all nearby lookup services [4]. In the multicast discovery, the service or the client sends out a multicast request and all Jini lookup services that see the request will respond to the requester [11].

- **The Multicast Announcement Protocol** is used by Jini lookup services to announce their existence [4]. When a new lookup service starts up on the network, any interested parties will be informed via the Multicast Announcement Protocol [4].

**2.3.2 Lookup Service**

The most fundamental part of Jini is the lookup service. The lookup service is like a service center that keeps tracking all available services on the network and provides service matching process between clients and services. When a service wishes to make itself available on the network, it registers with the lookup service by publishing its service proxy [15]. Then clients can connect to the lookup service and use the searching criteria to locate its desired services [3]. Jini supports a number of ways to search for
services, especially searched by content process that is the most difference from the traditional naming servers [9]. The ways to search for potential services are:

- **Service Types**: Clients can use array of proxy types such as interface it supports or service’s super classes and super interfaces to locate the services [8].

- **Service Attributes**: Clients can search services based on the descriptive attributes that associated with the service’s proxy [4]. The typical attributes include service’s vendor name, location, and comments. In the case of a Jini-enabled building that has multiple printers, clients can use the location attributes such as “first floor” to retrieve the printer service that is located at the first floor of the building.

- **Service ID**: Every service has a unique ID in the Jini network [8]. Clients can use this number to identify the service from the network.

Figure 2.4 on the next page reflects the basic flow of Jini lookup mechanism. The sequence of operations is:

1. **Discovery**: A network service discovers available Jini lookup services on the network.
2. **Join**: A network service joins the Jini network by sending its service proxy to the lookup service.
3. **Discovery**: A network client discovers available Jini lookup services on the network.
4. **Lookup**: The network client sends a request to the lookup service to find desired services.
5. **Receive**: Jini lookup service sends registered service proxy to the network client.
6. **Use:** The network client uses service proxy to interact directly with the network service [15].

![Flow Diagram of Jini Lookup](image)

**Figure 2.4:** The Flow Diagram of Jini Lookup [Adapted from 15]

### 2.3.3 Leasing

Leasing is the method to grant a use of services under a limited timeframe [9] that enables Jini network to be robust in the face of network failures [11]. All Jini federations including clients, services and lookup services are participated in the lease model [9]. When a service registers with the lookup service, it receives a lease object from the lookup service. The service must periodically renew the lease with the lookup service in order to maintain its presence on the network. Once the service fails to renew its lease due to network failures, the lookup service automatically removes that expired service from the Jini network [8]. With the leasing mechanism, the Jini system can be more efficient by quickly detect unreachable services or clients and remove potential garbage without administrative intervention.
2.3.4 Remote Events

Jini system supports distributed events, which allow sending notifications to the registered clients when their interested services join the Jini network [14]. The clients can register interest in receiving an event with the lookup service and proceed with other tasks instead of waiting for desired services to appear. Figure 2.5 describes the client, which can be notified when its desired service joins the Jini network.

![Remote Events Diagram]

Figure 2.5: Remote Events

2.3.5 Jini Transactions

Transaction mechanism is a way to group a set of operations becoming a single indivisible unit that means either all of the operations complete, or none of them do [5]. Transaction mechanism is designed to provide ACID properties. The ACID properties are:

- **Atomicity:** All the operations of a transaction occur or none of them do [10,16].
- **Consistency:** The completion of a transaction must leave the system in a consistent state that means transforming the system from one consistent state to another consistent state [10,16].
• **Isolation:** The activities of one transaction should not be affected by any other transactions [10,16].

• **Durability:** The results of a transaction should be persistent [10,16].

Jini transaction is one of the less frequently used features of Jini, however, this mechanism provides a powerful tool for distributed systems to operate correctly in the presence of partial failure [4]. Jini transaction mechanism takes a different approach from traditional transaction systems, which enables the clients and services to implement various level of ACID properties to reach their desired [9,16]. The main parties involved in Jini transactions are:

• **The transaction client:** Transaction client is an object that initiates and terminates the transaction [11]. The client starts the process from obtaining a transaction object from a transaction manager, and then passes that transaction object as a parameter to a series of method that the client wants treated as an atomic unit. At the end, the client can invoke the commit or abort method to complete a transaction [16].

• **The transaction manager:** Transaction manager is an object that oversees the entire transaction [11]. Transaction manager provides transaction objects to transaction clients and coordinates the activities of all transaction participants [16]. Sun’s implementation of the transaction manager is called *Mahalo* [8] that is ready to be used and located in the *lib/mahalo.jar* and *lib/mahalo-dl.jar* files from Sun’s Jini Technology Starter Kit.

• **The transaction participants:** Transaction participants are objects that execute operations of a transaction and able to interact with the transaction manager to complete transactions properly [16].
The practice of Jini transactions is employed the two-phase commit protocol. These two phases are:

- **Phase 1- Prepare:** The first phase is called prepare [4]. This is the voting process that a transaction manager asks all the transaction participants to vote if they are ready to perform their operations. During this moment, all the participants execute their operations and record their results into some temporary storage such as a log file. Then, each participant should report its vote to the transaction manager about whether it is ready to move forward to the next phase. Once the transaction manager gets all the votes from the participants, the phase one is completed and the process moves into phase two [4]. This phase is shown in Figure 2.6.

![Figure 2.6: Prepare Phase [Adapted from 4]](image-url)

- **Phase 2- Commit:** Phase two is the closing up stage of the transaction. The result of this phase is either all participated operations are performed or all aborted. During the phase two, the transaction manager calculates the votes from all the participants.
If all the participants vote for the ready as shown in Figure 2.6, then the transaction manager informs each participant to commit their operations like Figure 2.7.

![Figure 2.7: Commit Phase [Adapted from 4]](image)

This commit action means each participant copies the results from the temporary storage into the permanent storage, thus, the transaction client sees all operations that wrapped in a transaction to occur simultaneously [4]. However, if any of the participants vote for not ready during the phase one, then the transaction manager should notify all the participants to abort their actions and rollback to their initial state [4,8,16].

### 2.4 JavaSpaces Overview

JavaSpaces technology is a Jini service that can be located by using Jini lookup services. The Sun’s implementation of JavaSpace services is called *Outrigger* [9] and placed in the lib/outrigger.jar and lib/outrigger-dl.jar files from Sun’s Jini Technology Starter Kit. JavaSpaces service provides a reliable virtual storage system for distributed users to communicate, coordinate, and share objects across a network [7]. The overview of JavaSpaces service is shown in Figure2.8.
The key features of JavaSpaces are:

- **Shared**: Distributed servers and clients can access a JavaSpace concurrently for exchanging tasks, requests and information. The concurrent control already handled by the JavaSpace service [7].

- **Persistent**: The objects stored in a JavaSpace can be persistent. This persistent indicates that a collection of data remains intact even its source is disconnected from the network [12].

- **Associative**: JavaSpace service takes the advantage of Jini technology and enables users to retrieve stored objects according to their content [13].

- **Transactional secure**: All JavaSpaces operations are fully integrated with the Jini transaction mechanism [8].

The objects that stored in the JavaSpaces are Entry objects and these Entry objects can serve as templates to match and retrieve the stored objects. The templates usually have some or all of its fields set to specified values that must be matched exactly and the unset
fields are left as wildcards [13]. When an object implemented as the Entry type, the following rules must be applied [7].

- Each field in the Entry object must be declared as public.
- Fields cannot be primitive types. They must be objects.
- Fields need to be serializable.
- A public and no argument constructor must be provided.

2.5 JavaSpaces Operations

JavaSpaces interface supports four primitive operations: Write, Read, Take, and Notify. The following sections take closer look at these JavaSpaces operations.

2.5.1 Write Operation

The write operation places a copy of an entry object into a JavaSpace [13]. The syntax of the write method is:

```java
public Lease write( Entry entry, Transaction txn, long lease)
        throws TransactionException, RemoteException
```

This method takes three parameters: the Entry object that is to be written, the transaction object for this operation, and a lease time for the Entry object to be stored. The return value of the write method is a lease object that enables users to manage the written entry’s storing time [7]. Each write operation places a new copy of the Entry object into the JavaSpace, even if the same Entry object that is already existed in the space [13].

2.5.2 Read Operation

The read operation reads an Entry object that matches the given template from the JavaSpace [13]. There are two kinds of read operations: read and readIfExists. Their syntax are:
Both of these methods take three parameters: the Entry object that serve as a template, the transaction object for this operation, and a timeout interval used to wait for a matching entry to appear in the JavaSpace. The return value of the read method is the matching Entry object. If none is found within the specified timeout, then a null value will be returned [11]. The only difference between read and readIfExists is their treatment of the timeout value [7]. The timeout value of the readIfExists method is used for the only matching entry that is under a non-finished transaction and this timeout value specifies how long the client is willing to wait for the transaction to settle before returning a value [13].

2.5.3 Take Operation

The take operation is similar to the read operation and the only difference is the take operation is not only reads the matching entry, but also removes the matching entry from the JavaSpace [7]. Except this removing ability, the syntax and the difference between take and takeIfExists are exactly the same as the read operation.

2.5.4 Notify Operation

The notify operation provides “an asynchronous mechanism for being informed when interesting entries are written into a space” [7]. Hence, when the matching entries are written into the JavaSpace, the specified Remote Event Listener will eventually be notified. The syntax of the notify operation is:

```java
public EventRegistration notify( Entry tmpl, Transaction txn,
                               RemoteEventListener listener, long lease, MarshalledObject handback )
throws TransactionException, RemoteException
```
This method takes five parameters: the Entry object that serves as the template, the transaction object for this operation, a remote event listener that will receive the remote events, a lease time for the registered listener, and a marshaled object that will sent to the remote event listener as part of the event data [11].

2.6 JavaSpaces and Transactions

All of the JavaSpaces operations are fully integrated with Jini transaction mechanism [8]. The effects of operations under a transaction are:

- **Write Operation:** When a transaction object is passed for the transaction parameter, the Entry object is written under a transaction. This Entry object is not seen or accessible outside of the transaction until the transaction commits [13]. If the transaction is aborted, the Entry object will not be written into the JavaSpace. For example, Figure 2.9 shows that when the *write* operation is carried with the transaction objectA as the parameter, the Entry is written under the TransactionA block. All the Entry objects that are written within TransactionA block will not be visible to outside of the TransactionA until the TransactionA commits. When the write operation is not carried the transaction object and use null value as the parameter, the Entry object is directly written into the JavaSpace and able to be seen in the entire space.

![Figure 2.9: Write Operations and Transactions](image)

Figure 2.9: Write Operations and Transactions
• **Read Operation:** If a transaction object is passed for the transaction parameter, the matching Entry, which is written under that transaction, can be read [13]. Without the transaction object passed as the parameter, the *read* operation is not able to read the Entry that is written under the transaction because that Entry is invisible for outsiders of the transaction. For example, Figure 2.10 shows that when the *read* operation carries the transaction objectA, this *read* operation is able to enter the TransactionA block and reads the matching Entry object. Without the transaction object, the read operation is only able to read the Entry objects that are not written under the transaction.

![Figure 2.10: Read Operations and Transactions](image)

• **Take Operation:** The *take* operation has the same effect as the *read* operation, when a transaction process involved. If a transaction object is provided with the *take* operation, the matching Entry, which is written under that transaction, can be taken from the JavaSpace [13]. Without the transaction object, the *take* operation stays
outside of the transaction and is not able to see and remove the Entry object that is written under the transaction. See Figure 2.11.

![Figure 2.11: Take Operations and Transactions](image)

- **Notify Operation:** If a transaction object provided with the *notify* operation, the matching Entry objects which is written under that transaction, can be seen and the notification will perform. Otherwise, the Entry objects that are written under the transaction are still invisible; the notification will not proceed [13].

To summarize, Jini technology provides the robust and reliable network environment for distributes systems and JavaSpaces service offers the persistent virtual storage system with transactional ability. With these two technologies, this project’s graduate school admission workflow system is easier to build, and the system results are more reliable. The methodology of developing the school admission system is discussed in the next chapter, and the system results are shown in chapter 4.
CHAPTER 3: METHODOLOGIES

This applied project develops the graduate school admission workflow system to experience the transaction mechanism with Jini™ and JavaSpaces™ technologies. The system overview, system architecture, system classes, and the system transactions handling are discussed in this chapter.

3.1 System Overview

The school admission system’s architecture and basic implementation is based on Dr. Lindquist’s CET427/598 class example, workflow role manager system [6,9] and adds the necessary modifications to fit the school admission workflow. The goal of this admission system is providing transaction control to assure the consistent admission results. This distributed system not only automates the admission workflow that increase the admission processing speed but also prevents the inconsistent state which the student gets admission approval from the graduate school but is denied by the department. The three primary users of the system are:

- **Student**: Student users can use their student ID to login the system. After the login, the system will display a graphical user interface (GUI) for students to input their application information. After students finish inputting the data, they can use the GUI buttons to set or get their written data. Students can check their admission status and admission results by pressing the View Result button.

- **Department**: Department users can use the department ID to login the system. After login, the system will display the GUI with the Get Applications, Get Applicant Content, Set Recommendation and Done buttons. Department can use these buttons
to retrieve the list of applying candidates, to view the applying candidate’s information, and sets the departmental recommendations.

- **Graduate College**: Graduate College users can use school ID to login the system. After login, the system will display the GUI with five buttons. The Get Applications button is for Graduate College to get the list of applying candidates. The Get App Content button is used to retrieve the current selected candidate’s application information. The Set Approval button is for Graduate College to set the final admission approval. Graduate College can use the Announce Admission button to publish the student’s admission results and completes the admission transactions.

3.2 System Architecture

The school admission system uses Jini technology to handle network connections. The system data are stored in the JavaSpace and employ Jini Transaction Manager to process transaction control. There are six major components of the system:

- **Server Application**: Server application is responsible for publishing its service proxy to the Jini lookup service and making it available on the Jini network. It also manages the creation of student objects and transaction objects that associated with the admission workflow.

- **Jini Lookup Service**: Lookup service serves as the connection bridge between the server and the clients on the network. The implementation of Jini lookup services is provided with Sun’s Jini Technology Starter Kit.

- **JavaSpace**: JavaSpace is the system data storage. The student and student admission objects are stored in this place.
• **Three Client Side Applications**: The three clients are: Student, Department, and Graduate College. All the client side applications provide friendly user interfaces for easy data input and output. Jini lookup discovery is also handled by the applications in order to download the school admission service proxy from the network. Once the client applications have the service proxy, they can use the proxy to communicate with the server and manipulate data that are stored in JavaSpace.

Figure 3.1 illustrates each component graphically.

![Figure 3.1: Major Components of the School Admission System](image)

3.3 System Classes

The graduate school admission workflow system consists of twelve Java classes.

Figure 3.2 on the next page displays the whole system class diagram.
Figure 3.2: System Class Diagram
The following sections describe each class in detail by grouping them into server and client side classes.

### 3.3.1 Server Side Classes

The server side classes are: gradServerAdmin, gradServerImpl, gradServerProxy, and gradServerEntry. The studentEntry and the studentAdmEntry that are commonly used by the server and clients are also discussed in this section.

- **gradServerAdmin.class:** This class registers the server class (gradServerImpl) with the RMI Daemon as activatable service. With the activatable service, the server can run it on demand instead of running it all the time.

- **gradServerImpl.class:** This is the service implementation class. This class is responsible for registering the service proxy with Jini lookup service, maintaining leases, initializing JavaSpace and JavaSpaceAdmin variables. It also manages the creation of student workflow objects and posts these objects into JavaSpace. The transaction handling process is also taken care by this class that includes looking for a Transaction Manager from Jini Lookup service, creating transaction objects for each student and providing a method to pass transaction objects for clients.

- **gradServerProxy.class:** This is the service proxy class that serves as a pass-through for server operations. The gradServerImpl class sends this proxy class to Jini lookup service for registrations on Jini. Remote Clients download this proxy from Jini lookup service and use it to communicate with the server and manipulate student data.

- **gradServerEntry.class:** This is the service attributes class. Remote clients can use this class to search the admission services based on specific attributes. When multiple service providers use the common name to register with Jini lookup service,
this class can provide more specific searching criteria for clients to find the particular service. For example, if there are multiple ProcessAdmissions services available on the Jini network, clients can use unique vendor name such as ASU to retrieve the ProcessAdmission service from Arizona State University.

- **studentEntry.class:** This is the student data class that contains the student information along with application information and the final admission result. This class is stored in JavaSpace and manipulated by students, department and graduate college users. As being the Entry type, this class implements the Entry marker interface, supply a no-argument public constructor, and provides three main variables for attribute-based searching. The three main variables are: Vendor, Department, and ID. Clients can set specific values for all or some variables to retrieve the exactly matched student objects. For example, when the template Entry is `studentEntry("ASU", "ECET", "701223333")`, it will retrieve one student object that is applying ASU graduate school’s ECET department and its student ID is 701223333. When the template Entry is set for `studentEntry("ASU", "ECET", null)`, it will retrieve all the student objects that are applying ECET department of Arizona State University.

- **studentAdmEntry.class:** This is the student admission data class that contains the information of departmental recommendation, graduate college’s final approval, and the student admission result. This data class is also stored in JavaSpace and manipulated by department and graduate college users. This class has the same searching criteria as the studentEntry.class that uses vendor name, department name and student ID to retrieve specific student admission object.
3.3.2 Client Side Classes

The client side classes include studentGui, studentManager, departmentGui, departmentManager, graduateGui, and graduateManager classes.

- **studentGui.class**: This class contains the Java Swing components that make up a friendly user interface for students to manipulate studentEntry data and view the admission results.

- **studentManager.class**: This class extends studentGui class and handles the events from the studentGui Swing components. This class also processes the operations of discovering Jini lookup service, downloading admission service proxy, and provides access to studentEntry data through Jini and JavaSpace.

- **departmentGui.class**: This class contains the Java Swing components that make up a friendly user interface for department users to view applying candidates’ information and set departmental recommendations.

- **departmentManager.class**: This class extends departmentGui class and handles the events from the departmentGui Swing components. It also provides the functions of discovering Jini lookup service, downloading admission service proxy, and providing access to studentEntry and studentAdmEntry objects through Jini and JavaSpace. In addition, this class also handles the operations that are under the transaction. The details of handling transactions are discussed in section 3.4.

- **graduateGui.class**: This class also contains the Java Swing components that make up a friendly user interface for graduate college users to view applying candidates’ information, set the graduate college’s final admission decision, and publish student admission results.
• **graduateManager.class:** This class has the same functionality as departmentManager.class; the only difference is the user. In here, the user is the graduate college and the graduate college is the only client that is able to trigger the server to commits the admission transactions.

3.4 Handling System Transactions

Three key steps should be followed in order to use transactions with JavaSpace operations. The steps are:

1. **Locate a Transaction Manager:** Firstly, system developers should locate a transaction manager that can create and maintain transaction objects for the system. The way to locate a transaction manager is the same as looking for a service from Jini Lookup. Here is the code that gradServerImpl class uses to obtain a transaction manager proxy from Jini lookup service:

   ```java
   private TransactionManager findTransactionMgr()
   {
       TransactionManager tranMgr = null;
       try{
           Entry[] attrs = new Entry[1];
           attrs[0] = new Name("TransactionManager");
           ServiceTemplate tmpl = new ServiceTemplate(null,null,attrs);
           tranMgr =(TransactionManager)serviceRegistrar.lookup(tmpl);
           System.out.println("Found TransactionManager: "+tranMgr);
       }catch (Exception ex)
       {
           System.out.println("Exception in findTransactionMgr(): "+
                               + ex.getMessage());
       }
       return tranMgr;
   }
   
   2. **Create a Transaction Object:** After the system can access a transaction manager, we can create a transaction object by calling the create method of the TransactionFactory class. Here is the code that gradServerImpl class uses to create transaction objects:
private void createTran( String stuID) {
    Transaction txn = null;
    try{
        if(studentTrans.containsKey(stuID) )
            System.out.println("Already has Tran obj for stuID: "+stuID);
        else
            {  
                Transaction.Created trans = TransactionFactory.create(
                    tranMgr, Lease.FOREVER);
                txn = trans.transaction;
                System.out.println("create Tran obj: "+txn +
                    " for stuID:"+stuID);
                studentTrans.put(stuID,txn ); //put txn obj in hashtable
            }
    }catch( Exception ex)
        {System.out.println("Exception in createTran(): ": "
            +ex.getMessage());}
}

Since each student requires a transaction object to ensure its admission result, the admission system server must create multiple transaction objects to process the transaction control. In the above code, the gradServerImpl creates the studentTrans Hashtable variable to manage transaction objects for each student.

3. **Pass the transaction object for JavaSpace operations:** After transaction objects are created successfully, we can pass the transaction object to each JavaSpace operation that need to occur under the transaction. The operations that are involved in the admission transaction are shown in Figure 3.3.

![Figure 3.3: Space operations involve in the admission transaction](image-url)
According to Figure 3.3, both of the distributed clients, department and graduate college, need to perform some space operations that are under the admission transaction. In order for distributed clients to use a common transaction object for each space operation, the system assigns gradServerImpl class to create the transaction object and provide the getTransaction method to deliver the transaction object for remote clients. Here is the code that describes the department client to request a transaction object from the server and pass that object to the space operations:

```java
gradServerProxy wfsp = (gradServerProxy)vendors.get("ASU");
tran = wfsp.getTransaction( currID); //get txn obj from server
studentAdmEntry templA = new studentAdmEntry(vendor,depart,currID);
studentAdmEntry admR=(studentAdmEntry)
    space.exists(templA,tran,5000);
admR.setRecomm( recomm );
space.write(admR, tran, Lease.FOREVER);
```

The next chapter shows the steps to run this admission workflow system and presents system results. The final conclusions are covered in Chapter 5.
CHAPTER 4 : SYSTEM RESULTS

This chapter covers the school admission system’s platform requirements and shows the system running procedure. At the end, the system results are presented in sequential screenshots.

4.1 Platform Requirements

Before running the school admission system, it requires some software to install in advance. These requirements are:

  
  JDK1.4 is available to download from: http://java.sun.com/j2se.

- Jini Technology Starter Kit, Version 1.2 (Jini1_2).
  
  Jini1_2 is available to download from: http://java.sun.com/jini.

- Windows 2000 professional operating system.

4.2 System Batch Files

The system comes with seven batch files, which enable users to run the system easily. These batch files are based on Dr. Lindquist’s example [9]. The description of these files are listed below:

- setEnv.bat: This file sets environment variables to ensure Java and Jini classes are accessible. The following variables should be adjusted according to the user’s computer:
  
  ```
  set CODEBASE=d:\jinil_2\lib
  set JINIHOME_BACKSLASH=d:\jinil_2
  set JAVA_HOME=c:\jdk1.4
  ```

- make.bat: This make file compiles the system source code.
• **jiniServices.bat**: This file starts the six necessary services: one HTTP server, a RMI Registry, a RMI Daemon, a Jini Lookup service (Reggie), a Transaction Manager service (Mahalo), and a JavaSpace service (Outrigger). Four parameters should be given when invoking this file. The invoking syntax is:

```bash
jiniServices httpPortNo rmiRegPortNo rmidPortNo parentDir
```

For example:
```
jiniServices 8081 2575 9999 c:\gradAdm
```

• **runServer.bat**: This file starts the Server Application and takes the same parameters as the jiniServices.bat. The sample invocation is:

```bash
runServer 8081 2575 9999 c:\gradAdm
```

• **runStudent.bat**: This file starts the Student Application and takes two parameters. The syntax is:

```bash
runStudent httpPortNo DiscoveryOrLocalhost
```

For example:
```
runStudent 8081 localhost
```

• **runDepart.bat**: This file starts the Department Application and takes two parameters. The syntax is:

```bash
runDepart httpPortNo DiscoveryOrLocalhost
```

For example:
```
runDepart 8081 localhost
```

• **runGrad.bat**: This file starts the Graduate College Application and takes two parameters. The syntax is:

```bash
runGrad httpPortNo DiscoveryOrLocalhost
```

For example:
```
runGrad 8081 localhost
```

4.3 System Results

Here are the steps to run the school admission system and its screenshot results:

• Start a DOS command prompt and extract the gradAdm.jar file.
• The gradAdm directory will be created after the extracting and change the current directory to gradAdm.

• Compile all system source code by invoking **make.bat** file.

• Start all Jini services by invoking **jiniService.bat** file. After the invoking, the computer screen should display six windows for each service and two of the six windows will disappear after a minute. Please do not proceed following steps until these two windows have been disappeared. Otherwise, the server and the client applications are not able to run.

• After all Jini services are running successfully, users can start the system server by invoking **runServer.bat** file.

• When the system server is running, a user can start the student application by invoking **runStudent.bat** file. Then a login prompt will display to request student ID as shown in Figure 4.1.

![Figure 4.1: Student Login Prompt](image)

After inputting the student ID and pressing the OK button, the student user interface will appear on the screen as Figure 4.2 on the next page.
Figure 4.2: Student User Interface

The student user interface enables students to input information and view the admission results. The “Set Data” button is for students to write student data into JavaSpace. Figure 4.3 on the next page shows the system reaction after pressing the “Set Data” button.
Figure 4.3: Set Student Data

The “Get Data” button is for students to retrieve their information from the JavaSpace by using the student ID. The “View Result” button is for students to check their admission status and admission results. Figure 4.4 displays the screen output after pressing the “View Result” button.
There are four admission statuses:

1. **Application Received and Pending Transcript** status means the school already received the student’s application but waiting for the transcript, so both department and graduate college did not start processing the student record yet.

2. **Pending Department** status means the department is evaluating student record now.

3. **Pending Graduate College** status means the graduate college is now doing the final admission approval.
4. **Admission Result Granted** status means the final admission result is published.

When the result is admitted, the admission date and admitted program are shown.

- After the system server starts running, a user can start the department application by launching another DOS window and invoking `runDepart.bat` file. Then a login prompt will display as Figure 4.5.

![Department Login Prompt](image)

Figure 4.5: Department Login Prompt

After login the system by using department ID, the department user interface will display on the screen as Figure 4.6.

![Department User Interface](image)

Figure 4.6: Department User Interface
The “Get Applications” button is for department to get a list of applying candidates. The system reaction after pressing the “Get Applications” button is shown in Figure 4.7.

Then, department can view the selected candidate’s information by pressing the “Get App Content” button. See Figure 4.8.
At the end, department can set the recommendation by choosing the Yes or No radio buttons and press the “Set Recommendation” button as shown in Figure 4.9.

![Image: ECET Department Interface]

Figure 4.9: Set the Recommendation

- After the system server starts running, a user can start another DOS window and invoke `runGrad.bat` to run the graduate college application. The graduate college should use school ID to login the system and the user interface will display on the screen as Figure 4.10.

![Image: ASU Graduate College User Interface]

Figure 4.10: Graduate College User Interface
This user interface is similar to the department user interface except the “Set Approval” and “Announce Admission” buttons. The “Set Approval” button is for graduate college to set the final admission approval. If the approval is yes, the admission date and admitted program data should be input. Figure 4.11 shows the screen output after pressing the “Set Approval” button.

![Figure 4.11: Set Final Approval](image)

The “Announce Admission” button is for graduate college to publish the admission result and trigger the system server to complete the transaction. Figure 4.12 shows the reaction after graduate college presses the “Announce Admission” button.

![Figure 4.12: Announce Admission](image)
4.4 Analysis of Transaction Results

When using transactions with JavaSpaces, clients that are participated in the transaction process are able to see the Entry objects under the transactions. Therefore, the department and graduate college in this system should be able to read the student data all the time and see the intermediate data changes before the transaction commits. However, the student is staying outside of the transaction and not able to see the admission result until the transaction commits. Figure 4.13 shows the status before the transaction commits. The student is not able to see the admission result but both department and graduate college are able to see the result immediately.

Figure 4.13: Before the transaction commits
After the transaction commits, all the clients can see the consistent admission result. The outcome after the transaction commits is shown in Figure 4.14.

With the transaction control by Jini and JavaSpace technologies, the school admission system can always generate consistent results from the unstable network environment. Consequently, the inconsistent cases such as approved by the graduate college but denied by the department can be avoided.
CHAPTER 5: CONCLUSIONS

5.1 Discussion of Problems Encountered

During the system implementation, the following problems are encountered:

- When the URL codebase is not set properly, the system will report “classes cannot found” errors. The URL codebase defines the path for remote clients to dynamically download necessary classes from the network. The following statements should be written properly in order to solve the “classes cannot found” problems.
  1. In setEvnt.bat file: set CODEBASE=d:\jinil_2\lib
  2. In runServer.bat file: -Djava.rmi.server.codebase="file:\%CODEBASE%"
  3. In runStudent.bat, runDepart.bat, and runGrad.bat files:

    -Djava.rmi.server.codebase=http://localhost:%1/

- When updating data objects from the JavaSpace, it requires the take operation before the write operation. If we don’t remove the object from JavaSpace before writing it back, every write operation will input a new copy of the object into JavaSpace instead of overwriting and updating the old object.

- The system developers should find the way to distinguish the null value from the student object that does not exist in the JavaSpace or the null value from the student object is currently under a transaction in order to report proper admission status. The way used in this school admission system is the login process. When the student login the system with student ID, the system server will post the student object into JavaSpace and make sure that student object is existing in the space. Once the student application gets the null value, it means the student object is under an unfinished transaction.
5.2 Conclusions

Jini™ technology provides a robust networking system that enables all kinds of services to join the network and allows clients to download services dynamically. JavaSpace™ is the first Jini service that fully integrated with Jini Transaction mechanism and provides a transaction secured data storage. With Jini and JavaSpace technologies, the implementation of the school admission system is simplified in many ways. First, the system server can focus on the service design and implementation without dealing with complicated threaded socket networking connections. Second, the system can easily access to JavaSpaces from Jini lookup services without handling Java Database Connection (JDBC) processes. Moreover, all JavaSpace operations are inbuilt with transaction mechanism and enable distributed systems to operate correctly in the presence of partial failures. According to the experiences of this project, a distributed system is easy to build and able to deal with unpredictable network failures by using Jini and JavaSpace technologies.

5.3 Future Work

This project can expand in the following ways:

• Including other activities that are involved in the school admission workflow: The school admission system in this project only handles the admission evaluating process to demonstrate the use of Jini transactions with JavaSpace. However, other activities such as verifying students’ immunization records or checking official transcripts should also be implemented for a complete admission workflow system.
• **Security protections:** This project does not implement security protections for the system. The password protections or other data encryptions or decryptions handling can be added to the system.

• **Enable mobile devices to access the admission service by using Jini surrogates:** Many mobile devices are not able to reach Jini network due to their limited power and memory. Thus, PDA or cell phones are not able to download and use this project’s school admission service. However, with the help of Jini surrogate architecture and re-design graphical user interfaces by J2ME programming language, the school admission service is capable to use by any Jini-enable mobile devices.
REFERENCES


http://www.sigda.org/Archives/ProceedingArchives/Dac/Dac99/papers/1999/dac9
9/pdf/09_3.pdf


http://wwws.sun.com/software/jini/specs/jini1_2.pdf


http://wwws.sun.com/software/jini/specs/jini1.2html/core-title.html

Retrieved February 28, 2002 from the World Wide Web:
http://www.cosc.brocku.ca/~cspress/HelloWorld/1999/03mar/
nouveau_networking.html