 UNIVERSITY OF CALGARY	Course Number: SENG 609.22	Course Name: Agent-based Software Engineering
	Session: Fall, 2003	Department: Electrical and Computer Engineering
		Document Type: Project Report

A Multi-Agent System for a Digital Data Warehouse

by

Harprit S. Grewal

(grewal@cpsc.ucalgary.ca)

and

Ibrahim Jadalowen

(jadalow@cpsc.ucalgary.ca)

Submitted as partial fulfillment of requirements for SENG
609.22 (Agent-based Software Engineering)

Course Instructor: Dr. Behrouz H. Far

University of Calgary, Canada

December 8th, 2003

Copyright © 2003 Harprit S. Grewal & Ibrahim Jadalowen

Table of Contents

Abstract	
1. Overall System	
Specification.....	4
1.1 Problem Statement.....	4
1.2 Suggested Improvement.....	4
1.3 Assumptions.....	5
2. Design Documents.....	6
2.1 System Architecture.....	6
2.2 Agent Descriptions.....	7
3. Detailed Design Documents.....	9
3.1 Use Case Diagrams.....	9
3.2 Sequence Diagrams.....	17
4. Communication Specifications.....	21
5. Data Specification.....	25
5.1 Detailed Data Elements Definition.....	25
6. Further Work.....	26
7. References.....	27

Abstract

This document outlines an approach to automate data archiving and management. The operation of data archiving involves a number of steps that requires a coordinated effort of the responsible roles to carry out a task. We use the agent-based paradigm to design a multi-agent system that involves a set of cooperating agents to fulfill the task of archiving the data and managing the business process of client requests and billing. The agents are responsible for managing the archived data and delivering the data to the client with minimal human intervention.

This document contains the problem statement and the suggested improvement from a high level view. It follows with overall system architecture and agent descriptions. Then the detailed design including the use cases and the sequence diagrams in UML notation are presented. The agent communication, using Java Messaging Service, and internal architecture of the agents, implemented as Message-driven Enterprise Java Beans is discussed. At the end, the detail data specification is presented.

1. Overall System Specification

1.1 Problem Statement

The operations at a digital data warehouse involve a set of steps that allow some external data to be housed on the service provider's storage system(s). Here are the high level steps that constitute the process of data archiving and retrieval:

1. Tagging the physical media as it enters service provider's premises (step not applicable to electronic data).
2. Storing the meta-information in the service provider's database that would relate to the actual data.
3. Reading the physical media and storing the data on temporary storage pool.
4. Converting/packaging the data in format that helps the retrieval later on and loading it to the permanent storage pool.
5. When a client requests data back, check the silo where that data resides and after determining the best place to start downloading, transfers that data to the client.

Note: The client requests the data back using the unique identifier that was given back by the service provider when the archiving was completed. The service provider's software tools find a match in the database for the given identifier and know the location of the data on the permanent storage pool. That data is retrieved and delivered either electronically or on other media as per the client's request.

From the above description we can assume that software tools are available to do the tasks involved in the various steps. The transition from one step to another has to be manually coordinated, as the next person in the workflow needs to know when he/she is supposed to execute his/her task. This can sometimes lead to mistakes as the person may either forget his/her duties or they may do it without knowing when they are supposed to be starting. For instance, data should not be loaded onto the permanent storage pool unless it has been properly accounted for in the database. Otherwise we would have data on the storage pool that has no meta-information in the database.

1.2 Suggested Improvement

This system may not be the perfect candidate to implement using the agent-based methodology as the operations involved are internal to the organization, but there are certain aspects that could be automated and delegated to software agents.

From a high level view, we can see five areas where smart agents can help. In step 2, an agent (*Indexing Agent*) allows the execution of software module that records the meta-information in the database only after it determines there exists a link to the client identifier. After the information is entered, it checks if the physical media is available for archiving. If not, it notifies the people on its communication list to have the data-available. The second agent (*Archiving Agent*), which is responsible for step 3, receives a message from the *Archiving Agent* that indicates the completion of step 2. It will consult with the database about the project and other information to complete its task. If this step

were not automated, the agent would still refuse to run the application until the pre-conditions were met. When the *Archiving Agent* is done with its task, it moves the data to a pre-determined location on the network. *Loader Agent*, which is responsible for moving the data to the permanent storage, checks the semantics of the final product before it can proceed. When a particular item has been loaded it consults the database to check if the whole project is done. If indeed it was the last item, it sends a message to the *Billing Agent*. The *Billing Agent* determines how the client should be billed (special rates for certain clients, etc.). If the *Loader Agent* determines the data is not suitable to be loaded to the permanent pool, it checks the creator of that item. If it happens to be *Archiving Agent*, it notifies that agent about the problem, else it will send an email to the concerned personnel for manual intervention. *Archiving Agent* will try to fix the problem, if it can. If not, it will send notification to the concerned parties for manual intervention.

An important requirement for data warehouse is to have redundancy in storage. Data is stored on different locations ideally different cities with sizeable distances between them. This ensures any calamity, natural or otherwise, has no affect on the quality of the service. Therefore, an *Interface Agent* from the service provider that runs on the client machine contact the *Retrieval Agent* that finds if the data is available from the nearest storage pool. If the server on that pool is busy or if that node is totally down, the agent could find the same data on other server, which is located in a different location.

1.3 Assumptions

- DDW gives a unique identifier number to each item that was archived. The client keeps this number to request their data back from DDW at a later date.
- A DDW application sits on the client machine that facilitates requesting of the data through the Interface Agent.
- There is unlimited storage space and we do not reject clients because of lack of space.
- The Billing module takes care of the regular monthly billing.

2. Design Documents

2.1 System Architecture

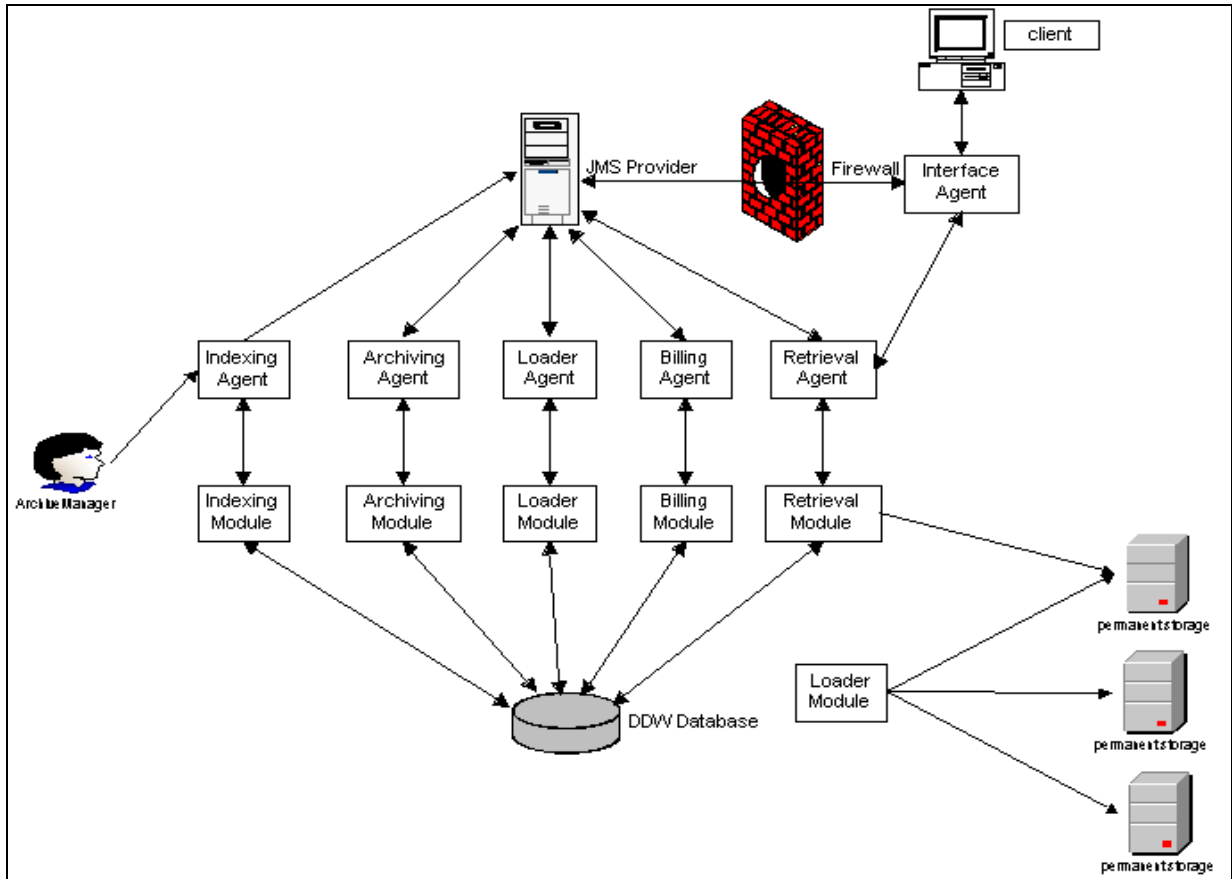


Figure 1 – High System Architecture of Digital Data Warehouse (DDW)

The modules that do the actual work are decoupled from the agents that drive them. This decoupling allows the design to be flexible in case there is a need to introduce a new methodology or to have the flexibility to execute individual modules without being bound to the agents. The agents themselves are implemented as message-driven Enterprise Java Beans (EJBs). The EJB technology allows the agents and their associated modules to reside on different machines yet provide the desirable transparency. The communication between any two agents will go through a Java Messaging Service (JMS) provider's queue/topic.

The agents interface with the modules that do the actual work. They determine when and how the modules are executed. The agents determine the criteria to launch their associated modules either by local knowledge or by consulting special tables in the database (which could be considered as part of their knowledge base).

2.2 Agent Descriptions

2.2.1 Indexing Agent

This agent is necessary as it ensures that the physical data is related with the meta-information about that physical data. For instance, if the data pertains to medical report for a patient, the agent must know which patient this data belongs to (it could be patient identification number in combination of which client it came from). Only when the metadata is available will it allow the module that populates the database tables to proceed.

2.2.2 Archiving Agent

This agent determines what sort of data it is dealing with and applies appropriate algorithms to handle the data. For instance, again taking the medical records as example, if the record to be archived is just textual information about the patients visits to the clinic/hospital, it will handle it as simple text and archive it as is. On the other hand, if the record is an image (x-ray or picture of cornea), it would probably be compressed before it is stored. Each archived item would need a unique number associated with it so that it makes it easier to retrieve that information at a later date. During this process, the original artifact may be renamed or an additional file may be associated with that item which contains the information about the actual data file (index file). Next, the *Archiving Agent* puts the file(s) in a special location on the network and moves on to the next job that it needs to archive. When it is done, it sends a message to a *Loader Agent* to start loading the items to the permanent storage.

2.2.3 Loader Agent

This agent is mainly responsible for loading the archived files to the permanent storage and replicating the files on servers that are at different locations. This agent determines if the file it is about to load is actually loadable. For instance, if some human has by mistake added some extra files in the location that it is pulling from. The permissions on the directory could be changed so that only the archiving agent could write to that directory, but someone with super-user access can still do the damage. When this agent is done with loading the files, it consults the database to check if the job/project is done. If not, it continues its task of waiting for new files to arrive. If the last file it loaded was the last file of the job/project, it sets a flag in the database that indicates that the project is done. At that point, it notifies the *Billing Agent* that the job is complete.

2.2.4 Billing Agent

The *Billing Agent's* primary job is to send invoices to the clients and process client payments. It could send invoices on a monthly basis or per job basis depending on the type of the customer. When the *Loader Agent* notifies this agent, the *Billing Agent* checks the information about the client and decides what action should be taken. If after this

agent has been notified by the *Loader Agent* about a job being done, it determines that the client should be billed at the end of the month; it keeps this information until the month end to bill the client at that point. However, if the client does not have the agreement for month-end billing, an invoice will be sent right away.

2.2.5 Retrieval Agent

This agent is different from the agents above because it is not connected to the workflow of archiving (upstream). Its main purpose is to retrieve the archived data for the client (downstream). Based upon the unique identifier it determines if the data is on the permanent storage pool. If it finds a match for what it is looking for, it will initiate a download for the *Interface Agent* that in turn will pass the data back to the client. If the data does not exist on the permanent storage pool, it will notify the contact people that the data is requested by the customer. It will also notify the *Interface Agent* that the data does not exist, but efforts are being made to deliver the data as soon as possible.

2.2.5 Interface Agent

This agent resides on the client side and contact the *Retrieval Agent* to get the data for the client. It also passes the invoice from the *Billing Agent* onto the client. Also, it is responsible for passing the payment from the client to the *Billing Agent*.

3. Detailed Design Documents for Each Agent for Digital Data Warehouse

3.1 Use Case Diagrams

This section contains the use cases that were identified for the archiving digital data using automated tools.

3.1.1 Use Case Diagram – Indexing Agent

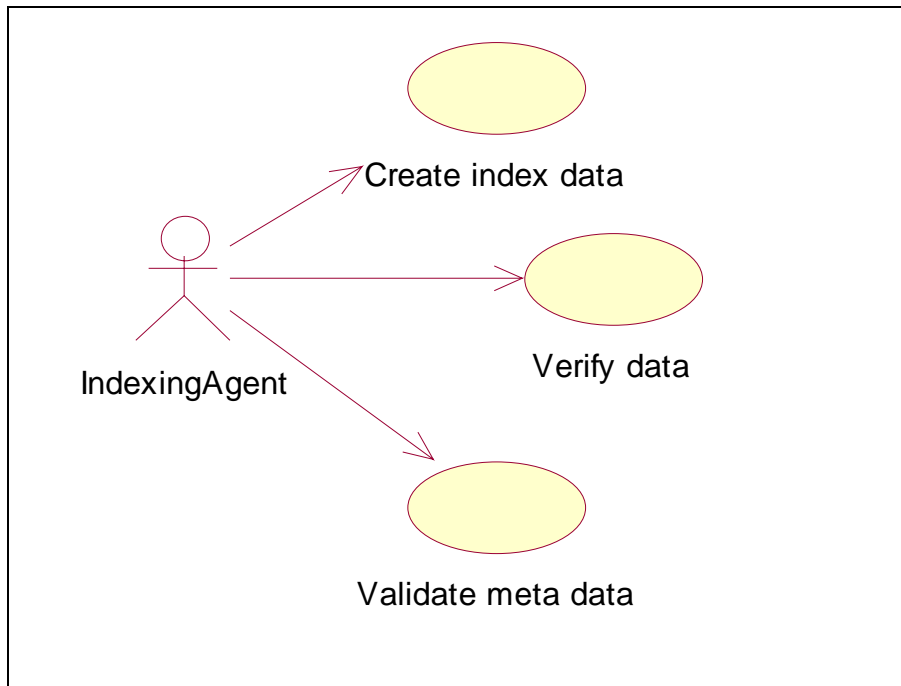


Figure 2 - Indexing Agent Use Case Diagram

The detail use case definition for the Indexing Agent is listed in Table 1 below:

Brief Description	Provides services for verifying and validating meta-data, and creates index for the items.
Pre-condition(s)	Archive manager actor initiates the process
Post-condition(s)	Notifies Archiving Agent after indexing is done
Process Steps	
1.	Verify that the meta-data exists in the index table.
2.	Validates that the meta-data refers to the physical data
3.	Creates an index for each data item
Exceptions	
	Notifies the archive manager if there is not meta-data corresponding to the physical data.

Relationships	
Initiating	Actor (Archive Manager)
Collaborating	Archiving Agent
Data Requirements	
	Meta for each item that has to be archived

Table 1 – Use Case Definition for Indexing Agent

3.1.2 Use Case Diagram – Archiving Agent

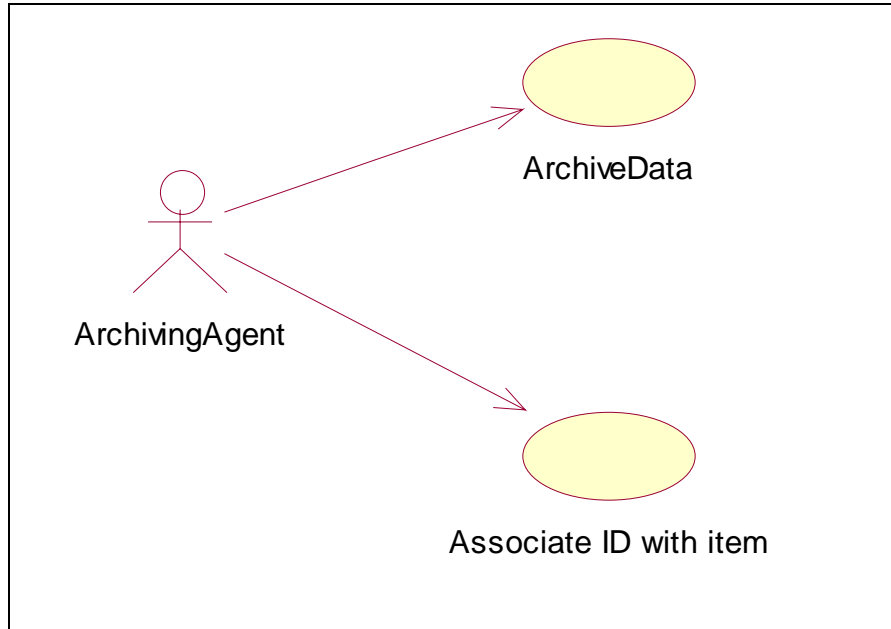


Figure 3 – Archiving Agent Use Case Diagram

The detail use case definition for the Archiving Agent is listed in Table 2 below:

Brief Description	This agent is responsible for archiving the data and associating an item ID with the archived item.
Pre-condition(s)	1. It must have completed any previous jobs before taking on a new job from the JMS message queue. 2. There must be data in the pre-configured location.
Post-condition(s)	Notifies the Loader Agent via JMS that job is complete.
Process Steps	
1.	Determine the data type.
2.	Apply compression based on the data type.
3.	Associate an ID with the archived item
Exceptions	
	1. Notifies the Archive Manager if the data type is not identifiable or it appears that the data is corrupt. 2. The number of items in the specified location does not match with the item indexes in the database.

Relationships	
Initiating	Self, after getting a message from the JMS queue.
Collaborating	Indexing Agent, Loader agent
Data Requirements	
	Physical data that will be archived must be present in the specified location and the meta-data in the index table.

Table 2 – Use Case Definition for Archiving Agent

3.1.3 Use Case Diagram – Loader Agent

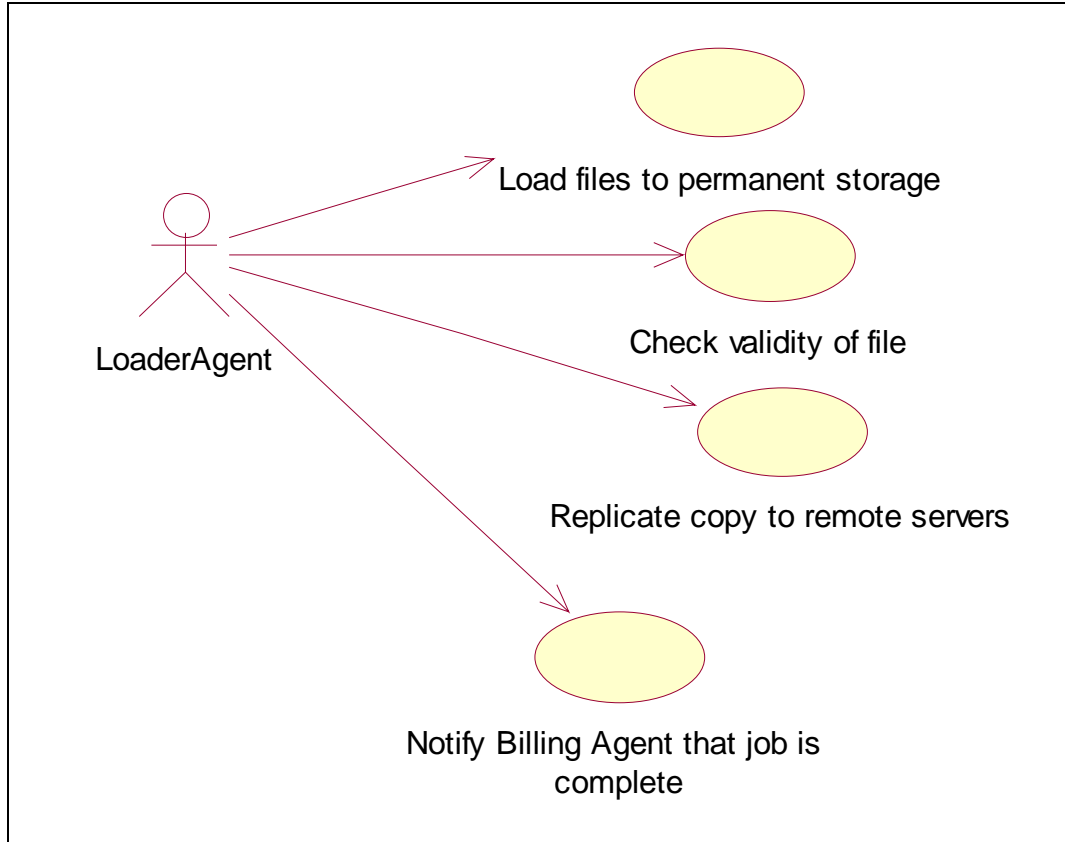


Figure 4 – Loader Agent Use Case Diagram

The detail use case definition for the Loader Agent is listed in Table 3 below:

Brief Description	Loads the archived files to the permanent storage, and notifies the Billing Agent when it is done with all the files for a given job number.
Pre-condition(s)	Items must be archived and its final checks were successful.
Post-condition(s)	1. Replicates to the remote servers and notifies the Billing Agent. 2. Deletes the temporary copy of the data from the local disk.
Process Steps	
1.	Checks if the archived files satisfy the final checks (the archived item has the item ID in the name etc.).
2.	Load files to the permanent storage (using the storage administrator client commands).
3.	Replicates the archived data to the remote servers for backup.
4.	Sends a message to the Billing Agent via the JMS that the job is complete.

Exceptions	
	If the checks failed, notify the Archiving Agent via JMS about the items that failed. If there is an unidentifiable problem, it notifies the Archive Manager.
Relationships	
Initiating	Self, after getting message from the JMS queue.
Collaborating	Archiving Agent and Billing Agent.
Data Requirements	
	The archived data must be present to be loaded and the job and item information should be valid.

Table 3 – Use Case Definition for Loader Agent

3.1.4 Use Case Diagram – Billing Agent

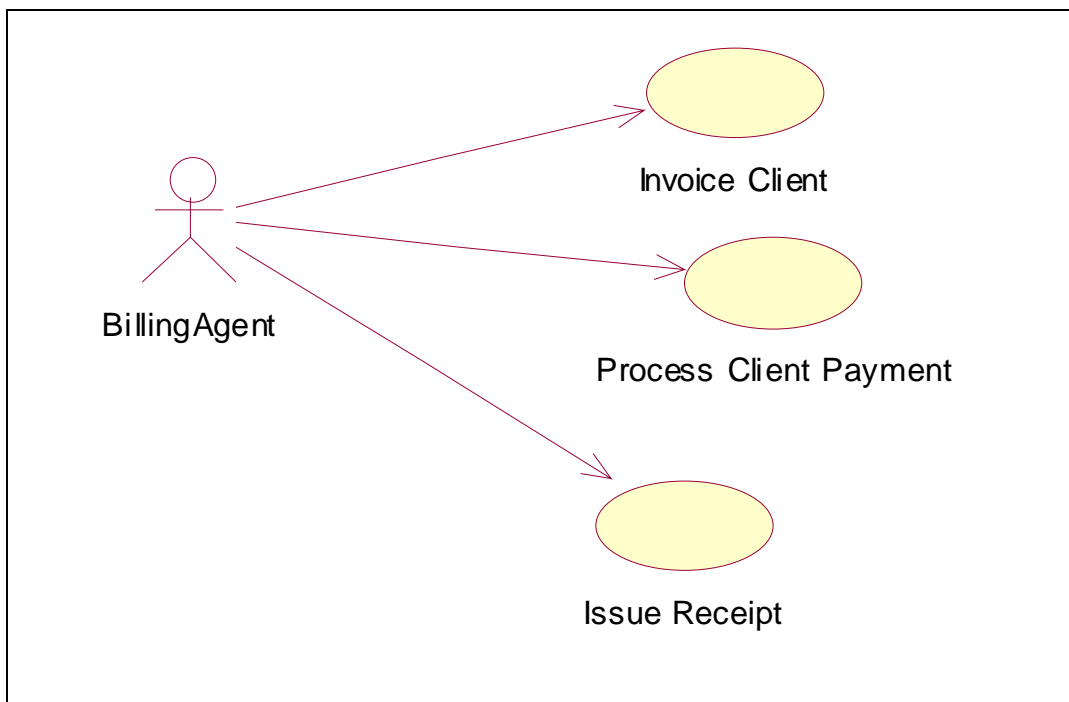


Figure 5 – Billing Agent Use Case Diagram

The detail use case definition for the Billing Agent is listed in Table 4 below:

Brief Description	Invoices the client and process client payments.
Pre-condition(s)	A job must be completed.
Post-condition(s)	N/A
Process Steps	
1.	Sends invoice to the client for a given job number.
2.	Accepts the payment from the client via the Interface Agent.
3.	Sends receipt to the client via the Interface Agent.

Exceptions	
	If the client billing address is invalid, notify the Archive Manager.
Relationships	
Initiating	Self, after receiving a message from the Loader Agent via JMS.
Collaborating	Loader Agent and Interface Agent.
Data Requirements	
	Meta-data regarding the file sizes to exist in database in order to bill properly.

Table 4 – Use Case Definition for Billing Agent

3.1.5 Use Case Diagram – Interface Agent

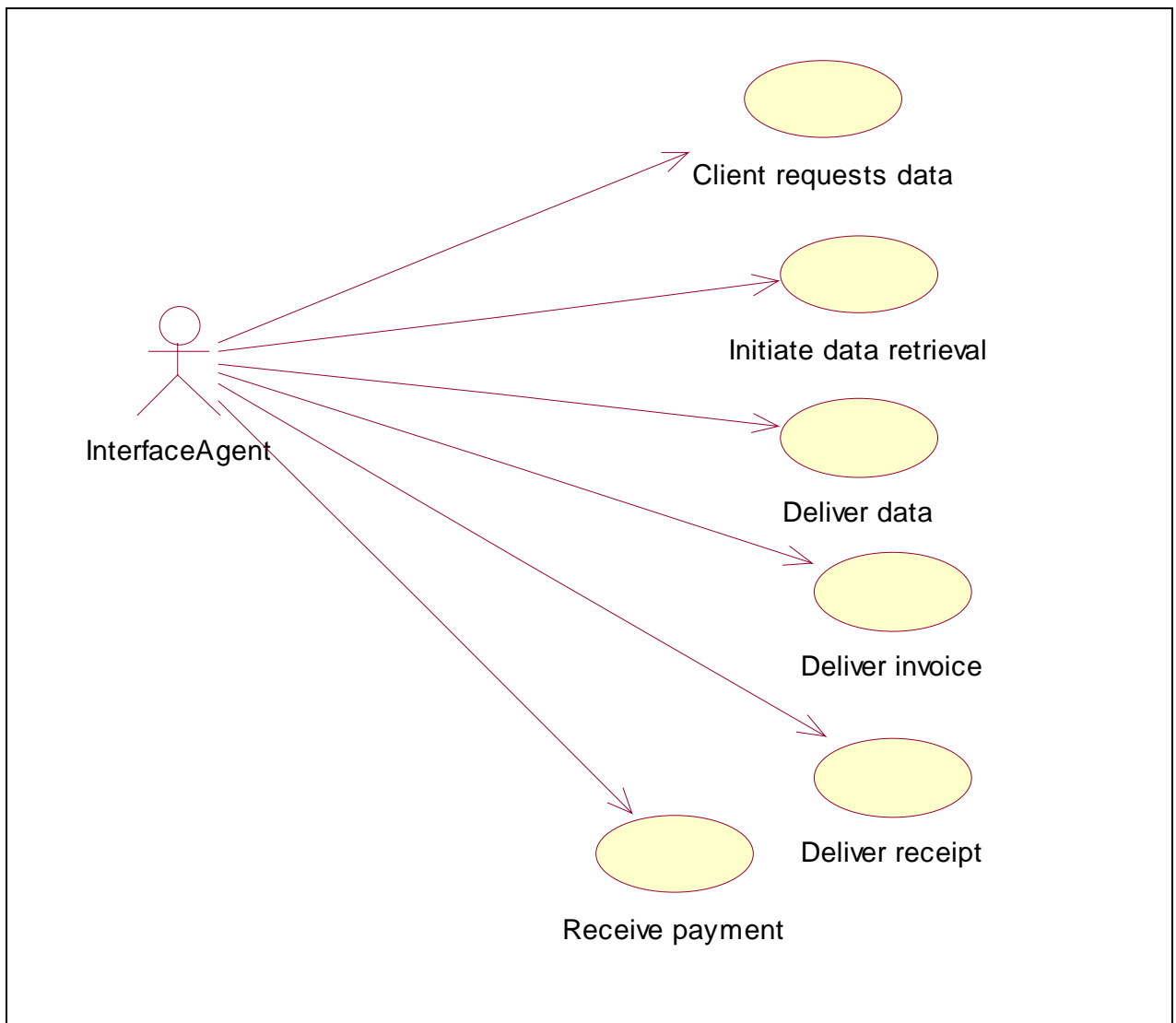


Figure 6 – Interface Agent Use Case Diagram

The detail use case definition for the Interface Agent is listed in Table 5 below:

Brief Description	Handles requests from client and sends system messages to client.
Pre-condition(s)	N/A
Post-condition(s)	N/A
Process Steps	
1.1	Client requests data.
1.2	Notifies the Retrieval Agent about the requested item.
1.3	Waits for the message from Retrieval Agent.
1.4	If the Retrieval Agent sends a message that the data is ready for downloading, this agent initiates the download.
1.5	Delivers the data to the client.
2.1	Receives message from the Billing Agent about the invoice via JMS.
2.2	Delivers the invoice to the client.
3.1	Receives the payment from the client.
3.2	Passes the payment information to the Billing Agent via JMS.
Exceptions	
	If the data is not available for download, it gets a message from the Retrieval Agent and in turn notifies the client.
Relationships	
Initiating	Client (Actor) and Self, using the messages from the JMS.
Collaborating	Billing Agent and Retrieval Agent.
Data Requirements	
	The item being requested must have a valid identifier ID number, and the client should be authorized to access the data.

Table 5 – Use Case Definition for Interface Agent

3.1.6 Use Case Diagram – Retrieval Agent

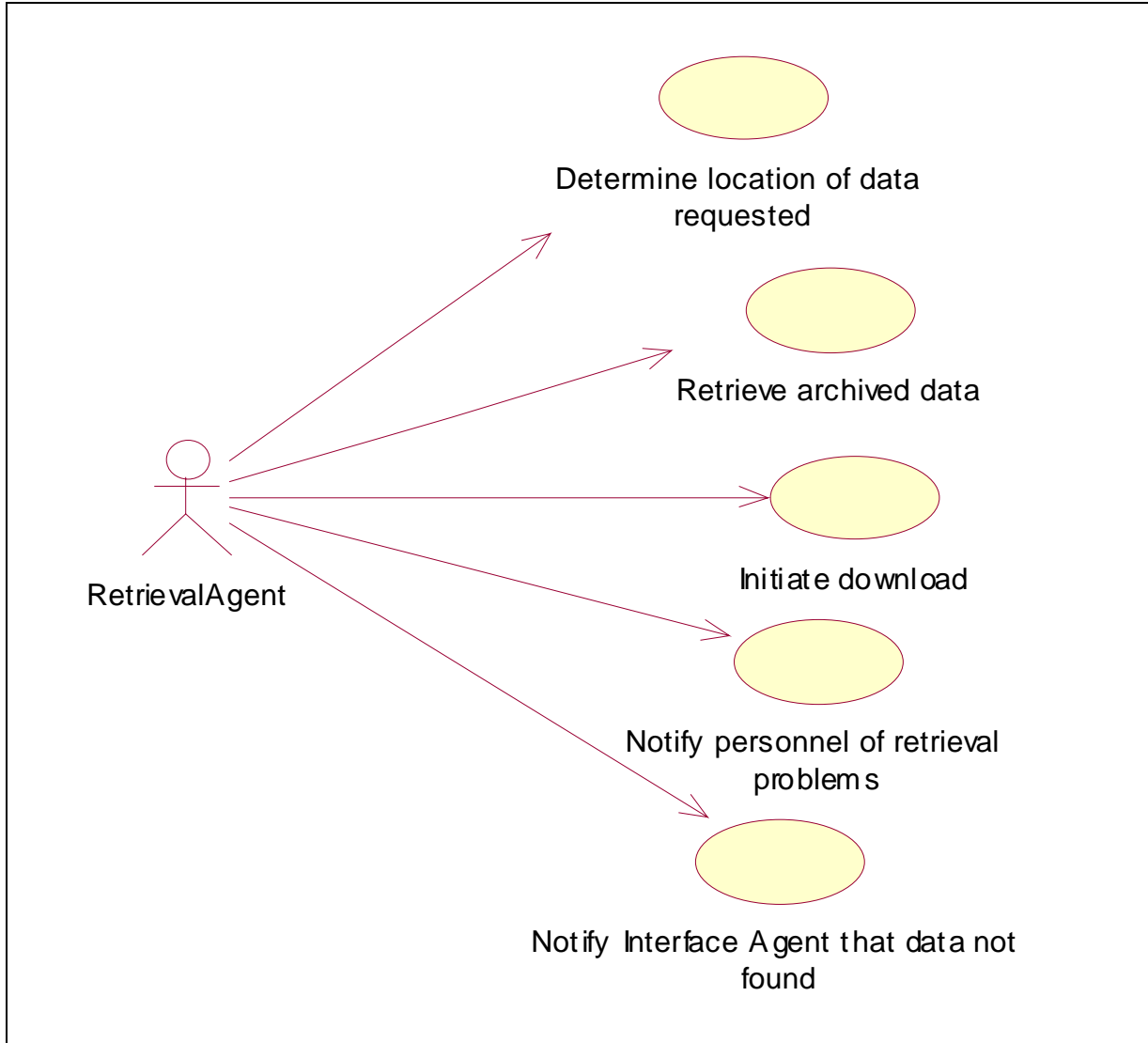


Figure 7 – Retrieval Agent Use Case Diagram

The detail use case definition for the Retrieval Agent is listed in Table 6 below:

Brief Description	Retrieves data from the permanent storage for the client and send it to the Interface Agent.
Pre-condition(s)	There has to be a message on the JMS queue from the Interface Agent.
Post-condition(s)	N/A
Process Steps	
1.	Receives request from the Interface Agent via JMS.
2.	Locates and retrieves the data from the permanent storage
3.1	If data is found, Initiates download with the Interface Agent.
3.2	If data is not found, notifies Interface Agent with error

	message, and notifies the Archive Manager
Exceptions	
	Send error message to the Interface Agent that data identification number is not valid.
Relationships	
Initiating	Self, using the message from the Interface Agent.
Collaborating	Interface Agent.
Data Requirements	
	The item being requested must have a valid ID number.

Table 6 - Use Case Definition for Retrieval Agent

3.2 Sequence Diagrams

In this section, sequence diagrams are drawn to show the sequence of interactions between agents and classes of the system.

3.2.1 Sequence Diagram: Indexing Items

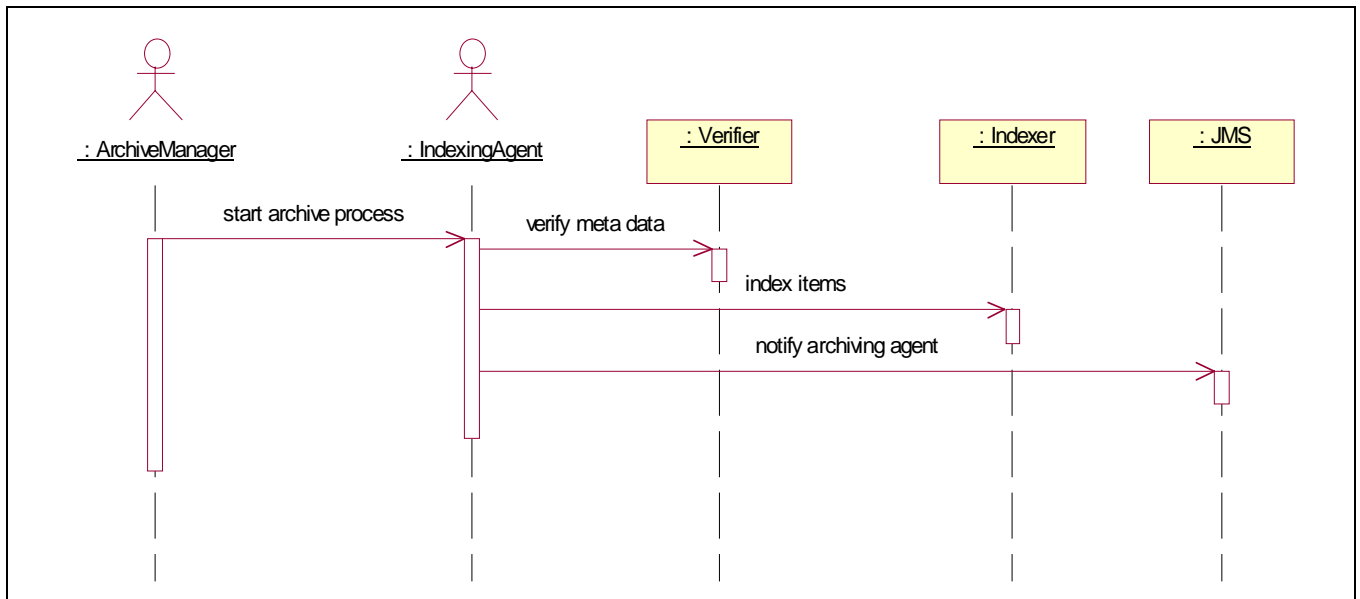


Figure 8 – Indexing Item Sequence Diagram

3.2.2 Sequence Diagram: Archiving Item

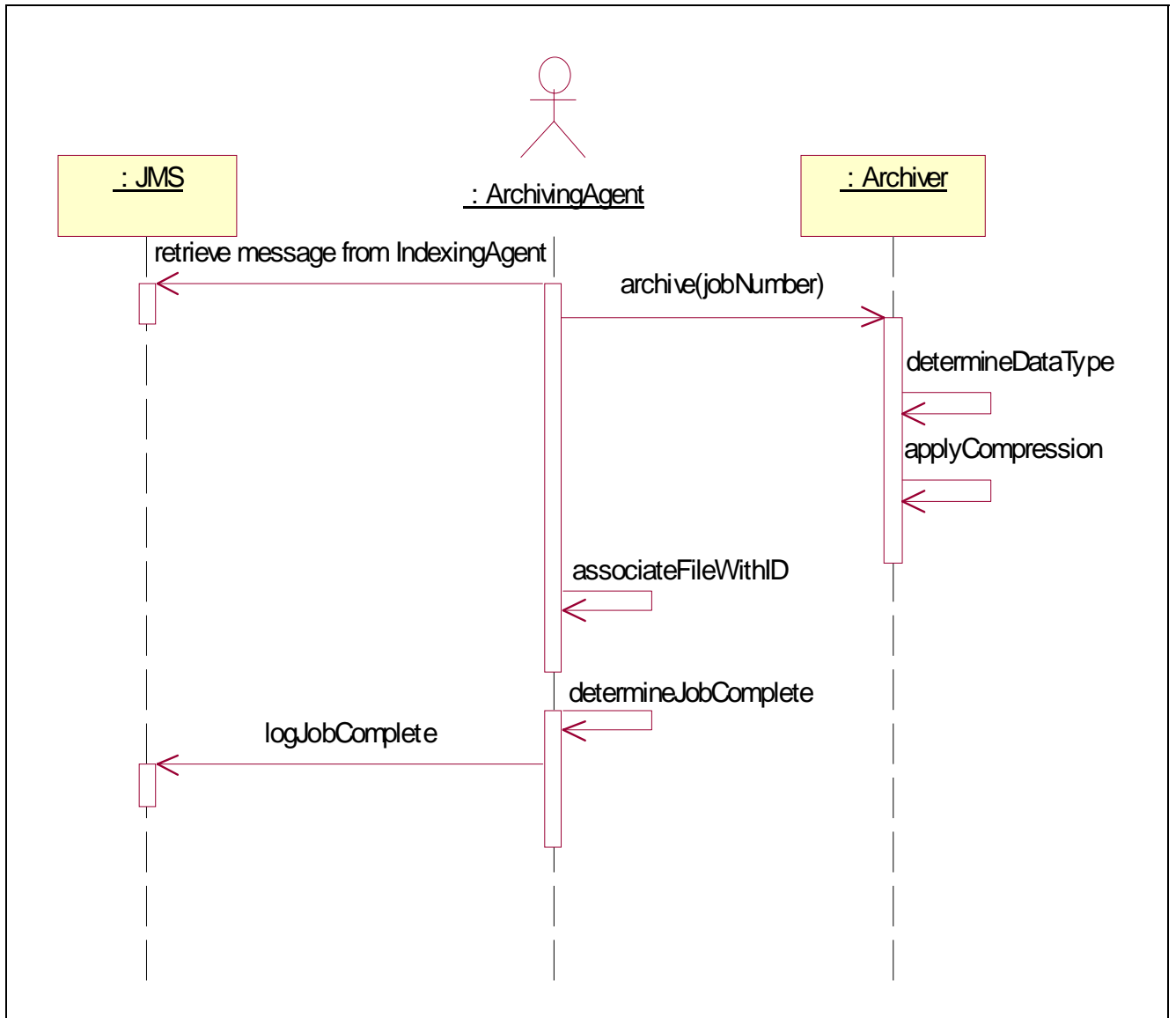


Figure 9 – Archiving Item Sequence Diagram

3.2.3 Sequence Diagram: Loading Items

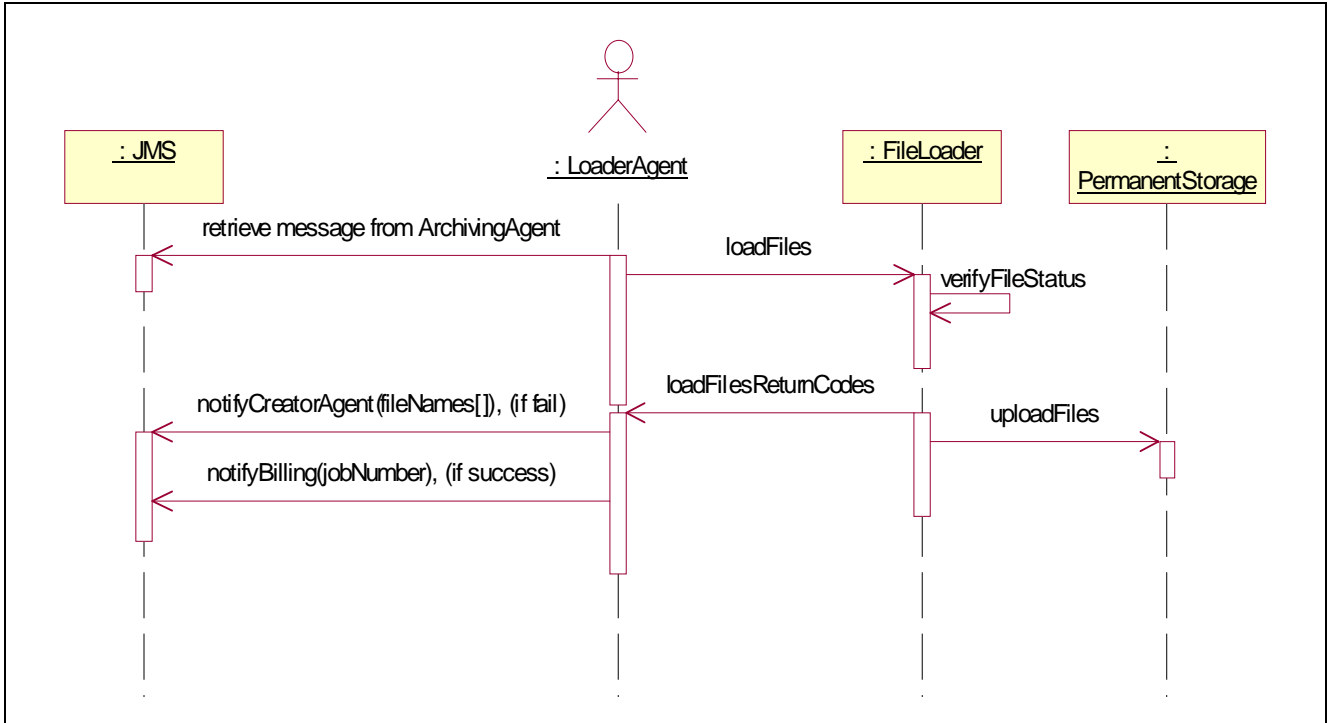


Figure 10 – Loading Items Sequence Diagram

3.2.4 Sequence Diagram: Billing the Client

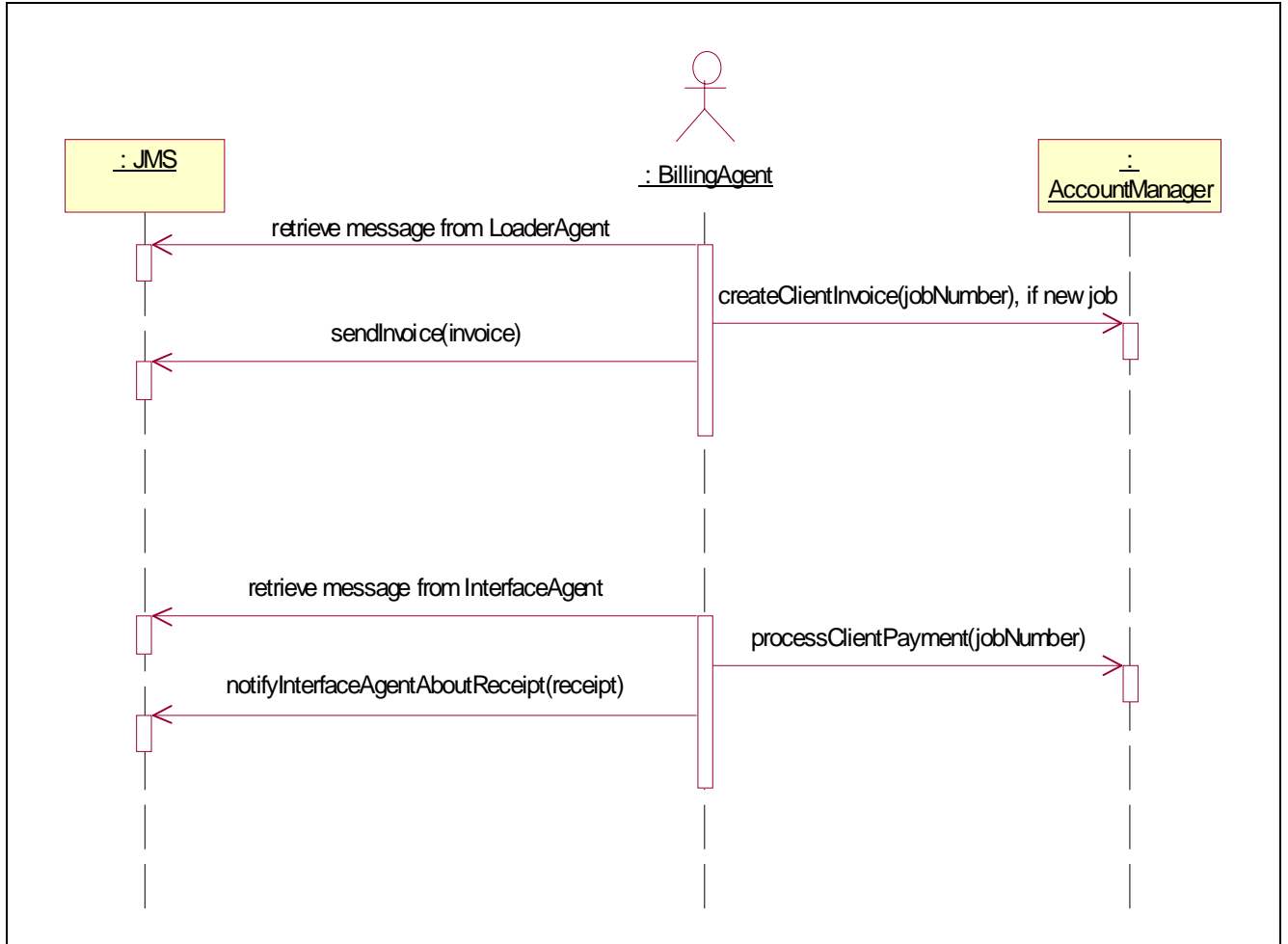


Figure 11 – Billing the Client Sequence Diagram

3.2.5 Sequence Diagram: Data retrieval, Invoicing, and Payment

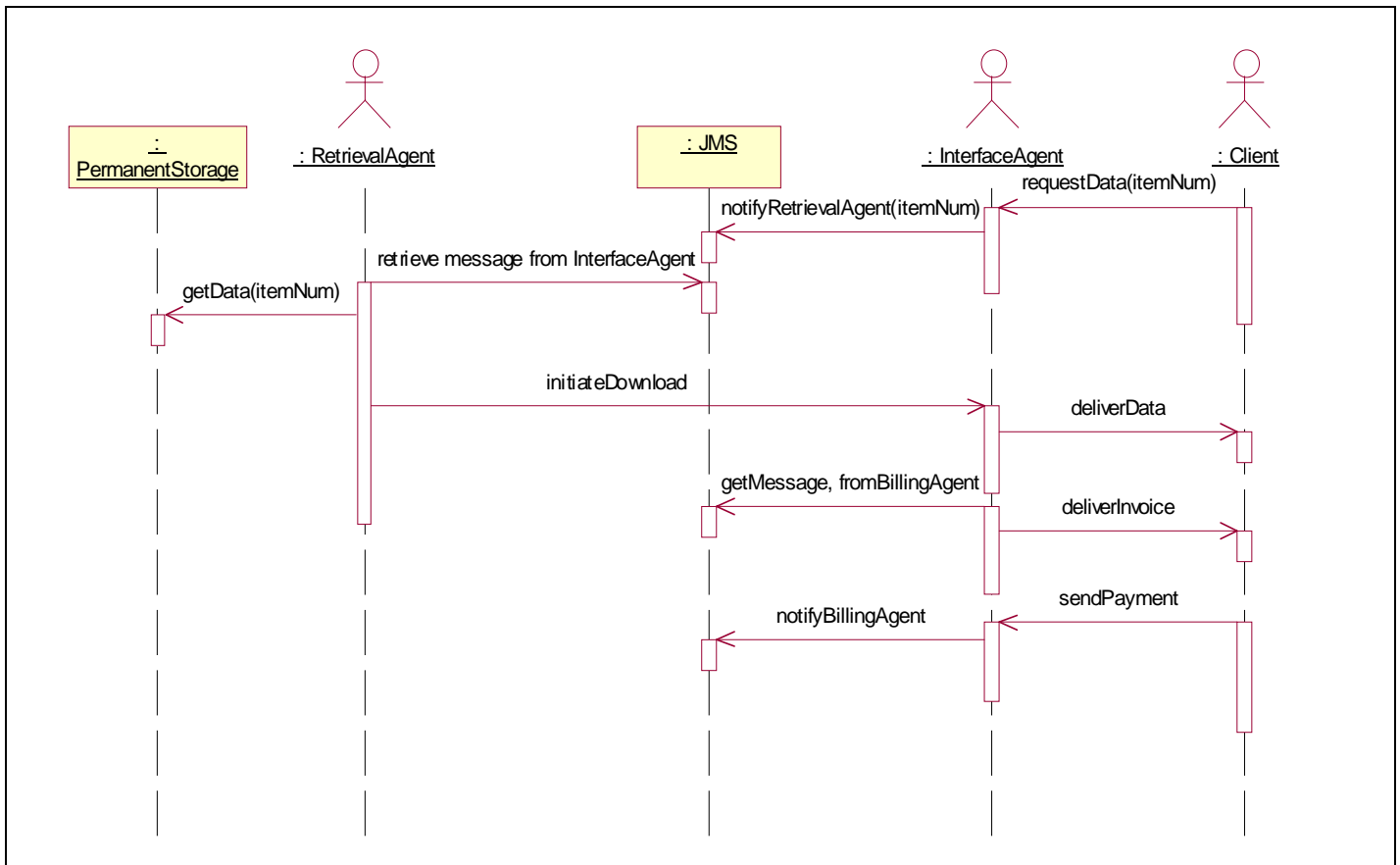


Figure 12 – Data retrieval, Invoicing, and Payment Sequence Diagram

4. Communication Specifications

As shown in Figure 1 and explained in section 2 above (System Architecture), the agents are implemented as Message-driven EJBs which communicate with each other using an implementation of Java Messaging Service. This section will briefly outline the JMS architecture and how it can be used to facilitate communication between agents.

JMS is a Java API that enables messages to be communicated between components. The implementation could be publish-subscribe (all subscribers receive published message) or point-to-point (only one recipient receives the message). For our implementation, we would deal with point-to-point implementation. Each message-driven bean (agent) is

registered with a JMS provider. When an agent wants to communicate with another agent, it sends it a message via JMS. The message is delivered to the recipient as soon as it is ready to accept it (it may be busy when the message actually arrived). This way the sender agent is not bound with the recipient. It sends the message away and continues with its work. This asynchronous behavior is desirable, as it does not stop the flow of things just because one of the modules is busy. The JMS client could be standalone applications that interface with the API or they can be Message-Driven Enterprise Java Beans (EJBs) that hide most of the lower level implementation of sending and consuming from the application developer.

EJBs are distributed components that can be located using their home interfaces via a directory service such as JNDI and have their methods executed as if they resided locally (bean may or may not be on different physical machines).

Message-driven EJBs can interface with the JMS API with very little effort and allows the ability to easily send and receive messages for the agents in our application. JMS defines the concept of *Message* that embodies the message that is to be communicated between the agents. For our application, the agents can communicate using the XML encoded messages and send them as part of the message. The header of the message contains the MessageID and the sender and receiver information. There has to be a DTD or XML schema for a message exchange between two agents

Following messages are exchanged between the agents. Not all messages are presented for the purposes of brevity.

4.1 StartArchiving

This message is sent by the *Indexing Agent* and received by the *Archiving Agent*. The table below indicates the elements of the message that are accepted by the *Archiving Agent*. The 'jobNumber' element indicates the job number that needs to be worked on and 'itemStart' and 'itemEnd' indicate the starting index and ending index for items that need to be processed.

StartArchiving
<pre><DDWMessage="StartArchiving"> <job jobNumber="xxx"> <itemStart>123</itemStart> <itemEnd>456</itemEnd> </job> </DDWMessage></pre>

4.2 LoadData

This message is sent by the *Archiving Agent* and processed by the *Loader Agent*. The *Archiving Agent* invokes this message after it is done with a job. The *Loader Agent* knows which files to load based upon the job number and item numbers that are passed in the message by the *Archiving Agent*.

LoadData
<pre><DDWMessage="LoadData"> <job jobNumber="xxx"> <itemStart>123</itemStart> <itemEnd>456</itemEnd> </job> </DDWMessage></pre>

4.3 BillClient

This message is sent by the *Loader Agent* and processed by the *Billing Agent*. This message just exists to let the *Billing Agent* know that the project is complete. It is up to the *Billing Agent* to invoke the *Billing Module* to bill the client instantaneously or wait till the end of the month to do that. The situation may arise that a job is not completed and it is the end of the month. In this case, again it is the *Billing Agent's* decision to bill for whatever data is archived or wait until the end of the project.

BillClient
<pre><DDWMessage="BillClient"> <job jobNumber="xxx"> <completionDate>date</completionDate> </jobNumber> </DDWMessage></pre>

4.4 FixData

This message is processed by the *Archiving Agent*. This message is sent if the *Loader Agent* determines that the output data was created by the *Archiving Agent* and it does not meet a set of rules. For instance after the *Archiving Agent* finished its job and notified the *Loader Agent*, the disk crashed and the data was lost. In this case the *Loader Agent* would have nothing to load. It will ask the *Archiving Agent* to re-create the output data, if possible and notify the *Loader Agent*. If it cannot fix the problem it will contact the person in-charge to do some manual intervention.

FixData

```
<DDWMessage="FixData">  
  <jobNumber jobNumber="xxx">  
    <itemNum>123</itemNum>  
  </jobNumber>  
</DDWMessage>
```

4.5 RetrieveData

This message is sent by the *Interface Agent* and processed by the *Retrieval Agent*. It will try to deliver the data if it exists on the permanent storage pool. If it does not exist, it will notify the *Interface Agent* that request has to be sent to customer service department.

RetrieveData

```
<DDWMessage="RetrieveData">  
  <clientId>requestorClient</clientId>  
  <jobNumber>jobNum</jobNumber>  
  <itemNum>123</itemStart>  
</DDWMessage>
```

4.6 InitiateDownload

The message is sent by the Retrieval Agent and processed by the Interface Agent. This message tells the Interface Agent that the data is ready for download and it should start a download.

InitiateDownload

```
<DDWMessage="InititateDownload">  
  <jobNumber>jobNum</jobNumber>  
  <itemNum>123</itemStart>  
  <sourceNode>ipAddress</sourceNode>  
</DDWMessage>
```

5. Data Specification

In this section we represent the data specification details by outlining the definitions of the data elements. We have seven core tables in the DDW database schema that facilitate the process of archiving, retrieval, and client billing. There are obviously more tables that are needed, but this is the minimal functional set.

5.1 Detailed Data Elements Definition

This section presents the details of the data that is captured in the DDW database.

Data Element	Description	Type	Mandatory
client_id (pk)	Unique identifier for a client	NUMBER	yes
client_name	Name of the client	VARCHAR2(50)	yes
address	Street address of the client	VARCHAR2(100)	no
city	City name	VARCHAR2(30)	no
prov_state	Region	VARCHAR2(30)	no
country	Country	VARCHAR2(30)	no
phone	Phone number	VARCHAR2(17)	no
fax	Fax number	VARCHAR2(17)	no
date_created	Date this record was created	TIMESTAMP	yes
created_by	Person who created this record	VARCHAR2(50)	yes
date_modified	Date this record was last modified	TIMESTAMP	yes
modified_by	Person who last modified this record	VARCHAR2(50)	yes

Table 7 – Client_Info Table

Data Element	Description	Type	Mandatory
job_number (pk)	Unique job number of each job	NUMBER	yes
client_id (fk)	Foreign key from the client	NUMBER	yes
job_start_date	Start date	TIMESTAMP	yes
job_end_date	End date	TIMESTAMP	yes
num_items	Number of items in this job	NUMBER	yes
job_status	Status (NEW, PROGRESS, DONE)	VARCHAR2(20)	yes
input_path	Location where the items are hosted	VARCHAR(256)	yes
date_created	Date this record was created	TIMESTAMP	yes
created_by	Person who created this record	VARCHAR2(50)	yes
date_modified	Date this record was last modified	TIMESTAMP	yes
modified_by	Person who last modified this record	VARCHAR2(50)	yes

Table 8 – Job Table

Data Element	Description	Type	Mandatory
job_number (fk, pk)	Unique job number of each job	NUMBER	yes
item_number (pk)	Unique Item within a job	NUMBER	yes
input_file_name	Original file path	VARCHAR(256)	yes
size	File size	NUMBER	yes
type	Data type	VARCHAR(20)	no
start_date	Start time and date	TIMESTAMP	no
end_date	End time and date	TIMESTAMP	no
status	Status (from a reference table- NEW, etc)	VARCHAR2(20)	yes
date_created	Date this record was created	TIMESTAMP	yes

created_by	Person who created this record	VARCHAR2(50)	yes
date_modified	Date this record was last modified	TIMESTAMP	yes
modified_by	Person who last modified this record	VARCHAR2(50)	yes

Table 9 – Index Table

Data Element	Description	Type	Mandatory
client_id (fd,pk)	Unique identifier for a client	NUMBER	yes
credit_limit	Credit limit of client	NUMBER	no
status	Status of account	VARCHAR2(20)	no
balance	Current balance owing	NUMBER	no
client_rating	Evaluation rate of client	NUMBER	no
date_created	Date this record was created	TIMESTAMP	yes
created_by	Person who created this record	VARCHAR2(50)	yes
date_modified	Date this record was last modified	TIMESTAMP	yes
modified_by	Person who last modified this record	VARCHAR2(50)	yes

Table 10 – Accounts Table

Data Element	Description	Type	Mandatory
invoice_no (pk)	Unique identifier for invoice	NUMBER	yes
client_id (fk)	Unique identifier for a client	NUMBER	yes
amount_owing	Total amount owing on this invoice	NUMBER	yes
amount_received	Total amount received	NUMBER	yes
job_number	Unique job number of each job	NUMBER	no
fiscal_year	Fiscal year of invoice	NUMBER	no
fiscal_period	Fiscal period of invoice	NUMBER	no
pay_terms	Terms of Payment	NUMBER	no
due_date	Due date of payment	NUMBER	yes
date_created	Date this record was created	TIMESTAMP	yes
created_by	Person who created this record	VARCHAR2(50)	yes
date_modified	Date this record was last modified	TIMESTAMP	yes
modified_by	Person who last modified this record	VARCHAR2(50)	yes

Table 11 – Invoice Table

Data Element	Description	Type	Mandatory
request_no (pk)	Unique number identifying order request	NUMBER	yes
client_id (fk)	Requesting client's ID	NUMBER	yes
requested_by	Client rep. who requested this data	VARCHAR2(50)	no
date_requested	Date this request was made	TIMESTAMP	yes
date_completed	Date this order was completed	TIMESTAMP	yes
status	Status of the request (from ref. table)	VARCHAR2(20)	yes
date_created	Date this record was created	TIMESTAMP	yes
created_by	Person who created this record	VARCHAR2(50)	yes
date_modified	Date this record was last modified	TIMESTAMP	yes
modified_by	Person who last modified this record	VARCHAR2(50)	yes

Table 12 – Request Table

Data Element	Description	Type	Mandatory
request_no (fk, pk)	Request number foreign key	NUMBER	yes
item_id	Unique DDW item identifier	VARCHAR2(50)	yes

	(combination of job number and item_no)		
status	Status of the request (from ref. table)	VARCHAR2(20)	yes
time_completed	Time this item was delivered	TIMESTAMP	yes
date_created	Date this record was created	TIMESTAMP	yes
created_by	Person who created this record	VARCHAR2(50)	yes
date_modified	Date this record was last modified	TIMESTAMP	yes
modified_by	Person who last modified this record	VARCHAR2(50)	yes

Table 13 – Requested Items Table

6. Further Work

This specification presented here was an attempt to automate the process of archiving of digital data with the agent-oriented paradigm of software development. The agents presented in this document are de-coupled from each other and from the underlying modules making them flexible to future changes. They are autonomous in that they realize their duties and try their best to fulfill them. The agents attempt to resolve issues by internal communication before asking for help from outside.

We would have liked to do more analysis of the problem domain and gather more requirements to automate the system further. We hope that we would be able to pursue this further and try out a prototype of the system to prove its viability.

7. References

- [1] Stevens, P. and Pooley, R. (2000) *Using UML – Software Engineering with Objects and Components*, Update Edition, Addison Wesley.
- [2] Fowler, M. with Scott, K. (1999) *UML Distilled Second Edition A Brief Guide to the Standard Object Modeling Language*, Addison Wesley.
- [3] Far, B. H. (2003), Agent Based Software Engineering Lecture Notes, Software Engineering Research Network, University of Calgary, Canada.
- [4] Anonymous (December 2002) *Multi-Agent System for Electronic Realtor*, A Project Report for SENG 609.22, Agent Based Software Engineering, University of Calgary.