MS Project
Satellite Data Ingestion Tool

Nimmi Kalinati
Department of Computer Science
Oregon State University
Corvallis, Oregon

October 2002

ABSTRACT

Satellite data ingestion tool is an automated database application for processing and archiving the ocean and land data that is broadcasted by a remote sensing satellite. This Ingestion system has a two-tier architecture, with data processing algorithm forming the first tier and the database server forming the second tier. The raw satellite data is an HDF (Hierarchical Data Format) file. An HDF reader has been developed that reads this satellite data file to extract the required data. A program has been written, that converts HDF files into images. A database schema has been developed in such a way that all important parameters of the satellite file can be inserted into it along with the locations of data and image files.

This report consists of a detailed description of design and implementation of this ingestion tool along with the design of the database schema.

ACKNOWLEDGEMENT

The project that I have presented here involves ingestion of satellite data and this would not have been possible without Dr. Mark Abbott, who is the Principal Investigator. I am deeply indebted to him for providing me with all the resources and an opportunity to work as a graduate research assistant in the College of Oceanic and Atmospheric Sciences (COAS). I am extremely thankful to my major professor Dr. Bose Bella for being very supportive and providing me with invaluable guidance throughout my stay as a graduate student at Oregon State University. Many thanks to Dr. Bruce D'Ambrosio for agreeing to be my minor professor.

The most important part of this project was understanding ocean data. Curt Vandetta and Jasmine Nahorniak from COAS, were of utmost help in this regard. I am grateful to them for being very patient with my queries and for their constant support in successfully completing this project.

I would like to thank my friends for being highly understanding and motivating. Finally, a big thanks to my family for being there for me all the time and for showering all the love.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 Background

The Earth Observing System (EOS) project, a part of the U.S Global Change Research Program, collects data and analyzes it to develop computer models to study the global ecosystem. The satellite data ingestion tool described in this report has been developed in COAS (College of Oceanic and Atmospheric Sciences), which is one of the data analysis sites for the EOS project. The data is collected either by deploying sensing instruments into the world's oceans or by launching satellites into space. We are mainly concerned with the data that is being collected by satellites. This remote sensing data comes to the college from two sources; one, through the dish antenna installed in the college and two, from external research agencies such as NASA's Goddard Distributed Active Archive Center (GDAAC).

Some years back, due to the unavailability of the kind of technology that is prevalent now, the rate at which data was gathered was slow. Also there were no data centers and researchers had to collect data by their own means. This left very few options for data sharing. But this scenario changed considerably in recent years because, the data collection speed has been constantly increasing due to technological advancements. Satellites have become one of the most common and effective modes of gathering data to study the ecosystem. Since launching and operating satellites are sophisticated and expensive affairs, sharing data has become a must. Research dealing with such high loads of scientific data necessitates development of scientific applications that automate the management, manipulation and analysis of these huge data sets. The EOS-Data Information System aims at working towards this cause with the main goal being design and development of information management and data exploration systems. These systems aid in efficient storage and access to a wide range of data. I developed a database backend by processing and inserting data gathered into a database. The rest of the team members are involved with developing the visualizing tools for the archived data using Active-X and XML technologies.

Once the data is archived here at COAS, it is made available to the scientific community by uploading it on the web. Researchers could then obtain the data from the web site using FTP or HTTP technologies. But this method is not without bottlenecks. With large volumes of data coming in everyday, managing static web pages becomes cumbersome. Also, archiving the data is a big problem that continues to exist. This project doesn't comply with the Distributed Information System that the college is trying to build. And there are still other data sets which have not been archived and need to be taken care of by this project.

The aim of this system was to design a tool that would automate the tasks of processing the earth science and space data and insert it into appropriate tables in the database. It had to accommodate multiple types of data sets to generate a large storehouse of data. It was to form a strong backbone of a multi-tier distributed information system consisting of remote sensing data.

The satellite data ingestion system has been developed using a two-tier architecture. The Java application forms the first tier and the database server at the backend forms the second tier. Since the implementation is in such a way that the application resides in the client machine, it can be called a 'fat' client - 'thin' server architecture.

1.2 Motivation

In the past, data was available in magnetic tapes. It was loaded manually, processed using Perl scripts and then ingested into SQL Server in batch mode, using 'bcp' (Bulk Copy Program) utility. But this method was not sophisticated enough to handle high loads and varied formats of remote sensing data that are now being collected through cutting edge data transmission-reception technology, such as the direct satellite broadcasting system.

Perl scripts were used with the data collected from the ocean sensors to order the column formats, based on the corresponding table columns. Using 'bcp' utility, data files were then inserted into the database. The image data was separately inserted using the 'text copy' utility.

The following are the requirements for using 'bcp' utility:

- The data file should contain data in text-only format or a format previously generated by 'bcp' utility, such as native format.
- Destination must already exist.
- Columns in the table should be compatible with the fields of the data file being copied.
- Access permissions on source and destination tables should be set.

The 'bcp' utility, with such rigid requirements seemed fine for small ascii files with fixed length data . But it was soon realized that satellite data files containing variable length image and ascii data, with sizes that ranged from gigabytes to terabytes, could not be effectively manipulated using this utility.

The disadvantages of adopting this command prompt utility to handle the ingestion of satellite data has been listed below:

Data Size

The use of 'bcp' utility for even small ascii data file, requires much manual intervention to set it in the format required by the table. With large volumes of data coming in at a rate that has been increasing rapidly, using this utility will severely slow down the ingestion process.

File Format

The 'bcp' utility can work only if the data files can be represented in row-column format. But the satellite data files are in a format called HDF (Hierarchical Data Format) which is not understandable by this utility.

Content of Data

For some earlier satellite data sets, it was required that the data files and their image files be inserted into the database. The data files could be stored in the database as character objects and image files as binary objects. BLOB (Binary Large Object) and CLOB (Character Large Object) are the data types in JDBC that support this kind of data. Both these data types are capable of storing very large data in the corresponding formats. A logical pointer points to the object in the database that the instance of the data type represents. This pointer is of 16-byte size and it points to the data stored in a collection of 8 KB pages. These pages are not always located next to each other and are ordered logically in a B-Tree structure. The image and data files were converted into streams of data for convenience and then inserted into the database.

But, using bulk copy utility to insert data into these large object becomes a problem. This is because it is not possible to insert an image or text stream in between columns of data and delimit it with any terminator.

Metadata

The bulk copy utility requires that all the data to be inserted into the table be available in the data file itself. Information such as when and where the data file was collected is present in the form of metadata (description of data) in each file. But some data sets are not in self-describing format. So for such data sets, information has to be gathered from the researcher and inserted manually into the data files, which eventually increases the execution time.

Hence, the complexity and varied formats of data files, along with their massive sizes necessitated the design and development of an efficient and robust ingestion tool.

CHAPTER 2. SATELLITE DATA INGESTION TOOL

2.1 Overview

There are several satellites that are being launched and controlled by NASA as part of its Earth Observing System project. 'Terra' is one such satellite, which was launched in December 1999 to unravel the mysteries of climate and environmental changes. There are several instruments onboard this satellite to measure various parameters of Earth. The Moderate Resolution Imaging Spectroradiometer (MODIS), is the keystone instrument on Terra. It is an imaging radiometer employing multiple in-track detectors, a cross-track scan mirror, collecting optics, and a set of individual detector elements which will provide imagery of the Earth's surface and cloud cover in 36 discrete spectral bands. Contiguous scan swaths of 2330 km in cross-track by 10 km in-track are acquired to provide 2-day repeat observations of the Earth, providing scientists a nearly comprehensive global view. The data obtained, which is of the order of a few tera bytes at one pass, is intended for the purpose of understanding the short term and long term atmospheric trends, as well as, regional and global phenomena. The main goal of this project has been creating a system to receive this volume of data from MODIS, organizing the diverse data products, and then making them available to the research and scientific community.

There are two phases involved in design and development of the automated satellite data ingestion tool. One is the design and development of the ingestion procedures and the other is the design of database schema.

1. Ingestion System

There are three factors that had a major influence on the design of the ingestion system:
1. Format of the data files, 2. Requirements from the scientists and 3. Geographic location of the raw data. The main focus was to build a robust system that could handle a perpetual flow of data in large volumes and seamlessly ingest them into the database as they are received. The satellite data that is available is either collected through the dish antenna installed in Oregon

State University by Ocean Physics and Ecology Laboratory or from a data archiving center called Goddard Distributed Data Active Archive Center (GDAAC). Data from both the sources is visualized through high-resolution images. The system on the whole is a sophisticated one with processing algorithms running in the background to ingest the data.

2. Database Schema

Designing of the database schema for this data accounted for a major portion of work in developing this system. Database schemas were designed, based on the format of the data files and the particular needs of the scientists who would be using this data. Database design is represented using Entity Relationship diagram, under the "Implementation" topic in this report. The database consists of a main table that contains common information about all the transmission devices onboard a satellite. The other tables are the ones that store data that is specific to each transmitting device. The initial idea was to ingest the image created for each data file into the table along with the data file. This could have been done, by having a data type called "image" in the table. JDBC (Java Database Connectivity) supports methods like *setBinaryStream.* This method opens a Java class file as an *InputStream* object and uses that *inputstream* to populate a column in the database. When a SQL INSERT statement is executed, the bytes are read from the stream and stored in the image column of the row inserted into the database. But this method could not be implemented due to two major reasons. Firstly, since the image was to be made available on the web, the IIS server would query into the table, the image from this column would be copied into IIS server bit by bit and then it would draw image onto the user's monitor. All this would take several minutes and things would worsen if the user had a slow speed connection. Secondly, the JDBC driver, MERANT's DataDirect SequeLink driver, that we used did not support ingestion of data into "image" field of the database. As an alternative to storing the data and image files directly into the database, the paths where these files reside in the machine are put in the tables. As the next phase of this project, the goal is to invoke the processing algorithms from different machines based on the type of data set. A daemon process running on a machine will monitor the arrival of the data set. On the arrival of data, based on the type of data set, the corresponding processing algorithm will be triggered. The processing algorithm works on the data and populates the appropriate table in the database. The main aim is to get the whole system working with much less human intervention.

2.2 Satellite File Format

The EOS data arrives in a format called Hierarchical Data Format (HDF). The Hierarchical Data Format is a multi-object file format for the transfer of graphical and numerical data in a distributed environment. The National Center for Supercomputing Applications (NCSA) in University of Illinois developed HDF to cater to the needs of diverse groups of scientists working on projects in varied fields. For each data object in an HDF file, there is information about the type, dimension and amount of data including its location in the file. Being "multi-object" means that it allows multiple types of data such as symbolic, numerical and graphical data within one HDF file. NCSA HDF libraries provide command utilities to analyze the structures of existing HDF files and display their contents on the user's screen. There are several software tools existing, including Collage, IDL, Noesys, HDF Explorer etc. to view these files. HDF was designed in such a way that it addressed many specifications of scientific data, such as the following :

- ➢ Portability to multiple machines.
- ➢ Self documented.
- ➢ Capable of storing multiple data structure or types within the same file.
- ➢ Efficiency in storage and access of large data sets.
- ➢ Extensibility for future enhancements and compatibility with other standards.

2.3 Requirement Analysis

This section has the description of important high-level design and system requirements, which are the basis for defining the architectural design of the ingestion tool.

2.3.1 Design Requirements

Extensibility

The system that we develop must be extensible so that in case a new data set arrives we should be able to add new components to the existing system without much difficulty. The system should be able to take care of new data types with as few changes as possible.

Adaptability

There has been a steady change in the format of data. The system should be able to accommodate it with few changes and adapt to the existing database schema. This is a crucial requirement since the distributed information system being developed relies on the database and any significant changes in the existing schema will result in rewriting major portions of the code.

Centralized Database Schema

A centralized approach has been followed in designing the existing schema and the system being developed should follow the existing principles. Since there is a range of data sets available, following these principles becomes difficult. The main reason behind creating such a design is that it would reduce the number of tables containing the information and thus make work easier for future developers of the information system.

Processing of Data

Before being ingested into the database, the satellite data needs to be processed using various algorithms. The number of processing stages and intensity of processing may vary depending on the amount of data and the computational complexity of the algorithm being used. The system should provide software and hardware resource for efficient execution of such processing tasks. The volume of satellite data is in the range of gigabytes and terabytes. Hence the system should be robust enough to handle the pressure of the load.

Portability

The system should be portable to other environments in case of any future requirement that requires it to be run on a different operating system. As two most popular platforms are Unix and Windows, this tool should be able to be executed in both environments.

Performance

Performance of the system is a crucial factor, since there is going to be a continuous flow of data from the satellite dish. The goal is to procure the received data and concurrently process and ingest it into the database. The system should be able to handle fluctuations in the load. The performance of the system should not degrade as the amount of data transfer increases.

2.3.2 System Requirements

The ingestion system developed was supposed to meet a few system requirements.
The main requirements being :

- It should be portable on two major platforms – Sun Sparc and Windows NT.
- The system developed should be able to interact with the backend database.
- Database access over the network should not be a bottleneck.
- Manipulation of JPEG and HDF files.

Taking all the above criteria into consideration, Java emerged as a unanimous choice. This application tool has been developed using Java 2 (Production Release) platform. It uses JDBC to communicate with the database. Connectivity to MSSQL Server database is provided by SequeLink JDBC driver.

JDBC

JDBC is an application programming interface for accessing virtually any kind of tabular data. It consists of a set of classes and interfaces written in the Java programming language that provide a standard API for tool/database developers to gain access to a wide range of databases, either directly or through middleware. JBDC can be used to connect to various kinds of data sources by using appropriate drivers. Advantages of using JDBC are listed below :

- It makes the access of database over the network easier
- It can access multiple database servers within a single transaction.
- It allows connection pooling which results in performance enhancement. A connection pool is a cache of database connections that is maintained in memory so that the connections may be reused.
- It makes it easy to send SQL statements to relational database systems and supports all dialects of SQL.
- It has updateable result set, which gives the ability to use Java programming language commands rather than SQL.

SequeLink JDBC Driver

Connection to SQL server was obtained using MERANT's DataDirect SequeLink 5.0. This is a middleware product that provides point-to-point connections from client to server using the latest data access standards. It allows central configuration of the data access environment and data access activity management. It has the following features :

Data Connectivity – SequeLink provides universal data connectivity for the latest JDBC standards to a variety of data stores. Its component implementation allows one to manage one's entire data access environment regardless of the underlying operating system on which the SequeLink component runs. Also, SequeLink's client component is database independent and so no extra client components are required to incorporate additional data store technologies in data access infrastructure.

Interoperability – It allows one to leverage existing and evolving technologies by adhering to industry standards. It allows use of Lightweight Directory Access Protocol (LDAP) for centralized connection and configuration information and provides secure internet and intranet communication.

Security – The messages between SequeLink middleware components that involve data requests and data transmitted over the network, internet or intranet can be scrambled. It supports authentication mechanisms provided by the database and/or the operating system on which its components run. Also, it supports read-only data connections to keep the data in the data store secure from updates.

Systems Management – SequeLink provides Reliability, Availability and Serviceability (RAS) by providing dynamic service attributes.

Scalability – It provides superior performance and scalability through connection pooling at the client through efficient use of server resources.

2.3 System Architecture

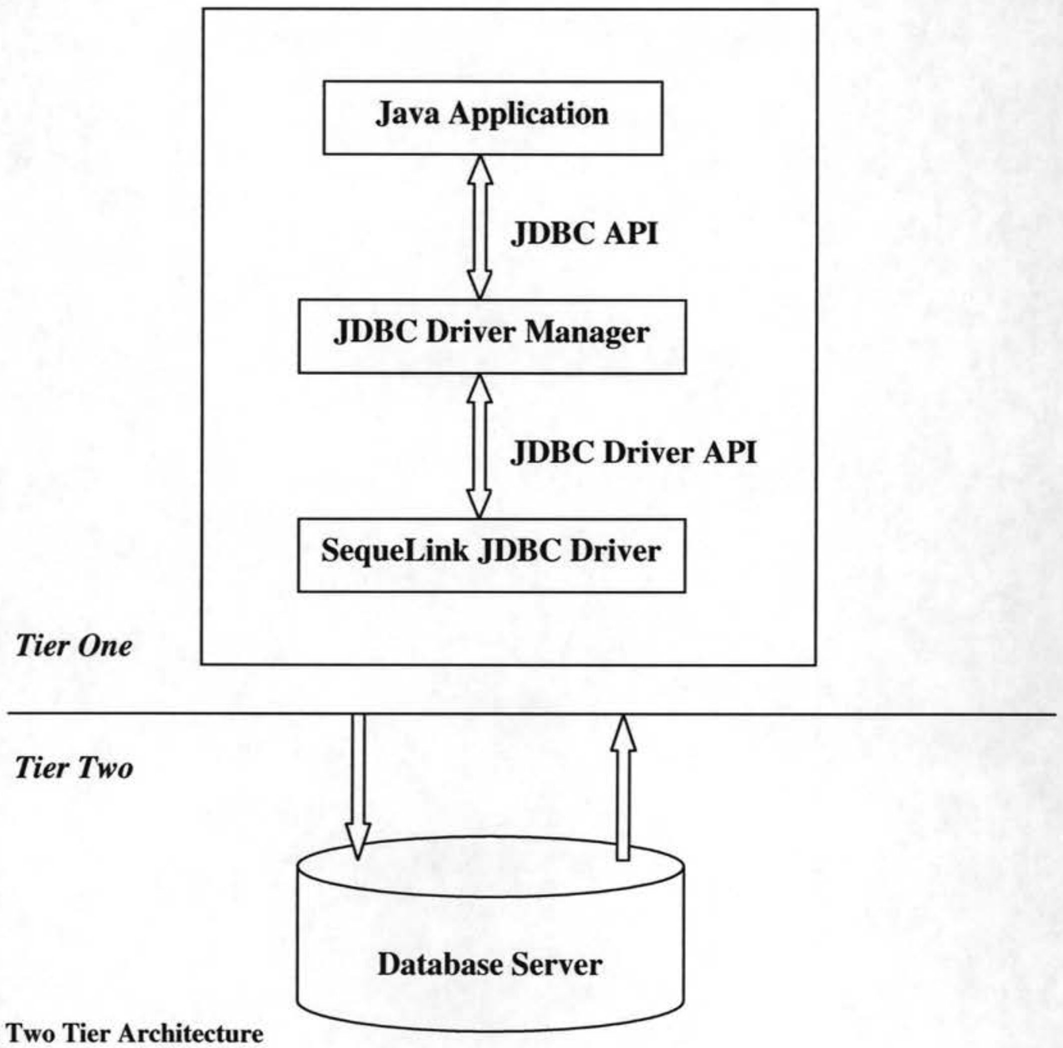

Figure 1: Two Tier Architecture

This satellite data ingestion tool has a two-tier architecture. The application forms the first tier and the database server forms the second tier. The diagram (Fig. 1) shown above represents the framework of this system. The JDBC architecture is based on a collection of Java interfaces that together enable connection to the data sources, creation and execution of SQL statements and retrieval and modification of data in a database. The command-line application developed in Java, interacts with the database at the backend using JDBC API. JDBC API sends queries to a database specific driver using JDBC Driver Manager. JDBC Driver Manager interacts with the JDBC Driver using JDBC Driver API. The driver actually connects to the database and returns the information from the query or performs the action specified by the query, the communication being duplex with both the queries and the result sets travelling in both directions.

Satellite Data Files

The data files received from the satellite are in a self-describing format called HDF (Hierarchical Data Format). HDF is a physical file format that allows storage of many different types of scientific data including images, multidimensional data arrays, record oriented data, and point data.

The satellite data files collected,

- were processed using special libraries to produce level 3 files that have to be finally ingested into the database.
- have metadata, describing their format and content.
- have a very high volume.
- have to be processed into "JPEG" images and then their locations have to be ingested into the database as well.

The system extracts the useful and required information from the metadata, creates image files and then ingests the information extracted along with locations of files and images into the database.

CHAPTER 3. IMPLEMENTATION

The satellite data ingestion system follows a two-tier architecture, with the command line application developed in Java forming the first tier and MSSQL database server forming the second tier. The pipeline involves data acquisition, processing, extraction of metadata and finally ingestion into the database. This system ingests the Moderate Resolution Imaging Spectroradiometer (MODIS) data, which will be discussed in detail in the following sections.

Use Case Representation of the Ingestion System

The use case described below showcases the functioning of the whole system in brief. A use case is a collection of possible sequences of interactions between system under discussion and its external actors, related to a particular goal.

USE CASE # 1	Ingest satellite data	
Goal in Context	To ingest the satellite data into the database upon its arrival.	
Scope & Level	College of Oceanic and Atmospheric Sciences Modis Data Ingestion Project	
Preconditions	Satellite dish collects real time data. GDAAC data arrives in COAS.	
Success End Condition	Data is archived in the database	
Failed End Condition	Data is lost or still in the parent directory in which it arrived	
Primary, Secondary Actors	Database administrator and the agent (computer) acting for him	
Trigger	Arrival of satellite data	
Description	**Step**	**Action**
	1	Satellite data arrives.

	2	Invoke the processing algorithms.
	3	Data is ingested into the database.

The above table depicts a set of paths called scenarios, that traverse an actor (an object external to a system) from the trigger event, which is the start of the use case to the goal, which is the success scenario. The database administrator primarily uses this system. The arrival of the satellite data triggers the ingestion process. The final state is either a success or failure, based on the completion of the ingestion process.

3.1 MODIS Data

MODIS data is available in different types, based on various research requirements. They are primarily in three categories : Level 1, Level 2, and Level 3. Level 3 data is our main concern, as that is the type of data that this ingestion system will be archiving.

Level 1 A and Level 1 B – This data consists of calibrated radiance and geolocation details along with spatial resolution, temporal coverage, approximate file size in megabytes and the transfer rate in files/time.

Level 2 – Each Level 2 granule represents five minutes of Terra viewing. This data consists of ocean color products collected during the day and sea surface temperature products collected both day and night. There are thirty six ocean color parameters and four sea surface temperature parameters available.

Level 3 – This level of data is of two types, Level 3 binned data and Level 3 mapped data.
Level 3 binned products are global products. Spatial bins have 4.63 km spatial resolution in an integerized sinusoidal equal area grid. Only bins with data values are present; land bins and bins with no data are not in the files. This data is categorized as daily, weekly, monthly and yearly, based on the duration in which it is collected. Level 3 mapped products are global products as well. Bins for the entire globe are present in the data files, including fill values for

land bins and bins with missing data. Spatial resolution varies per product: 4.63 km, 36 km, or 1 degree.

3.2 Data Processing and Ingestion

The Level 3 MODIS data, which is our primary interest, is obtained either by applying processing algorithms on the raw satellite data collected by the dish antenna at Oregon State University, or from GDAAC (Goddard Distributed Active Archive Center), NASA. When the data arrives from the satellite, it is in digital form containing just strings of numbers. This data has to be converted into a meaningful remote sensing data for any further use by the scientific community.

3.2.1 Data Processing

The data processing algorithms were developed by RSMAS (Rosenstiel School of Marine and Atmospheric Science), University of Miami. Figure 2 shown below represents the data processing pipeline.

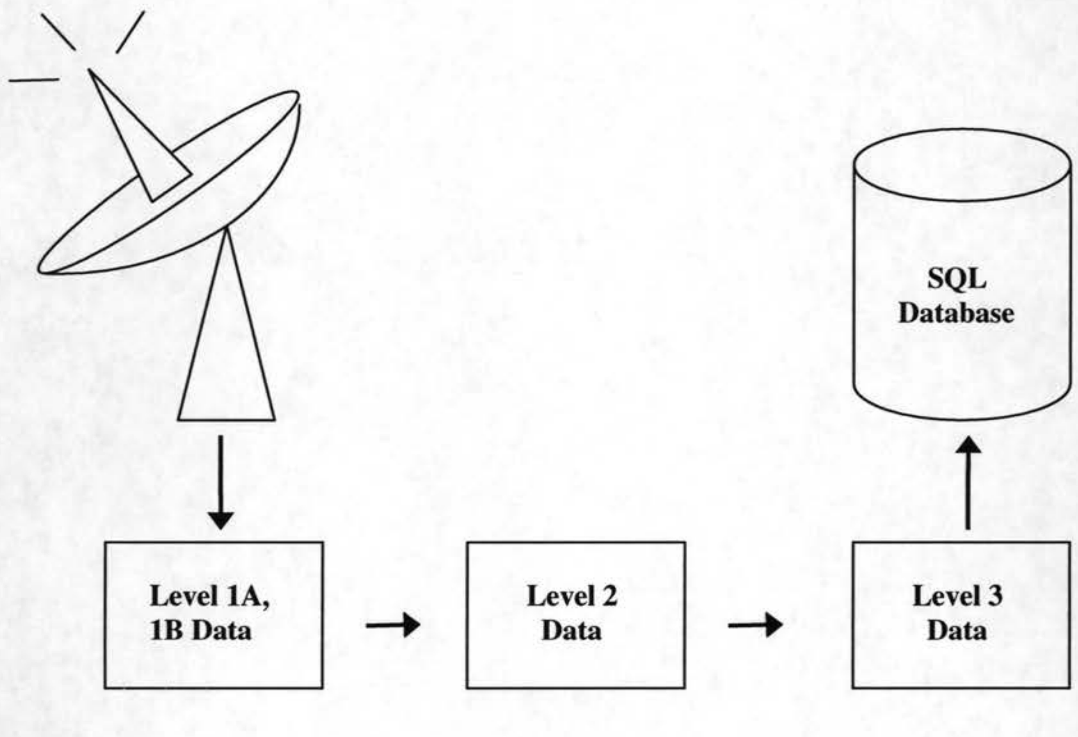

Figure 2: Data Processing Pipeline

Raw data is in original packets, as received from the satellite. This data which is at its original resolution, time ordered, with duplicate packets removed, is reconstructed into unprocessed instrument/payload data at full resolution with all communication artifacts like synchronization frames, communication headers etc. removed and forms the Level 1A data. This Level 1A data that have been processed to sensor units and radiometrically corrected and geolocated, forms the Level 1B data. Level 2 data comprises the products, ocean color and sea surface temperature (SST), calculated from the Level 1B data at the same resolution. Level 2 products mapped on uniform space-time grid scales, usually with some completeness and consistency form Level 3.

One important data processing aspect that had to be taken care of before ingestion, was the date/time format of MODIS data. This information is not explicitly present in the metadata that is extracted. The only values available are, “Year”, “DayofYear” and “Millisecond”. It was very important to calculate accurate date and time when the image was scanned by the satellite from this information, as queries to the data set tables are usually performed using time periods. The DateAnalyzer class in the application takes care of this. Suppose the value of “Year” is 2001, “DayofYear” is 56 and “Millisecond” is 43800000, the *Calendar* class in Java takes “Year” and “DayofYear” as parameters and returns the date. The “Millisecond” value is divided by 1000 once and 60 twice. The integer portion of the result is the hour value. The decimal portion is multiplied by 60 and the integer portion of the resultant is minutes. Seconds is calculated in a similar fashion. In the example given here, the date/time value appears eventually as 2001-2-25 12:09:59.

3.2.2 Ingestion

The Level 3 MODIS data that has to be ingested into the database, is in HDF format as mentioned earlier. This data format cannot be unpacked easily by knowing byte ordering, word locations etc. These files can be accessed only using HDF library subroutines and function calls from FORTRAN or C. This ingestion system processes the satellite data files using HDFv.4.2 ‘C’ libraries developed by NCSA. Using these library functions, HDF files are read, metadata is extracted and is dumped into a text file. This metadata consists of parameters that describe the contents of the file. Some of the parameters are listed below.

- Product Name
- File Location
- Latitude
- Longitude
- Date Sampled
- Processing Time
- Algorithm ID used to process the file

Integration of C code with Java

The program written in C to extract the metadata had to be integrated into our ingestion tool, which is a Java application. This was done using JNI (Java Native Interface) programming. Here are the steps involved in implementing this under JDK.

1. Declaration of native method in a class.

```
public class NativeModis
{

       public native void readModis(String fileName);
       ...................
 }
```

The native keyword alerts the compiler that the method will be defined externally.

2. Creation of C header file, NativeModis.h using javah utility.

```
   #ifndef _Included_NativeModis

...................

extern "C" {
#endif
/* Class:      NativeModis
 * Method:     readModis
 * Signature: (Ljava/lang/String;)V
 */
JNIEXPORT void JNICALL Java_NativeModis_readModis
  (JNIEnv *, jobject, jstring);

#ifdef __cplusplus
}
..................
```

3. Code implementation using function prototype from the header file.

```
JNIEXPORT void JNICALL
Java_NativeModis_readModis (JNIEnv *env, jobject obj, jstring
                                  fileName)
{
  C code goes here
      ..................

}
```

4. Compilation of C code into dynamically loaded library.

5. Addition of System.loadLibrary method to ensure that virtual machine will load the library prior to the first use of the class.

```
public class NativeModis
{
      ..................

  static
          {
              System.loadLibrary("modis");
      }
      ..................
}
```

Image Creation

JPEG images were created from the HDF files using Matlab version 6 software. For every HDF file representing mean, three jpeg images are created. The first one is the biggest image file (image.big.jpg), the second one represents an image which is smaller than the original one for browsing purpose (image.jpg) and the third one is very small image which acts like a thumbnail image (image.tn.jpg).

Here are a few commands along with their description, used in Matlab in creation of image files:

```
eppathnew='/home/modis/modisk1/enviprogs/';
coastfile=[eppathnew 'worldcoast.dat'];
eval(['load ' coastfile]);
```

- 'eval' function evaluates the expression enclosed in brackets. 'load' command imports ACII data file into workspace by reading its contents into a variable with the same name as the file.

```
[data, slope, intercept, equation, name, nits] =
readMODISHDF([indatapath file]);
```

- This statement is for getting data from the file.

```
[matlabeqn]=GUIscaleeqn(equation);
eval(matlabeqn);
finaldata=newdata;
```

- Here the data is scaled using scaling equation.

```
figure(99)
   set(gcf,'Visible','off')
   axesm eqdcylin
   colormap(njet)
   setm(gca,'maplonlimit',[-180 180],'maplatlimit',[-90 90])
..................

plotm(worldcoast(:,2),worldcoast(:,1),'LineStyle','-'
,'Color','w','Clipping','on');
   drawnow;
..................

eval(['print -djpeg70 -r70 ' imagepath file(1:end-4) '.jpg']);
```

- This piece of code is for making an image. The function 'figure' creates a figure graphics object and makes it visible by raising it above all the figures on the screen. 'axesm' creates a map axes. 'colormap' sets the colors used in creating the image. 'setm' sets multiple properties at the same time. 'plotm' displays projected line objects on the current map axes. 'drawnow' command completes pending drawing events.

```
eval(['print -djpeg70 -r150 ' imagepath file(1:end-4) '.big.jpg']);
clear file finaldata ilims name h;
```

➢ The 'eval' method uses 'print' command to save the image as 'imagename.big.jpg'. 'clear' commands clears the variables .

JDBC Operations

For making connection to a database, the corresponding database driver class has to be loaded. Once the driver is loaded, it generates its own instance and registers that instance with the 'DriverManager'. Then the DriverManager creates a 'Connection' object. The function 'getConnection' takes the database URL (Universal Resource Locator), username and password as its parameters and returns Connection object. The following piece of code shows how this is implemented.

```
Class.forName("com.merant.sequelink.jdbc.SequeLinkDriver").newInstance();
Connection connect =
DriverManager.getConnection("jdbc:sequelink://SUGAR:19996;databaseName=
                                EOS_DB", "kalinati", "*****");
 ..................
Connect.close();
```

Once the connection to the database is obtained, a 'Statement' object is constructed to provide a workspace to create an SQL query. Then, "Insert" statement is created and passed to the 'executeUpdate' statement which executes it.

```
Statement stmt = conn.createStatement();
String insertstmt = "INSERT INTO ......";
stmt.executeUpdate(insertstmt);
```

As one of the measures to decrease the execution time, 'AutoCommit' feature is used in this application. Normally, the connections to the database are set to "AutoCommit' mode by default, causing every transaction to be committed. This increases the execution time. SQL Server maintains a transaction log for each database to recover transactions if required in the future. A transaction log is a serial record of all changes that have occurred in the database along with the transaction that caused the change. In our case, every transaction is committed as soon as it is written into the log file. And once a transaction is committed, it is written to the disk, which is a bottleneck in terms of execution time. To overcome this, we could commit a batch of transactions rather than every single transaction. JDBC has a feature that allows the 'AutoCommit' mode to be set to true or false. This has been used in our application to achieve better performance. The following code snippet shows how this has been achieved.

```
connect.setAutoCommit(false);
 .................. (database update operations)
connect.commit();
connect.setAutoCommit(true);
```

Database Schema

The database schema has been developed based on the data file formats. It is designed in such a way that it can accommodate data from all existing sensors and has a provision for including data from new sensors in future as well. To identify the measurements, data regarding each file needed to be stored . Hence along with the measurements, the metadata information about the file is stored too. Since we are concerned with data collected by the sensor MODIS, our discussion will be centered around the table "Modis" (named after the name of the sensor that collects the data ingested into it) and "Sat_Images". Sat_Images is the main table that contains identification information of the file, like the date and time it was sampled, sensor that collected it, satellite on which the sensor was loaded etc.

The table below represents the existing database schema for the table Sat_Images.

Column Name	**Datatype**
Image_Name	varchar
Date_Sampled	datetime

East_Longitude	float
West_Longitude	float
North_Latitude	float
South_Latitude	float
Version	varchar
Satellite	varchar
Instrument	varchar
Parameter	varchar
Spatial_Resolution	varchar
Image_Size	int
Image_Type	varchar
Image_Path	varchar
Satellite_Data_Size	int
Satellite_Data_Type	varchar
Satellite_Data_Path	varchar
Year	int
Month	smallint
Day	smallint
Hour	smallint
Temporal_Resolution	varchar

"Image_Name" is the primary key, which identifies each sensor data row in the table shown above uniquely.

Each sensor has its own set of parameters which have to be archived. Hence a separate table schema, Modis, was designed to accommodate data pertaining to the MODIS data set alone. Schema for this table is shown below.

Column Name	**Datatype**
Image_Name	varchar

Product_Source	varchar
Product_Level	varchar
Met_File_Path	varchar
Met_File_Type	varchar
Met_File_Size	int
Processing_Time	datetime
Thumbnail_Path	varchar
Thumbnail_Type	varchar
Thumbnail_Size	int
Browse_Image_Path	varchar
Browse_Image_Type	varchar
Browse_Image_Size	int
Product_Status	varchar
Swath_Image_Path	varchar
Swath_Image_Type	varchar
Swath_Image_Size	int
Swath_Thumbnail_Path	varchar
Swath_Thumbnail_Type	varchar
Swath_Thumbnail_Size	int
Swath_Browse_Path	varchar
Swath_Browse_Type	varchar
Swath_Browse_Size	int
Image_Root_Name	varchar

This table is joined to the main satellite table using the "Image_Name" as the foreign key.

The Entity Relationship (ER) diagram (Fig. 3) shown below represents the relationship between the tables Sat_Images and Modis. The Modis table is a weak entity, which means that it cannot exist without the main Sat_Images table. There is a one-to-one relationship between these two tables meaning, for every entry in the Sat_Images table, there is an entry in the Modis table as well.

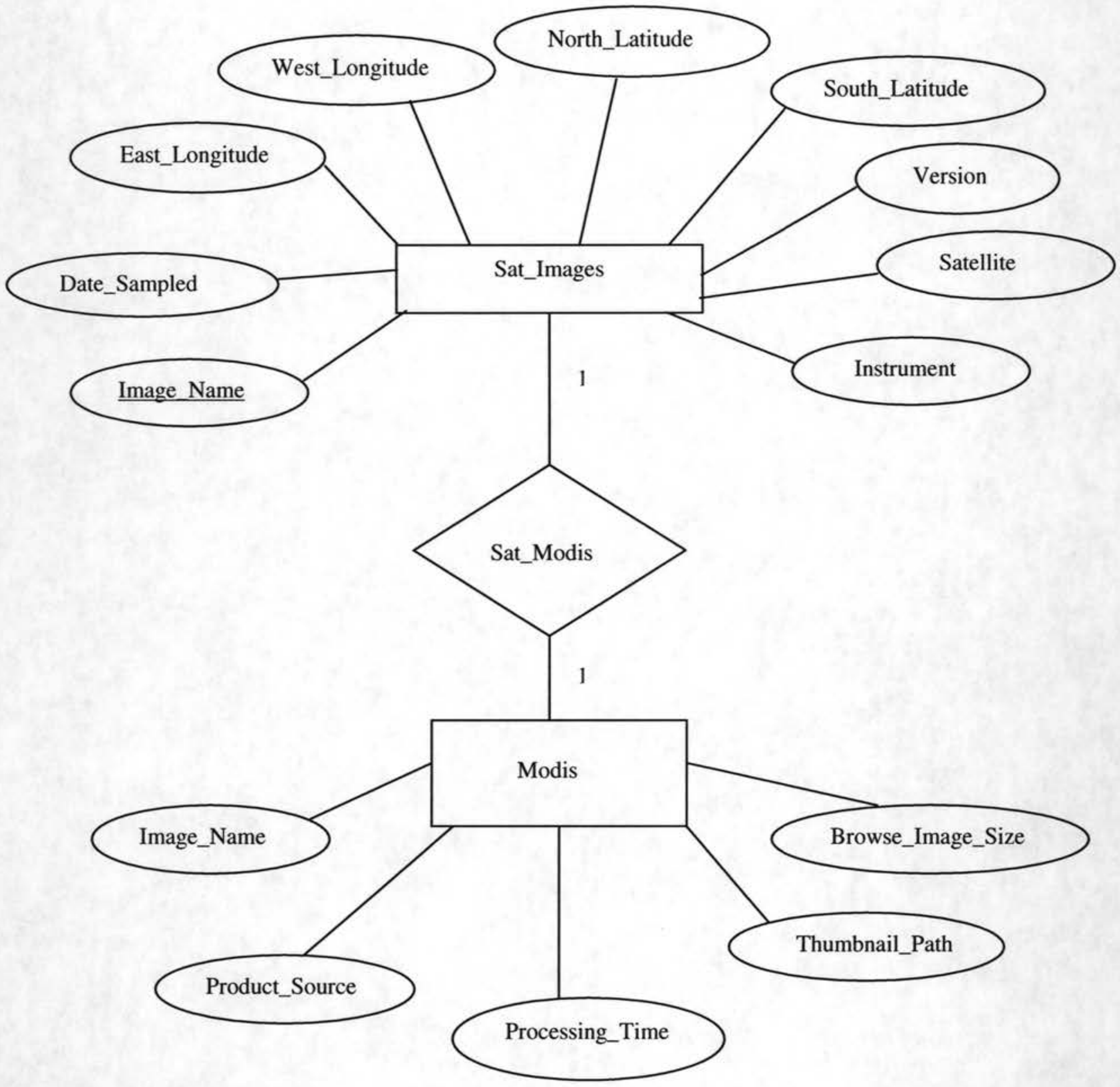

Figure 3: ER Diagram

Ingestion Sequence

The ingestion process takes place in a series of steps that include execution of several script files and Java files. This is illustrated in the sequence diagram (Fig. 4) shown below. It shows how the different classes in the application interact among themselves to accomplish the task of satellite data ingestion.

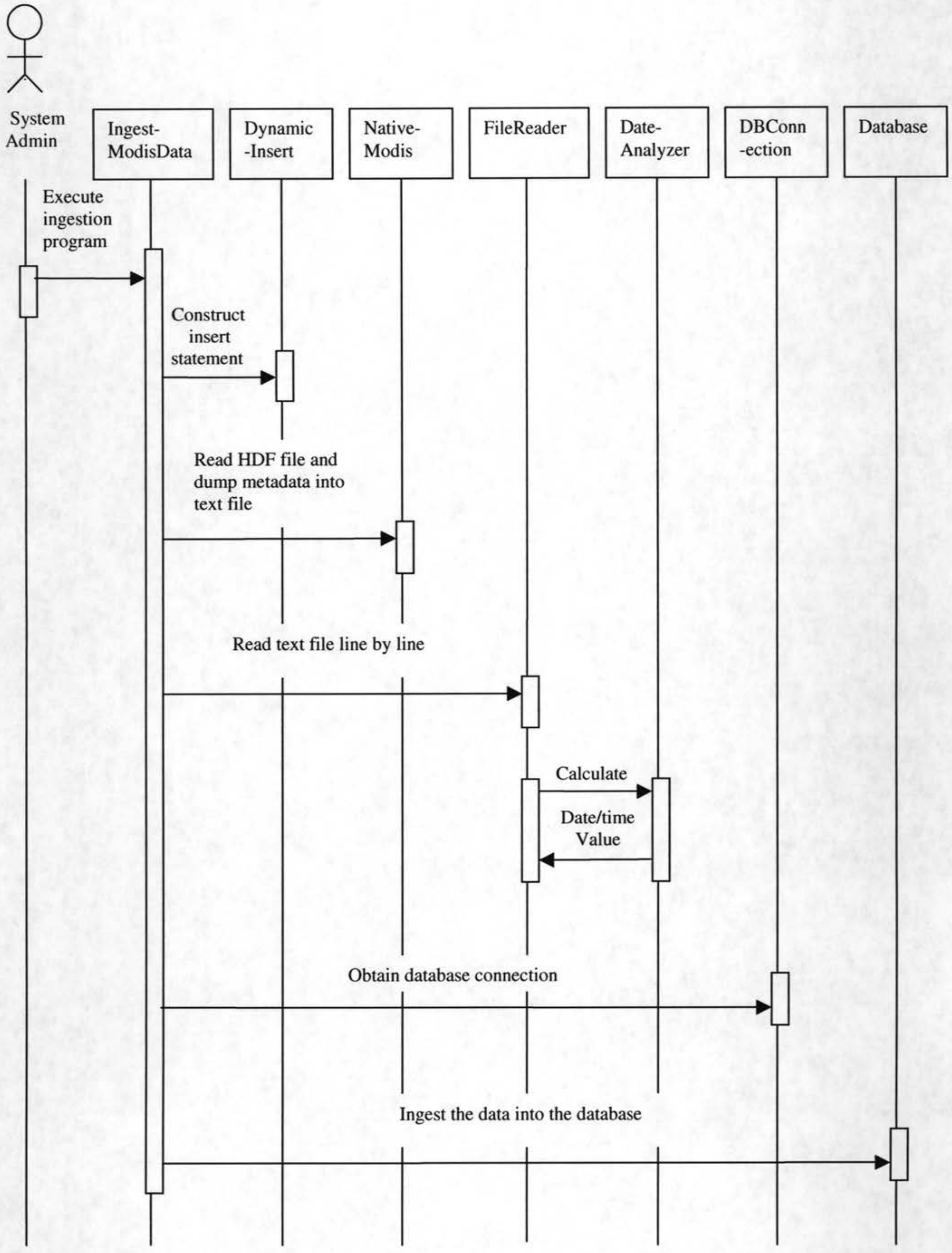
System Admin
Ingest-ModisData
Dynamic-Insert
Native-Modis
FileReader
Date-Analyzer
DBConn-ection
Database
Execute ingestion program
Construct insert statement
Read HDF file and dump metadata into text file
Read text file line by line
Calculate
Date/time Value
Obtain database connection
Ingest the data into the database

Figure 4: Sequence Diagram

Once the data arrives in the source directory, images are created using a shell script that runs the Matlab program. Then as the next step, the Java application is executed, which extracts the metadata from the HDF file and dumps it into a text file. This text file is then read line by line and the values are ingested into the data base along with the locations of data files and image files.

Description of the classes shown in sequence diagram is as follows:

- *DBConnection:* This class is responsible for opening and closing the database connections. It loads the appropriate database driver and creates the connection object that could be used by other classes to communicate to the database.

- *NativeModis:* This class opens an HDF file and reads it using C routines and dumps the required metadata into a text file. JNI technology is implemented in this class.

- *DynamicInsert:* This class creates *Insert* statements dynamically based on a database table definition and initializes columns. The class also provides interfaces to represent the column names and data types of columns of the table as required by the SQL server.

- *FileReader:* This class is responsible for reading the text file line by line and formatting the values in such a way that they are in accordance with the column format in the tables.

- *DateAnalyzer:* This class calculates the accurate date and time values from the metadata. This class had to be developed, as date/time values are not explicitly present and instead a represented by the values "Year", "DayofYear" and "Millisecond".

- *IngestModisData:* This is the main class that is finally responsible for data ingestion. It uses all the classes described above to ultimately archive the satellite data.

CHAPTER 4. CONCLUSION

The main goal of the EOSDIS project at College of Oceanic and Atmospheric Sciences, was to provide a web-based tool for data management and exploration by the scientific community. The satellite data ingestion tool, efficiently builds the database infrastructure by automating the data ingestion process, and lays a strong foundation for the web-based tool. It has been designed to operate under varying levels of load, including the real-time data gathered through the dish antenna recently erected by the college.

This tool efficiently taps several features in Java language that aid in developing scientific applications. Java Native Interface was one of them. HDF processing algorithms have been developed using Native 'C' libraries. This native code was integrated into the Java application using the JNI programming feature. Java Database Connectivity was another feature that was well utilized in this system. The built-in support for database operations in JDBC utility has simplified the task of building this data intensive database by many folds.

Some of the issues that this ingestion system tries to take care of are :

- Centralized database schema
- Sharing of data
- Extensibility to add new data sets
- Compatibility with the already existing database schema
- Effective archival of data products

4.1 Possible Enhancements

With a few enhancements, the current ingestion tool can be tuned to perform better. The following are the factors that could be considered for this purpose.

The Java Language

Interpreted languages are typically slower than compiled languages. This might pose as a disadvantage while using Java programming language, as the Java Virtual Machine interprets byte codes. But the presence of some built-in features in it make the development of scientific applications easier and overcomes this disadvantage described above. Moreover, database applications have other important factors such as client, server and query performance which can affect the performance more than the overhead from interpreting byte codes.

Database

There are a lot of performance issues while accessing databases. Poorly optimized queries, network latency, disk latency and difference between static SQL, dynamic SQL and stored procedures are some of them.

SQL update transactions are made individually in this project. But sending multiple update statements to the database at a time as a unit, is more efficient than sending each of them separately. A feature called *'batch updating'* present in JDBC 2.0 does the work of sending updates across the network in batches. By calling *addBatch function*, all the update statements can be grouped. And once this is done, the whole batch can be sent to the database for execution using *executeBatch* command, which will definitely reduce execution time.

This Java application connects to the database using JDBC driver manager. The JDBC driver that is used for creating a database connection, needs to be registered with the JDBC driver manager first. This is done using an appropriate static initializer with the driver class, using the following statement:

Class.forName("MyJDBCDriverName").newInstance();

In this application, "MyJDBCDriverName" is the SequeLink driver. This approach has a drawback. The fact that the JDBC driver class name identifies a particular driver vendor, renders it non-portable by making it specific to one product. More so, an application has to specify a JDBC URL while connecting to a database through the driver manager as shown below.

Connection connect = DriverManager.getConnection(dbURL,userName,passwd);

This makes application not only specific to the driver but to a particular machine as well. So, maintenance of the application becomes difficult as the computing environment changes. The Java Naming and Directory Interface (JNDI) eliminates this problem by allowing to specify a logical name to a particular data source, making deployment and management of the application easier. But it could not be implemented in the ingestion project, as the SequeLink database driver did not support this feature.

Image Creation

Currently, JPEG images are created from HDF files using a Matlab program. This program creates images using a few other files as input that contain information like, source/destination, HDF file paths, limits for color bar etc. Thus, the ingestion tool is dependent on other software for image creation. The future versions of NCSA HDF libraries are supposed to encompass this problem by possessing the capability to create images, removing the dependency on external software.

4.2 Future Work

Presently, real-time data is being collected at COAS using dish antenna. This raises the need for dynamic data processing and ingestion, due to the perpetual data flow. So, as the next phase, this project aims at automating the whole pipeline – conversion of analog satellite signals to digital format, production of various data products using a series of processing algorithm and ingestion of the resultant data into the database.

The arrival of data has to be monitored using a daemon process. Once the data arrives, an identity algorithm has to be executed, that identifies the category of the data set. Then, the corresponding processing algorithm needs to be triggered to process the data. And finally, the data should be ingested into the database. Work has to be initiated in the design of these daemons and their communication protocol.

REFERENCES

1. Cay S. Horstmann, Gary Cornell. Core Java 2, Volume I-Fundamentals, Prentice Hall 1999.

2. Cay S. Horstmann, Gary Cornell. Core Java 2, Volume II-Advanced Features, Prentice Hall 2000.

3. Ivor Horton. Beginning Java 2, Wrox Press 1999.

4. Seth white, Maydene Fisher, Rick Cattell, Graham Hamilton and Mark Hapner. JDBC API Tutorial and Reference, Second Edition. Addison-Wesley 1999.

5. Date.C.J, An Introduction to Database Systems, Sixth Edition, Addison-Wesley.

6. The NCSA HDF Home Page - http://hdf.ncsa.uiuc.edu/

7. MODIS Ocean Data Home Page - http://modis-ocean.gsfc.nasa.gov/