2000

Approaches to creating anonymous patient database

Shijun Shen
West Virginia University

APPROACHES TO

CREATING ANONYMOUS PATIENT DATABASE

Shijun Shen

Thesis submitted to the College of Engineering and Mineral Resources
at
West Virginia University
in partial fulfillment of the requirements for
the degree of

Master of Science
in
Computer Science

V.Jagannathan, Ph.D., Chair
Ramana Reddy, Ph.D.
Sumitra Reddy, Ph.D.

Department of Computer Science and Electrical Engineering
Morgantown, West Virginia
2000

APPROACHES TO

CREATING ANONYMOUS PATIENT DATABASE

Shijun Shen

ABSTRACT

Health care providers, health plans and health care clearinghouses collect patient medical data derived from their normal operations every day. These patient data can greatly benefit the health care organization if data mining techniques are applied upon these data sets. However, individual identifiable patient information needs to be protected in accordance with Health Insurance Portability and Accountability Act (HIPAA), and the quality of patient data also needs to be ensured in order for data mining tasks achieve accurate results. This thesis describes a patient data transformation system which transforms patient data into high quality and anonymous patient records that is suitable for data mining purposes.

This document discusses the underlying technologies, features implemented in the prototype, and the methodologies used in developing the software. The prototype emphasizes the patient privacy and quality of the patient data as well as software scalability and portability. Preliminary experience of its use is presented. A performance analysis of the system's behavior has also been done.

ACKNOWLEDGEMENT

I acknowledge with gratitude a large and continuing intellectual debt to my advisor Dr. V. Jagannathan. His expert guidance has been very helpful.

I would also like to thank the other members of my committee Dr Ramana Reddy and Dr. Sumitra Reddy for their support and review of several drafts of this thesis.

I am extremely grateful to the people at the Department of Computer Science and Electrical Engineering for having provided me with an opportunity to work with cutting edge technologies in the field of computer science.

-Shijun Shen

TABLE OF CONTENTS

1 INTRODUCTION

Outcomes measurement, a term used by the health care industry is a form of data mining in that it is based on past behavioral information which is used to improve the quality and efficiency of care for patients (Bresnahan, 1997). Outcomes measurement involves the examination of clinical information, insurance claims and billing data to gauge the results of previously used treatments and procedures.

A Norfolk, VA, HMO based health system, Sentara, used outcomes measurement to reduce its 12% mortality rate of pneumonia patients, as well as reduce the number of cases which developed complications requiring expensive antibiotics and extended hospital stays. Sentara's quality improvement team start exploratory data mining into claims which uncovered the high pneumonia mortality and complication rates. They found out later that doctors were ordering sputum cultures many times for a single patient in hope one test might return and yield useful, timely information. Meanwhile, the patient's illness progressed as physicians waited for the lab results which took several days. The quality team quickly developed a new system which allowed the transfer of the culture from the patient to the lab and lab results back to the physicians within two hours. Not only did the mortality rate for all pneumonia patients drop to 9%, but the average hospital stay decreased to one week and the cost to manage a single pneumonia case was reduced to $2000(Bresnahan, 1997).

Data Mining can benefit the health care industry in the following area:

- Identify which methods work and which don't in particular cases, allows them to recognize areas for improvement.

- Help health care providers cut costs and improve care by showing which treatments statistically have been effective.
- Identify people statistically at risk for certain ailments so that they can be treated before the condition escalates into something expensive and potentially fatal.
- Detect fraud in the health care system.

"Information has become the most valuable commodity in health care", according to Jeffrey C. Bauer, president of The Bauer Group, Inc., a Colorado-based consultant group. "In the past, studies have shown that as many as one-third of all medical interventions do not lead to an improvement in patients' conditions. In other words, about 33 cents on a dollar is spent on services that do not demonstrably make the patient better off. But today we can afford to provide only productive interventions. Outcomes data will finally allow us to weed out the resources that aren't making people better"(Bresnahan, 1997)

1.1 Problem Description

There is no doubt about the benefits of integrating data mining service into existing health care information systems. Certain factors, however, present problems. First of all, individual identifiable health information can not be protected if data mining analysis is directly applied upon the patient data collected. Any party who is performing data mining task over these data sets will be able to identify individual patient information. Secondly, quality of the patient data directly influence the outcome of the data mining analysis.

1.2 Objective

Most healthcare information management system generates data from its normal operations. If those patient data are transformed into "ready to be mined" data, the data

mining techniques can be directly applied upon those data sets and useful knowledge can be discovered. The goal of this thesis is to design a patient data transformation system which will provide "ready to be mined" patient data. "ready to be mined" patient data follows the following constraints. Patient data needs to be anonymous, which means no party who performs the data mining tasks should be able to draw any relation between the medical information provided with any individual patient. Yet at the same time, all entity relationships and business logic of the pre-processed patient records are preserved in the transformed patient records. A prototype of the system is developed as a component of CareFlow|Net's Careflow Development Kit (CDK), which is an intranet-enabled suite of middleware software engineered to support and integrate healthcare workflow processes. Patient records in CDK are XML based patient documents
(e.g., Patient Record Architecture). To summarize, the main objective of designing this system is:

- Protect individual identifiable information.
- Preserve the entity relationship of pre-processed patient data in transformed patient records.
- Preserve the business logic of pre-processed patient data in transformed patient records.
- Provide Logging and Error Recovery functionality.
- System is portable and scalable.

1.3 Methodology

The prototyping approach was used to design the system. Traditional approach like the water fall model or the life cycle approach could not be used since all

requirements could not be found in advance without testing the system and gathering more requirements from the users. Based on the feedback from the users, new features are implemented and the specification and design are modified accordingly.

The Objected Oriented (OO) approach was used to develop the software as it provided number of advantages over the traditional function-based approach. Applying a prototyping approach to function-based software would have been cumbersome; Moreover, function-based software does not lend itself easily to modification.

1.4 Preview of Chapters

The rest of this paper is organized as follows: Chapter 2 gives a literature review taken up to develop the prototype. Chapter 3 gives a functional overview of the prototype. Chapter 4 discusses the detailed design of the system. Chapter 5 describes the implementation issues. Chapter 6 discusses the testing/analysis of the system, also conclusion and future works are included in that Chapter.

2 BACKGROUND

2.1 Knowledge Discovery in Database(KDD) and Data Mining

I will use in this paper the definition given by Fayyad (Fayyad, 1996) to differentiate between KDD and Data Mining. According to Fayyad, KDD refers to the overall process of extracting high-level knowledge from low-level data, while Data Mining is just one step of the KDD process. "Knowledge discovery in databases is the non trivial process of identifying valid, novel, potentially useful, and ultimately understandable pattern or model in data". defined in Fayyad (Fayyad, 1996). Although general, this definition emphasize that KDD is a process that always involves experimentation, iteration, user interaction.

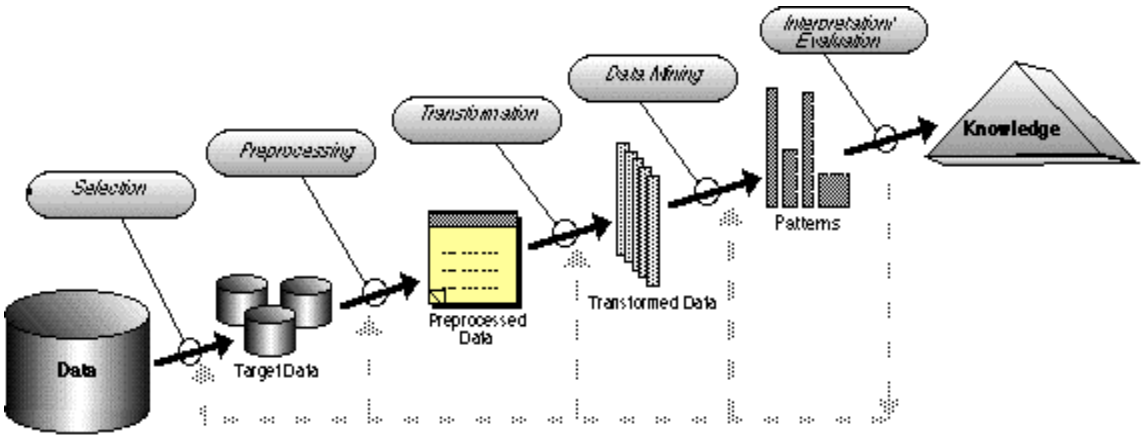

Figure 1. Overview of the KDD Process(Fayyad ,1996)

The following steps are usually followed in KDD. These steps are iterative, with the process moving backward whenever needed.

1. Develop an understanding of the application, relevant prior knowledge, and the end user's goals.
2. Create a target data set to be used for discovery.

3. Clean and preprocess data (including handling missing data fields, noise in the data, accounting for time series and known changes).
4. Reduce the number of variables and find invariant representations of data if possible.
5. Choose the data mining task (classification, regression, clustering, etc.)
6. Choose the data mining algorithm.
7. Search for patterns of interest in a particular representational form or a set of such representations.
8. Interpret the pattern mined, translating the useful patterns into terms understandable by users. If necessary, iterate through any of steps 1 through 7.
9. Using discovered knowledge.

The first two steps, understanding the application and determining the target data to be used for discovery are common to all forms of data analysis. This step includes knowing what relevant prior knowledge exists, the goals of the project, and what data exists. If a data warehouse is available, its metadata helps in pinpointing what data exists. The third step, cleaning and preprocessing the data is the usual one used in preparing data for warehousing. Basic operation includes removing noise or outliners if appropriate, collecting necessary information to model or account for noise, deciding on strategies for handling missing data fields, and accounting for time sequence information and known changes. The fourth step, data reduction and projection includes finding useful features to represent the data, depending on the goal, and using unidimensional reduction or transformation methods to reduce the effective number of variables under consideration

or to find invariant representations for the data. Because with too many variables, it is difficult to make any sense out of the result. The fifth, sixth and seventh steps are the heart of the KDD process, they are the data mining process. The fifth step choose the data mining task. Data mining breaks into four major categories: clustering, classifying, estimating and predicting, and affinity grouping.

Clustering:

Clustering is a pure example of undirected data mining, where the user has no specific agenda and hopes that the data mining tool will reveal some meaningful structure. For example, define new market segments.

Classifying:

An example of classifying is to examine a candidate customer and assign that customer to a predetermined cluster or classification. Another example of classifying is medical diagnosis. A classification is a decision.

Estimating and predicting:

Estimating and predicting are two similar activities that normally yield a numerical measure as the result. For example, we may find a set of existing customers who have the same profile as a candidate customer. From the set of existing customers we may estimate the overall indebtedness of the candidate customer. Prediction is the same as estimation except that we are trying to determine a result that will occur in the future, e.g. Stock market fluctuation. Estimation and prediction can also drive classification.

Affinity Grouping:

Affinity grouping is a special kind of clustering that identifies events or transactions that occur simultaneously. A well-known example of affinity grouping is market basket analysis. Market basket analysis attempts to understand what items are sold together at the same time. This is a hard problem from a data processing point of view and in a typical retail environment there are thousands of items sold together because the list quickly reaches astronomical proportions. The art of market basket analysis is to find the meaningful combinations of different levels in the item hierarchy that are sold together. For instance, it may be meaningful to discover that individual item Coca 12 oz is very frequently sold with the category of Frozen Pasta Dinners.

Step six and seven, choose data mining algorithm and searching for patterns. In these steps the actual mining is done. The algorithm selected solely depends on the data mining task to be performed. The following table shows what data mining algorithms can be used for which category of data mining.

Category of Data Mining	Corresponding Data Mining Algorithms
Clustering	Statistics Memory-based reasoning Neural networks Decision trees
Classifying	Statistics Memory-based reasoning Genetic algorithms Link analysis

	Decision trees Neural networks
Estimating and Predicting	Statistics Neural networks for numerical variables Algorithms described for classifying when predicting only a discrete outcome
Affinity Grouping	Statistics, Memory-based reasoning Link analysis Special purpose market basket analysis tools

Table 1 Overview of Data Mining Algorithms

The eighth step introduces the human back into the picture. The results of the data mining operation of step 7 are examined by an analyst who judges whether the outcomes are possible, internally consistent, plausible. Possible means that result is physically possible (e.g., doesn't exceed the speed of light). Internally consistent implies the result doesn't contradict itself. Plausible implies that the association found is believable, it could actually happen. If the analyst is unsatisfied with the result, he/she can returns the case with refined queries and conditions. That is this process is iterative. The analyst can repeat the above procedure until he/she is satisfied with the outcome. Then step nine will take over, presenting the result in a more understandable form to the decision maker. The form of representations of the result can be graphical diagram, natural language, etc. Finally, actions are taken based on the findings. And this is a complete KDD process.

2.2 HIPAA Patient Privacy Rules

The Health Insurance Portability and Accountability Act ("HIPAA") is a federal law with provisions that aim to protect patient privacy and confidentiality from the ease of communications through technology. The Department of Health and Human Services("HHS") drafted the proposed rules. The rules are authorized by HIPAA. HIPAA decreed that if congress failed to adopt patient privacy legislation by August 1999, then HHS must propose administrative rules for the same purpose, which HHS did in November 1999. Originally scheduled to be become effective February 21, 2000, the comment period was extended to February 17, 2000, and the rules' effective date is expected to be at least 60 days thereafter.

The proposed federal rules will impose demanding new duties on health care providers and health plans to protect the privacy of patient information. Stiff monetary and criminal penalties are authorized for violation of the rules and related statues. Compliance will require operational and administrative changes as varied as accounting to patients for disclosures of protected information and training employees with access to patient records. Larger and more sophisticated entities will be expected to monitor disclosures carefully to ensure that only necessary information is used and released. Policies and procedures, release authorization forms and contracts with outside entities will have to be revised. These and numerous other regulatory demands must be fully met within two years from the date the proposed rules become effective.

The proposed rules reflect the fear that increasing use of computers for storing and sending patient information will undermine patient confidentiality. The proposed rules, however, apply to individually identifiable information in any form once it has been maintained or transmitted by computer. As a practical matter, therefore, the

proposed rules will create a federal floor for the protection of all individually identifiable patient information. The proposed rules are expected to become final sometime spring, 2000. Given their breadth and complexity, and the civil and criminal sanctions authorized by HIPAA, two years will not be too much time to come into compliance. Compliance planning and implementation should begin as soon as possible. The following contents will highlight the principal features of the proposed rules.

Application

Covered Entities: The rules apply to health care providers, health plans and health care clearinghouse. "Health care provider" is broadly defined to include health care facilities, licensed practitioners, and suppliers of health care services or supplies.

Protected Information: The rules protect individually identifiable health information that is or has been maintained or transmitted by computer. The rules therefore extend to individually identifiable health information in any form once it has been maintained or transmitted by computer. This protection 'looks' forward and backwards: printouts of computer files are protected, as are paper records supplying the information from which computer files are created. Note that information need not be labeled with an individual's name to be identifiable: it is sufficient that the information, either alone or in combination with other information, permits an individual's identity to be determined. Note also that when state law allows minors to give informed consent, minors have the same privacy rights as adults if state law does not otherwise permit access by parents or others to their health information.

Relationship between state and federal law: HIPAA preempts state law that is less protective of patient privacy and preserves state law that is more protective. The result will be a patchwork quilt of federal and state regulations, with HIPAA setting a federal floor and state laws adding additional layers of protection. State laws specifically addressed to sensitive records, such as HIV test information and mental health records, will continue to be effective.

Authorization of Use and Disclosure

Prohibited authorizations: The proposed rules prohibit covered entities from obtaining patient authorization for uses or disclosures related to treatment, payment and "health care operations" such as quality assurance, premium rating and other health care-related purposes. Reasoning that these *pro forma* authorizations provide little privacy protection but may delay care or authorize inappropriate access to patient information, the rules actually proscribe the most routine instances of patient authorization.

Authorization required: The general rule is that patient authorization is required for any use or disclosure of protected information unrelated to treatment, payment or health care operations. Even disclosures within a covered entity must be authorized if the department receiving protected information is not engaged in health care delivery.

Exceptions to authorization requirement:

Unauthorized use and disclosure of protected information is permitted on certain public policy grounds: for law enforcement, research and public health purposes. Patient authorization is generally not required to respond to subpoenas and other

forms of legal process, although a court order will be required to obtain information concerning an individual who is not a party to litigation. Use and disclosure is generally permitted for banking and payment processes, emergencies, government data collection, and facility directories and for communications with family and close friends.

Business Partners

Contractual commitments: Covered entities may not disclose protected information to "business partners" rendering services to them or on their behalf without first obtaining contractual commitments safeguarding patient privacy.

Vicarious liability: Covered entities are liable for violations of patient privacy by business partners if they knew or reasonably should have known of the violation and failed to stop or mitigate the effects of the violation. The rules accordingly require covered entities to implement safeguards to ensure compliance by business partners.

Patient Rights

Notice: Upon request, covered entities must provide patients with a notice of their rights under the law and the entities' relevant policies and procedures.

Access: Within 30 days of a request, patients must be allowed to review, inspect and copy "designated record sets' kept by covered entities or their business partners.

Accounting for disclosures: Within 30 days of a request, covered entities and business partners must produce an accounting of disclosures unrelated to treatment, payment or health care operations.

Corrections and amendment: Individuals may request health care providers and health plans to correct or amend protected information contained in designated record sets. Records should be corrected or amended in such a manner that the original record is preserved and the date and author of the correction or amendment is noted.

Minimum use and disclosure: Covered entities may use or disclose only the minimum amount of protected information necessary to achieve the purpose of the use or disclosure.

Administrative Requirements

Designation of Responsible Personnel: The rules require the designation of a "privacy official" charged with developing and implementing the covered entity's privacy policies and procedures. A "contact person" of office also must be designated to receive complaints and to provide information on policies and procedures.

Training: All employees likely to have access to protected information must be trained in the privacy policies and procedure relevant to their job duties.

Safeguards: Appropriate technical, administrative and physical safeguards must be implemented to preserve the privacy of protected information.

Complaint Procedures: Procedures for the filling, recording, processing and preservation of complaints must be developed. Sanctions must be authorized and applied to personnel violating the rules or the entity's own policies and procedures.

Documentation of compliance: Covered entities must adequately document the policies and procedures assuring compliance with the proposed rules.

Some Areas of Special Concern

Private rights of action: HIPAA does not create a federal cause of action for violation of the proposed rules. However, it will create enforceable contract rights by requiring the patients and enrollees be designated as third party beneficiaries of contracts between covered entities and their business partners.

Minors: Minors' privacy rights may trump parental interests in access to their children's health information. Where minors have the power of informed consent but state law is silent regarding the rights of parents or others to access minors' health information, minor have the same rights as adults to control access.

Sanctions: HIPAA authorizes both civil and criminal penalties for privacy violations. Civil penalties of up to $25000 per privacy standard are authorized annually. These civil fines don't require any wrongful intent, an unknowing violation of the rules is sufficient. Given the rules' complexity, unknowing violations are not unlikely. Given the Criminal sanctions, of course, do require wrongful intent, but the sanctions are potentially devastating: fines ranging from $50000 to $250000 per violation and prison terms of up to ten years.

One interesting concept is that many practical problems disappear once patient information is no longer identifiable with an individual. How can we "de-identify" patient information upon receipt and to restore identifying information at appropriate junctures in the treatment and billing process?

2.3 Random Number Generation

The only true sources for (partially) random numbers involve measuring physical phenomena, such as the timing of radioactive decay, which can be distilled into purely random sequences using some mathematical tricks. Without access to physical devices, computer programs that need random numbers are forced to generate the numbers themselves. But the determinism of computers makes this algorithmically quite difficult. As a result, most programmers turn to pseudo-random numbers. Now, let's look at pseudo-random number generators (PRNG).

The most common type of a PRNG is the linear congruential random number generator. Begins with an initial value or "seed" r_0. Each successive random number r_{i+1} is generated by

$$r_{i+1} = (a \times r_i + b) \bmod n$$

where a, b and n are constants. Often, n is chosen as 1 more than the maximum number that can be stored in a computer world, so that this computation can be performed by discarding any portion of the intermediate result that exceeds storage. This generator produces random integers between 0 and n-1. The more bits of a seed you use, the harder the PRNG to break.

2.4 Extensible Markup Language

2.4.1 Overview

The Extensible Markup Language (XML) is a meta-markup language that provides a format for describing structured data. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language. It provides a structural representation of data that has proved broadly implementable and easy to deploy. XML is a subset of SGML that is optimized for delivery over the Internet; it is

defined by World Web Consortium (W3C), ensuring that structured data will be uniform and independent of application or vendors. It enabled a new generation of internet-based data viewing and manipulation application.

The power and beauty of XML is that it maintains the separation of the user interface from the structured data. XML uses markup tags as well, but, unlike HTML, XML tags describe the content, rather than the presentation of that content. By avoiding formatting tags in the data, but marking the meaning of the data itself with custom user definable tags, we actually make it easier to search various documents for a tag and view documents tailored to the preferences of the clients. This separation of data from presentation enables the seamless integration of data from diverse sources. Customer information, purchase order, medical records, and other information can be converted to XML on the middle tier, allowing data to be exchanged online easily. Data encoded in XML can then be delivered over the Internet to the client. No retrofitting is necessary for legacy information stored in mainframe databases; the client can just use Extensible Style Language (XSL) or Cascading Style Sheets (CSS) to view the data according to his/her preference.

2.4.2 XML Element

XML is a meta-markup language, a set of rules for creating semantic tags used to describe data. An XML element is made up of a start tag, an end tag, and data in between. The start and end tags describe the data within the tags, which is considered the value of the element. For example, the following XML element is a "FIRSTNAME" element with the value "John".

<FIRSTNAME>John</FIRSTNAME>

The element name "FIRSTNAME" allows you to mark up the value "John" semantically, so you can differentiate that particular bit of data from another, similar bit of data. For example, there might be another element with the value "John".

<LASTNAME>John</LASTNAME>

Because of each value is associated with a different tag. We can easily tell that first appearance of value “John” implies a person’s first name where second appearance of value “John” implies a person’s last name.

In addition, XML tags are case sensitive. <firstname> and <FIRSTNAME> are considered different tags in XML.

2.4.3 XML Document

A basic XML document is simply an XML element that can, but might not include nested XML elements.

2.4.4 Well formed XML Document vs. Valid XML Document

Document Type Definition (DTD) defines elements, attributes, and relationships between elements. When the XML document is processed, it will check against the DTD to be sure the document is constructed correctly and all tags are used in the proper manner. Once the XML document meets its DTD constraints, the document is considered as a valid XML document. Note a DTD is not required in order to create a XML document, it is only needed if one wants to validate his/her XML document. As for well formed XML document, it only needs to follow the XML syntax rules.

The above are only the basic of XML, a lot more functionality is provided by XML.

2.4.5 XML and Health Care

Healthcare information is, to a very large extent, exchanged and stored as unstructured or slightly structured text that can be processed and retrieved. XML opens a completely new perspective in document handling and processing and message transmission in healthcare.

Some usage of XML in Health Care arena include:

Patient Record Architecture (PRA)

The emerging HL7 PRA standard for encoding of patient record documents promises to provide a vendor-neutral, platform-independent means of exchanging clinical healthcare information from a wide variety of sources. The scope of PRA is the representation of clinical documents in a patient record, and a PRA document is the basic unit of a document oriented electronic patient record. Some key aspects of PRA include:

- PRA documents are encoded in XML.
- PRA documents derive their meaning from the Health Level 7 (HL7) Reference Information Model (RIM) and use RIM data types.

The PRA is receiving significant interest from healthcare software and was a fundamental feature of a prototype interoperability demo at Healthcare Information Management Systems Society 1999 and 2000.

XML Data Warehousing for Browser-based Electronic Health Records

Patient records are created and maintained as XML documents within an XML database.

XML as a Vehicle for the exchange of Electronic Patient Records

XML is used to wrap the data in an easy maintainable and accessible format and XSL provides mapping and presentation rules of the data in a distributed health care database.

2.5 XML parser

A XML parser is a program that converts XML documents into some object model. Once a XML document is parsed, it exists in the memory as a set of objects. Instead of manipulating the XML document directly, one can access and modify information stored in the XML document through the objects in memory. XML parser creates a document object representation of the XML document. This document object contains a tree of nodes, which represents the structural and content information of the XML document. This tree of nodes can be accessed and modified using the DOM API, which is created by W3C. However, the XML parser will have to provide the implementation of the DOM APIs. The entire conversion process is illustrated in Figure 2.

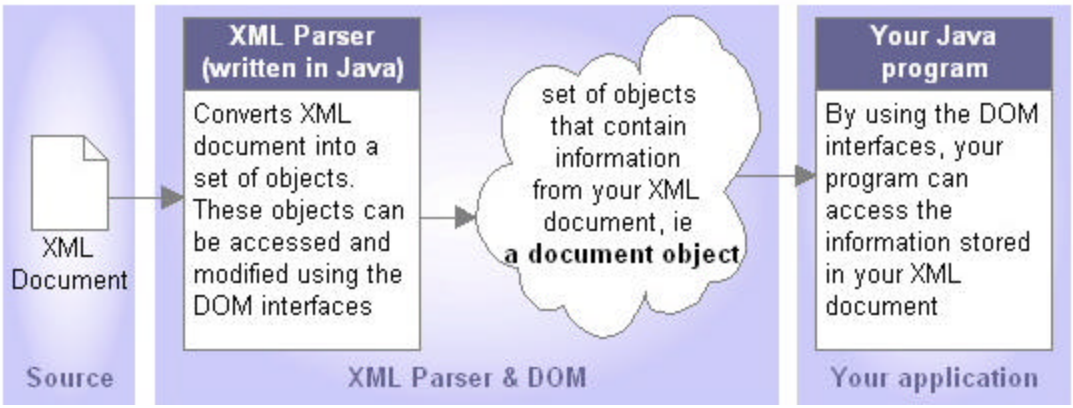

Figure 2. XML document to Document Object conversion process

Figure 3 shows the hierarchical structure of the XML document object

2.6 Document Object Model

Document Object Model (DOM), a platform and language neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. The Document Object Model provides a standard set of objects for representing HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs, thus increasing interoperability on the Web.

2.6.1 DOM API

The DOM API allows hierarchical access of the information stored in XML documents. Any XML parser implements DOM API needs to put all interfaces in org3.w3c.dom package. However, remember that the code to instantiate DOM object depends on the specific XML parser, all the other code can be completely DOM standards based and portable. If factory pattern is used to instantiate the objects that implement the DOM interfaces, then all the code are completely portable, no parser specific code will be embedded. In DOM, everything is a Node. The other interfaces are provided to make things more object-oriented. Figure 3 shows the inheritance relationship between some of the important interfaces.

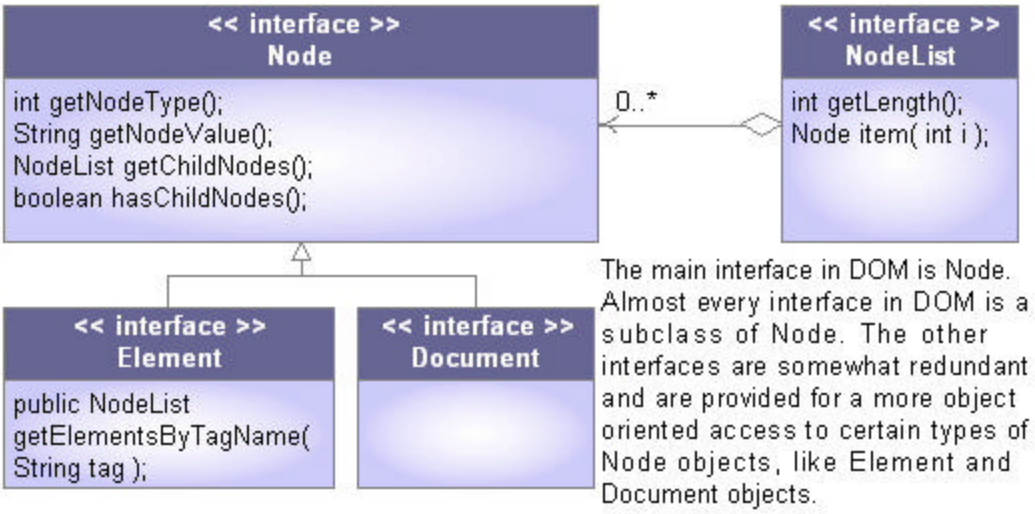

Figure 3 Inheritance relationship between important DOM interfaces

The root *Node* object of the document tree is also the *Document* object. *Document* is a subclass of *Node*. Every DOM object must have a root. Another important interface is the *Element* interface, which is a subclass of *Node*; the *Element* interface can be used to access the elements in a DOM Document object tree. The *Node* interface encapsulates access to a lot of information about a node. You can find out if a *Node* has children or not by calling *hasChildNodes()* method. The *getNodeType()* method returns the type of a *Node*; the type is just a constant integer that is used to identify types of Node. For an instance, *Node.ELEMENT_NODE* type identifies a *Node* to be an *Element*. *GetNodeValue()* method can be used to get the textual data stored inside a *Node*. The *Node* interface also has methods that allow the traversal of a Node tree. The *getChildNodes()* method is useful for getting all the elements inside a Node. This method returns all Nodes(if they exist) in an object that is a container for Node objects; the object implements the *NodeList* interface. *NodeList* is an iterator for a list of nodes.

XML document and a validating XML parser is needed in order to read information from a XML document into your programs. And DOM API is the tool used to access and modify the contents of the XML documents.

2.7 Java and JDBC

The JDBC API is a Java API for accessing virtually any kind of tabular data. As a point of interest, JDBC is trademarked name and is not an acronym; nevertheless, JDBC is often thought as standing for "Java Database Connectivity". The JDBC API consists of a set classes and interfaces written in the Java programming Language that provide a standard API for tool/database developers and makes it possible to write industrial strength database applications using all-Java API.

The value of the JDBC API is that an application can access virtually any data source and run on any platform with a Java Virtual Machine. In other words, with the JDBC API, it isn't necessary to write one program to access a Sybase database, another program to access an Oracle database, another program to access an IBM DB2 database, and so on. One can write a single program using the JDBC API, and the program will be able to send SQL or other statements to the appropriate data source. And with an application written in Java, one doesn't have to worry about writing different applications to run on different platforms.

2.8 Logging Package for Java

With Logging Package for Java (Log4J) it is possible to enable logging at run time without modifying the application binary. The Log4J package is designed so that these statements can remain in shipped code without incurring a heavy performance cost.

Logging behavior can be controlled by editing a configuration file, without touching the application binary.

Logging equips the developer with detailed context for application failures. On the other hand, testing provides quality assurance and confidence in the application. One of the distinctive features of log4j is the notion of inheritance in categories. Using a category hierarchy it is possible to control which log statements are output at arbitrarily fine granularity but also great ease. This helps reduce the volume of logged output and minimize the cost of logging.

The target of the log output can be a file, an “OutputStream”, a remote log4j sever, a remote Unix Syslog daemon or even a NT Event logger.

On a 233 MHZ ThinkPad running JDK 1.1.7Beta, it costs about 46 nano-seconds to determine if that statement should be logged or not. Actual logging is also quite fast, ranging from 79 u-seconds using the SimpleLayout, 164u-seconds using the TTCCLayout and around a milliseconds when printing exceptions. The performance of the PatternLayout is almost as good as the dedicated patterns, except that it is a lot more flexible.

3 SYSTEM OVERVIEW

3.1 System Requirements

The goal of this thesis is to develop a patient data transformation system which transforms XML based electronic patient records (e.g., PRA documents) into anonymous high quality patient data that is suitable for data mining purposes. The key requirements of the system are listed below.

- *Anonymousness of Patient Data*

Protect individual identifiable patient information in accordance with HIPPA. No party who performs the data mining tasks should be able to identify any health information with a specific patient.

- *Preserve the Entity Relationship of the pre-processed Patient Data*

As the input Patient data are transformed into anonymous patient records (e.g., patient ID numbers that won't make any sense), the entity relationships among input patient data needs to be preserved in the transformed anonymous patient records, so data mining over these data sets are possible.

- *Preserve the Business Logic of the pre-processed Patient Data*

All the properties of a patient record should be preserved except those properties, which needs to be altered in order to anonymize the patient record.

- *Logging and Error Recovery*

The history of the transformation process must be captured so any abnormal behavior will be spotted and appropriate error recovery procedure could be taken. This will further enhance the quality of the transformed patient records.

- *Portability and Scalability*

Integrating the transformation system into any healthcare information management system as it is. The system should be developed as a “plug and play” software component. Any further modification or enhancements can be made very easily on the current system if the software is component based. In other words, the transformation system should be scalable.

3.2 Software Architecture

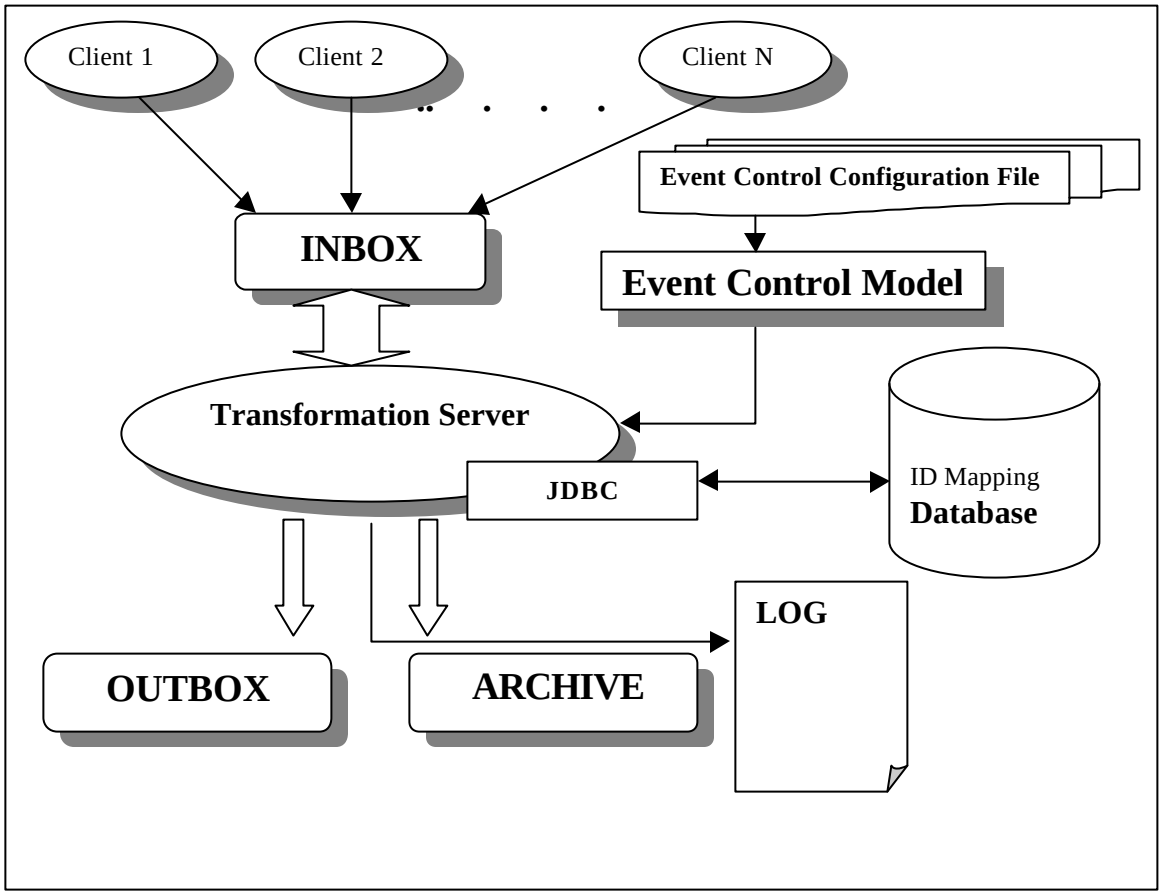

Figure 4 Software Architecture

Inbox

Inbox contains the pre-processed XML based patient records, which needs to be transformed.

Outbox

Outbox contains the transformed anonymous patient records which are also XML documents

Transformation Server

The component does all the transformation work, it is the core of the system. The transformation server checks the inbox every so often to see if new patient records have arrived. If so, it takes the new patient records and transforms them.

Event Control Configuration File (XML document)

Each group of patient documents (e.g., PRA documents with document type levelone) have a corresponding Event Control Configuration File. All the event control information is stored in this File. And it can be accessed via an Event Control Model Configuration GUI provided by the system.

Event Control Model

Event Control Model is an object indicates to the transformation server what actions needs to be taken to a particular patient record in order for the creation of the anonymous patient record. Each document type has its corresponding Event Control Model. Event Control Model is created on the fly based on Document type specific event control configuration file.

ID Mapping Database

ID Mapping Database stores some mapping information needed for the transformation process.

Clients, in the case of CDK, are different services of CDK that generates patient data and send them to the inbox. Transformation Server checks the inbox directory and

realize there are new patient records information to be processed, it grabs the patient records and transforms them based on event control model, dumps the anonymous patient records in the outbox directory. Copies of pre-processed patient data will be kept in archive directory. A log file is also generated to capture the history of the process and spot any abnormal behavior if possible. If any error occurred, the operator can check against the log file and figure out exactly what went wrong, get the pre-processed patient records that had process failures from the archive directory, retransform them. Therefore, no data loss will be encountered. The outbox is virtually an anonymous XML patient database, data mining techniques can then be applied upon this XML information pool.

3.3 Functional Overview of the Software System

3.3.1 Event Control Model Configuration

Event Control Model basically tells the transformation server the set of actions needs to be taken in order to successfully transform a specific patient record. A GUI is developed to ease the difficulty of creating, modifying Event Control Model. The underlying data of Event Control Model is the XML elements in Event Control configuration file.

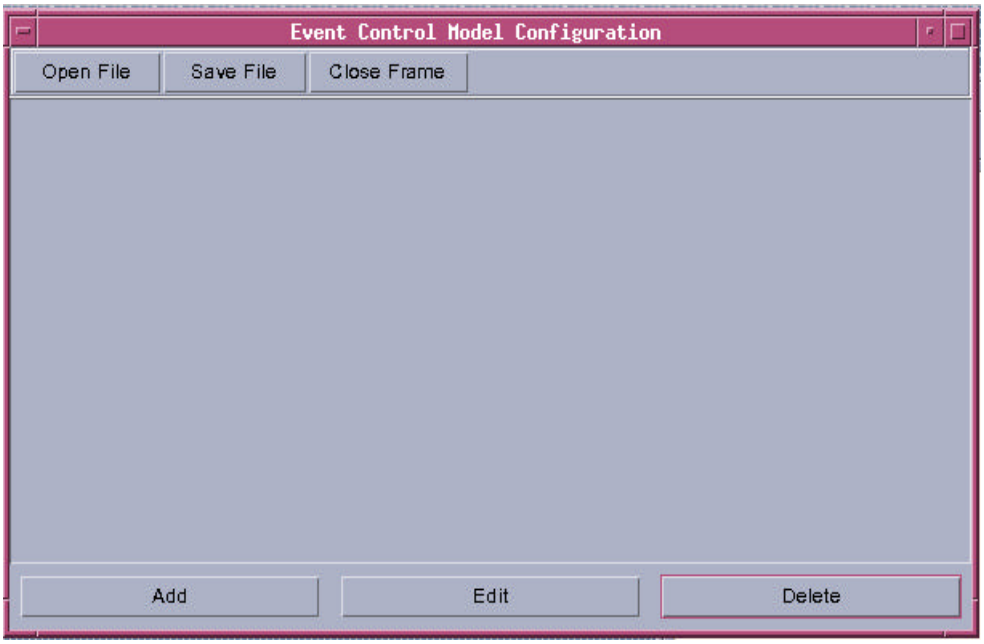

Figure 5 Event Control Model Configuration 1

Figure above illustrates the Event Control Model GUI. After operator clicks the "Open File" button. A dialog box will pop up and asks the operator for the Event Control configuration file location (as shown on Figure 6). The Event Control configuration file name are constructed in the following format: document type + "Config.xml". For an instance, the configuration file name for document type levelone of PRA documents will be "leveloneConfig.xml", the configuration file name for document type patientReport will be "patientReportConfig.xml". If the patient record doesn't specify a particular document type, "defaultConfig.xml" will be used as the default configuration file.

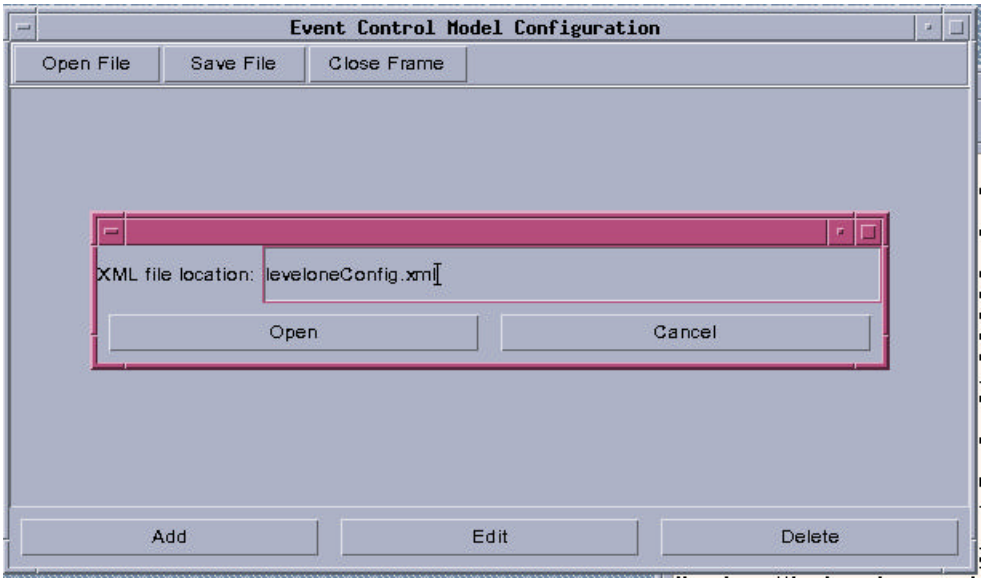

Figure 6 Event Control Model Configuration 2

The following shows the contents of "leveloneConfig.xml" and the DTD of Event Control configuration file.

```
<?xml version= '1.0'?>
<EventControl>

 <Item>
  <Scope>levelone</Scope>
  <TagName>id</TagName>
  <Attribute>V</Attribute>
  <Event>IDTranslator</Event>
 </Item>

 <Item>
  <Scope>levelone</Scope>
  <TagName>G</TagName>
  <Attribute>V</Attribute>
  <Event>HideField</Event>
 </Item>

 <Item>
  <Scope>actor</Scope>
  <TagName>F</TagName>
  <Attribute>V</Attribute>
  <Event>HideField</Event>
 </Item>

</EventControl>
```

LeveloneConfig.xml

```
<!ELEMENT EventControl (Item)>
<!ELEMENT Item (Scope, TagName, Attribute, Event)>
<!ELEMENT Scope (#PCDATA)>
<!ELEMENT TagName (#PCDATA)>
<!ELEMENT Attribute (#PCDATA)>
<!ELEMENT Event (#PCDATA)>
```

EventControl.dtd

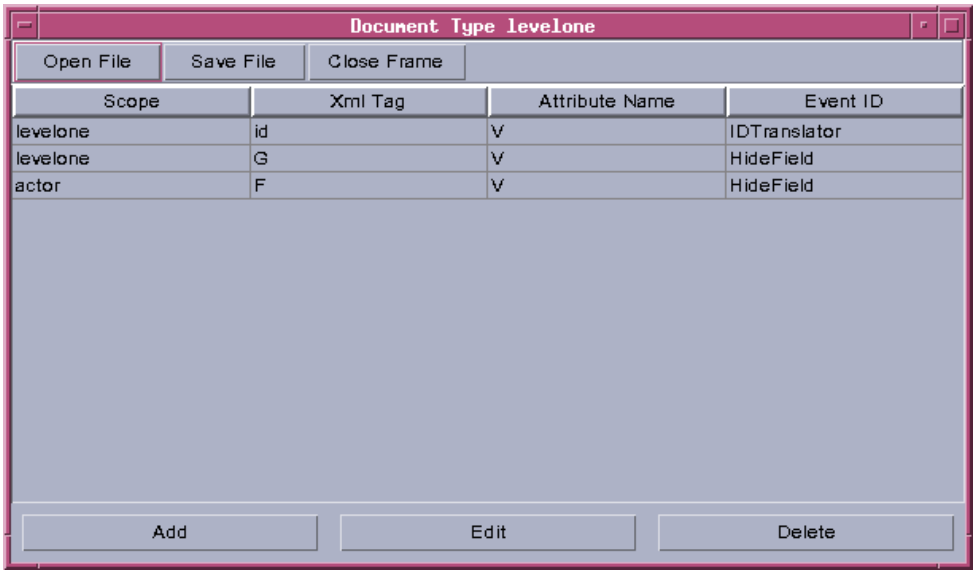

Figure 7 Event Control Model Configuration 3

As shown in Figure 7, the Event Control Model for document type levelone that is derived from "leveloneConfig.xml" is a table-structured data set. The "Xml Tag" and "Attribute Name" indicates the entities in the XML documents that need to be transformed. "Scope" column of the Event Control Model indicates in which part of XML document, the entities specified in "Xml Tag" and "Attribute Name" needs to be altered. If the operator wishes the transformation of certain entities taken effect over the whole XML document, the root element of that document should be given as the "Scope", otherwise, the "Scope" value needs to be assigned accordingly. "Event ID" indicates what action needs to be taken upon these entities. Event ID "IDTranslator" means encode the value of that entity (e.g., encode the patient ID), "HideField" means hide the value of the given entity (e.g., .hide patient name) For future work, more "Event ID" can be defined for enhanced capabilities (e.g., define Event ID "NoiseReduction" to be the indication of noise reduction on a given entity).

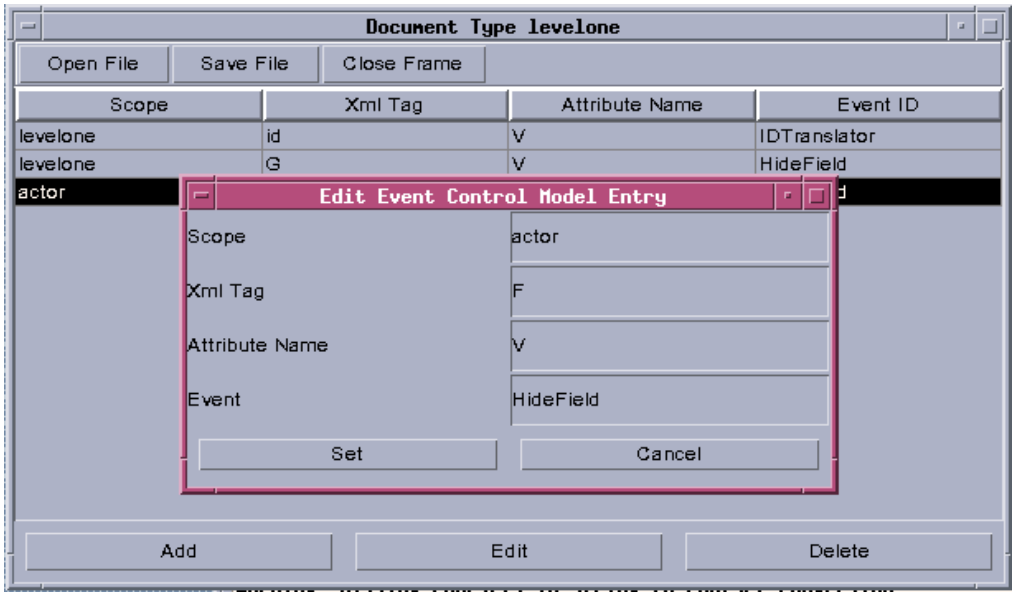

Figure 8 Event Control Model Configuration 4

Operator can modify the contents of Event Control Model by select the entity and click "Add", "Edit" or "Delete" button. Figure 8 shows the Edit function, after the operator select the entity with the "Xml Tag" valued 'F' and click "Edit" button, a dialog box pops up, modification on that entity then can be made. After operator finishes all the modification needed, click button "Save File" on the tool bar, all the modification will be saved into the corresponding configuration file. Click "Close Frame" to exit Event Control Configuration GUI.

To create a new Event Control configuration file, repeat the same step as described above. Upon exiting, click the "Save File" button on the tool bar, the system will create the configuration file automatically based on the information entered in the Event Control Model Configuration GUI.

3.3.2 Transformation Process

All the transformation process is done in the Transformation Server. The following chart illustrates the work flow of the transformation process.

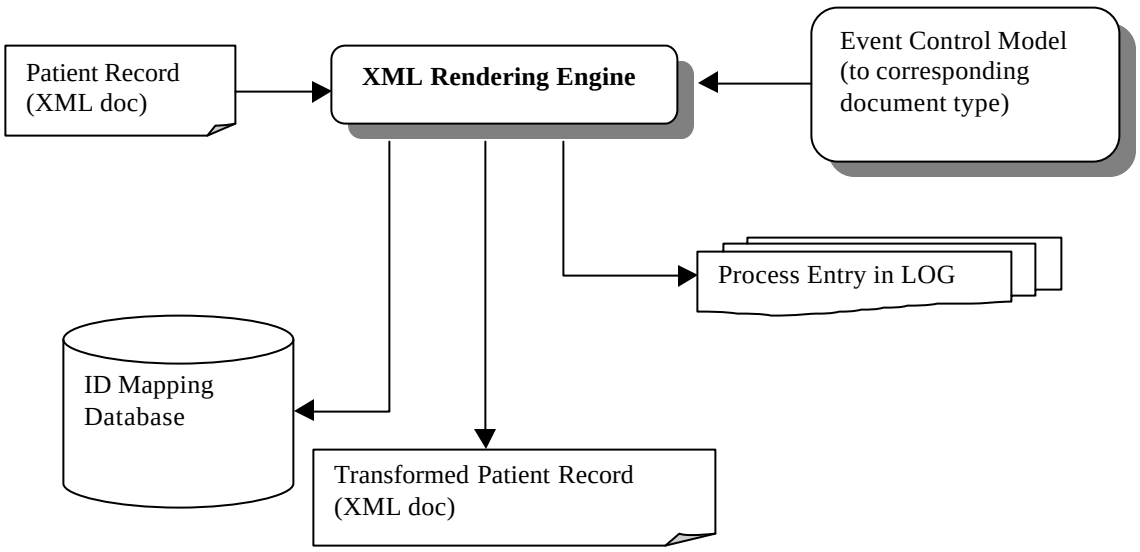

Figure 9 Transformation Process Work Flow Chart

A patient record is put into the XML Rendering Engine for document processing. The XML Rendering Engine will parse the patient record document and make necessary changes according to the Event Control Model, if ID encoding is required, the XML Rendering Engine will connect to the ID Mapping database through JDBC and retrieve the ID Mapping information it needs. Through out the whole transformation process, multiple log entries relating process behavior will be created and feed into the LOG file.

3.3.3 LOG and Error Recovery

Logging capability is provided by the system. A log file named "LOG" is created while the transformation server is started the first time in the system, and all the process history will be captured into the "LOG" file. Operator can decide what process history needs to be captured by modifying some of the parameters in the "LOG" configuration file at runtime. However, the corrupted processes are always reported into the "LOG" file for error recovery reasons. The operator is responsible to check the "LOG" file periodically for corrupted processes and take the necessary error recovery action.

Since all the input patient records are archived. Therefore, the operator simply needs to identify which patient records had process failures based on the "LOG" file, and put those patient records into Inbox to be transformed again. Also by analyzing the process history, useful information regarding the transformation process can be discovered. The following are some sample log entries.

```
2000-11-15 22:48:14,945 INFO [main] TransformationServer - Start transforming /usr08/shawny/thesis/Code/inbox/V2C.xml
2000-11-15 22:48:16,826 ERROR [main] IDTranslator - SQLException: ORA-01017: invalid username/password; logon denied

2000-11-15 23:12:49,798 INFO [main] TransformationServer - Start transforming /usr08/shawny/thesis/Code/inbox/V2C.xml
2000-11-15 23:12:56,512 INFO [main] XmlRenderingEng - /usr08/shawny/thesis/Code/inbox/V2C.xml transformation process finished
```

Figure 10 Sample Log Entries

4 DESIGN

This chapter discusses the various issues involved in the design of the Patient Data Transformation System. The design considerations, methodology and constraints are described in detail.

4.1 Design Approach to anonymize the patient record

4.1.1 Requirements Analysis

As one of the requirement states that no party should be able to associate any medical information with a specific patient. All electronic patient documents regarding a particular patient are associated via different ID, those meaningful ID numbers represent some aspect of a specific patient. For an instance, for patient “213233454” whose name is “John Smith”, his physician “phy-12345” whose name is “Jane Doe” created a report “10090” for him as event “00001” at facility “09890” (Radiology) on Dec, 18th, 1999. If those electronic patient records are provided as the data source for data mining analysis, the party who performs the data mining tasks will be able to identify the medical information provided with a specific patient which is direct violation of HIPAA. Therefore, a protocol needs to be developed so that all the information in a patient record or any combination of information from different patient records which could lead to possible individual identifiable patient information needs to be transformed into some pattern such that individual identifiable patient information could be protected. However, at the same time, all entity relationships among those patient records need to be preserved in order for possible data mining analysis.

Identifying Information that could lead to Possible Individual Identifiable Patient Information

First couple of things come to mind would be the patients' name, patients' address and phone number, these entities in a patient record is the direct indication of the patient's identity. Therefore, an operation needs to be provided to hide these entities.

A lot of patient data is provided as data source for the data mining tasks which means the data mining analyst would have access to all these patient data in order to perform the data mining task. If a set of patient records whose direct indications of identities (e.g., patient name, phone number) are all well hidden is provided as the data source, would this be good enough to protect individual identifiable information? Not quite, an in-depth analysis has been done which indicated that any ID in a patient record has the possibility becoming an indirect lead of an identification of a specific patient under different circumstances.

4.1.2 Design Approach

Hide Direct Indication of Patient's Identity

In a XML based patient record, in order to hide patient name and phone number, we simply needs to parse through the document and change the value of those entities to whatever dummy value we want them to be. However, with IDs, such approach would not succeed.

ID Encoding Schema

IDs are normally the entities in each patient record that establishes the entity relationship. If simply replace the ID value with some dummy value, those entity relationships among patient data would be broken. Therefore, the ID encoding schema must satisfy the following conditions:

1. There exist a one to one relationship between the ID and its encoded ID.

2. ID encoding should be a one way operation, meaning that reverse generation of the original ID from the encoded ID is impossible for unauthorized personnel.

4.1.3 Design of ID Encoding Schema

To realize the ID encoding schema described in section 4.1.2, a mapping table is first created which resides in a secured database.

Field Name	Data Type	Null?	Unique?
ID	VARCHAR2(20)	Not Null	Unique
RandomizedID	VARCHAR2(40)	Not Null	Unique

Table 2 "IDMapping" Table

In order to realize the one way ID encoding, we need to find an encryption algorithm that is efficient yet secure. The reason for efficiency is there are lot of ID encoding operations needed for each patient record, an efficient algorithm will be low cost in terms of computation time as we all know that encryption operations are expensive. In order for the reverse generation of the original ID becomes impossible. The original ID must not have any logical connection with the encoded ID, which implies that the original ID should never be used as the input for the encryption algorithm. Because there is always a possibility that brute force attack can be used to break the encryption. There was couple of alternatives present, such as Data Encryption Standard (DES), message digest, various password encryption algorithms and random number generators. After evaluated all the alternatives, DES and message digest are both considered to be too expensive and over qualified for the ID encoding schema. Linear congruent random number generator is taken as the ID encoding algorithm mainly because the following reasons:

- It is cheaper than DES and message digesting.
- It is sophisticated enough for ID encoding operation if original ID is not used as the seed for the random generator.

4.1.4 Design of ID Encoding Operation

Since the same seed would produce the same sequence of random numbers, unique seed needs to be feed into the PRNG every time in order to produce unique random number at each generation. Since we can't use the original ID as the seed because of the security reasons described above. What's a better choice than system clock time which is different every nanoseconds. Because of the linear congruent PRNG's mathematical properties, there is still a very slim possibility that a different seed might produce the same random number. A check is performed after each random ID generation to ensure that the random ID generated is unique within the "IDMapping" Table. The following Diagram illustrates the process flow of ID Encoding:

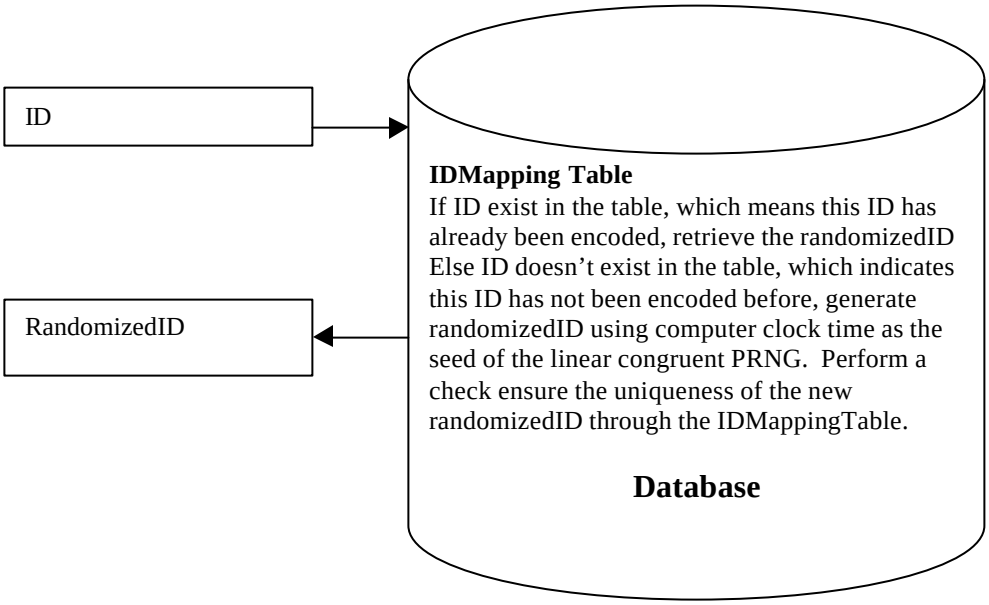

Figure 11 ID Encoding Process Flow Chart

All the criteria specified in section 4.1.2 has been meet in this ID encoding schema under the condition that only assigned personnel has access to "IDMapping" table.

4.2 Design Approach to Preserve the Business Logic

One of the requirements states that the business logic of the pre-processed patient data needs to be preserved in the transformed patient data. All the properties of a patient record should be preserved except those properties, which needs to be altered in order to anonymize the patient record. Each health organization normally has its own set of workflow procedures, patient records are generally created in a specific order; in the case of CDK, the creation date of each patient record represent sort of workflow logic. Therefore, the creation date of the transformed patient record should be as same as the creation date of input patient record. This could be easily achieved by setting the transformed patient record's timestamp as same as the timestamp of the input patient record. Most of patient record's document name has its own meaning, it would be wise to keep the transformed document's name as same as its original name. This also could

be easily achieved. The rule of thumb is “Don’t change anything unless you absolutely needs to do so!”

4.3 System Architecture

4.3.1 Design of Transformation Process

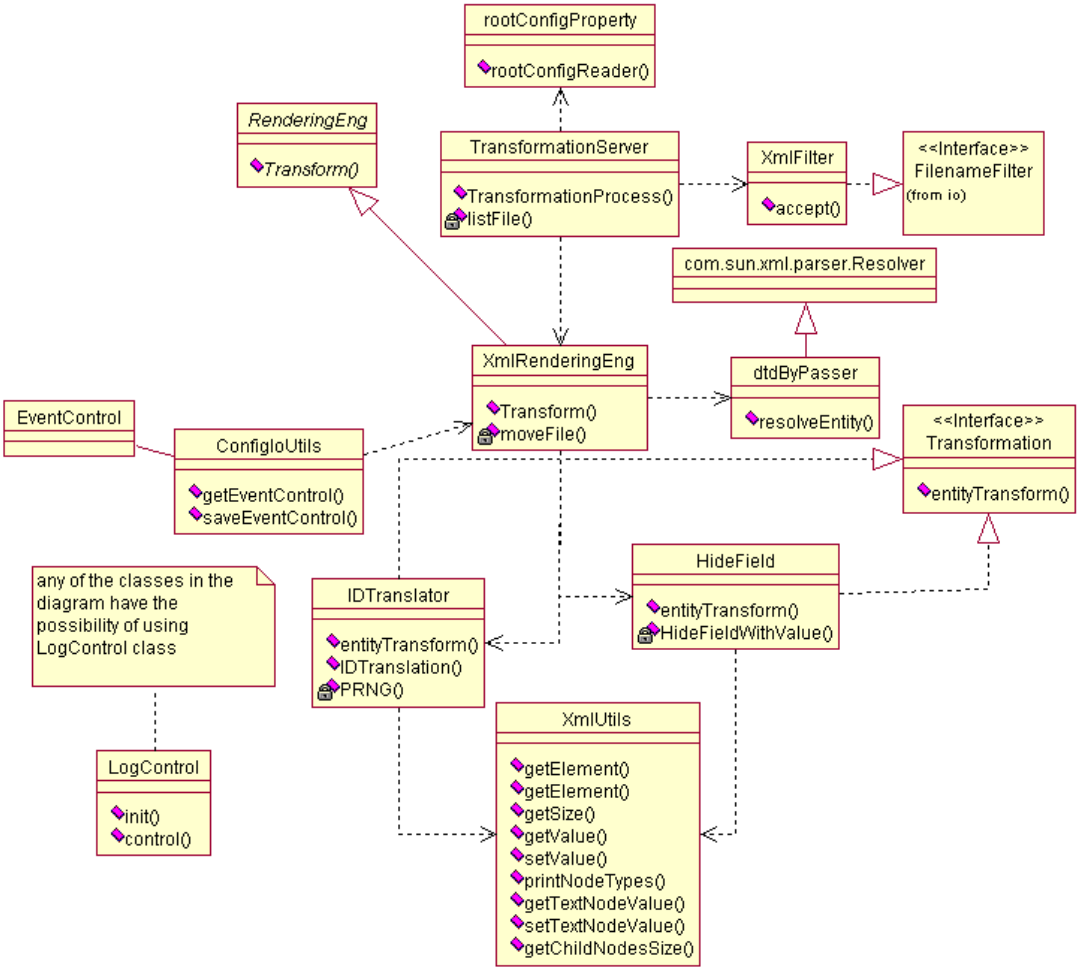

Figure 12 Class Diagram of Transformation Process

Description of Classes

Class "RenderingEng" : public, abstract

This class is provided as a prototype for document rendering engines.

Methods:

void Transform(String inputSource,String ouputSource) : public abstract

Method Transform takes input filename and output filename as parameters. Since this method is abstract. Any subclass of Class "RenderingEng" must provide document specific implementation for method Transform.

Class "XmlRenderingEng" : public, extends RenderingEng

This class is provided as a XML document rendering engine.

Constructor:

XmlRenderingEng(String inputFile,String outbox,String archive,long timeStamp)

Parameter "inputFile" indicates the input Xml document's file name (e.g., c:\inbox\1.xml). Parameter "outbox", "archive" indicate the "outbox" and "archive" file path. Parameter "timeStamp" indicates the timeStamp (creation date) of the input Xml document.

Methods:

void Transform (String inputFile, String outputFile) : public

Since class "XmlRenderingEng" is a subclass of class "RenderingEng". Class "XmlRenderingEng" must provide the document specific implementation for method Transform. The Transform method in class "XmlRenderingEng" takes in a Xml document, loads up the Event Control Model object based on the document type of that XML document. Get the Event ID from the Event Control Model, load up the corresponding class at run time and invoke the appropriate method dynamically. For an instance. Event ID extracted from Event Control Model is "IDTranslator", the "XmlRenderingEng" object will load up class "IDTranslator" at run time, instantiated it, and invoke the appropriate method (entityTransform) dynamically. This approach will ease the difficulty of future enhancement to the

system. In the future, if it is decided that additional transformation tasks such as noise reduction of certain entity is needed; a class “noiseReduction” will need to be created and in the Event ID field of the Event Control Model, the value has to be assigned “noiseReduction”. No modification is needed on the application binary because method Transform dynamically loads class “Event ID” and invoke method entityTransformation on that “Event ID” object.

Interface “Transformation”: public

Methods:

void entityTransform(XmlDocument doc, Item item) : public

Any class that implements interface “Transform” must provide its own implementation of method entityTransform. As discussed above about the dynamic method invocation of method entityTransform on “Event ID” object. Any class that is created regarding the transformation tasks (e.g., IDTranslator) must implement interface “Transformation”, therefore it is required to provide its own task specific implementation for method entityTransform.

Class “IDTranslator” : public, implements interface “Transformation”

Class “IDTranslator” performs the ID encoding operation over the specified entities in a patient record.

Methods:

entityTransform(XmlDocument doc, Item item) : public

Method “entityTransform” takes a XML Document and an Item object as the parameters. The Item object was first accessed in order to extract which entity in

the XML document needs the ID encoding operation. ID encoding operation will then be performed over the specified entities in the XML document.

String IDTranslation(String ID) : pubic, throws SQLException

Method “IDTranslation” takes the ID number as the parameter and make a JDBC connection to the IDMapping table in Oracle8i database. If the ID has already exists in the IDMapping table, which means this specific ID has already been encoded before. Method “IDTranslation” will retrieve the encoded ID from IDMapping table and return it to method “entityTransform”. Otherwise, if the ID is not in the IDMapping table, which means this specific ID has not been encoded before, the method will generate a random ID and store the newly generated random ID in the IDMapping table and return the encoded ID to method “entityTransform”.

PRNG() : default

Method “PRNG” is the pseudo random number generator which will generate the random ID. It generates the random ID based on the computer clock time.

Class “HideField” : public, implements interface “Transformation”

Class “HideField” performs the entity hiding task. It is called when entities in XML documents need to be hidden.

Methods:

entityTransform(XmlDocument doc, Item item) : public

Method “entityTransform” takes a XML Document and an Item object as the parameters. The Item object was first accessed in order to extract which entity in

the XML document needs to be hidden. Hide Field operation will then be performed over the specified entities in the XML document.

Class "dtdByPasser" : public, extends com.sun.xml.parser.Resolver

Class "dtdByPasser" is the quick and dirty way to bypass the DTD reference in XML documents programmatically, i.e., by creating your own "EntityResolver". Whenever there is a call to a DTD file, create and return an empty "InputSource". "com.sun.xml.parser.Resolver" is the default entity resolver, subclass it and override "resolveEntity()" method.

Class "XmlFilter" : public, implements interface "java.io.FilenameFilter"

Class "XmlFilter" implements the "accept()" method in interface "java.io.FilenameFilter" in such a way that only files with ".xml" extension will be accepted.

Class "XmlUtils" : public

Class "XmlUtils" is utility class who provides a set of operations, which are useful to extract information from a XML document. All the methods in class "XmlUtils" are declared static.

Class "rootConfigProperty" : public

Class "rootConfigProperty" reads configuration information from the "RootConfig.xml" which stores the root configuration information. Method "rootConfigReader" is provided to perform such task.

Class "TransformationServer" : public

Class "TransformationServer" initializes the Patient Data Transformation System.

Methods:

static void TransformationProcess() : public

Method “TransformationProcess” grabs the patient records in “inbox” directory and invokes “XmlRenderingEng” object to process these patient records.

static File[] listFiles(File entry) : private

Method “listFiles” takes the “inbox” directory as the parameter and grab all the files with “.xml” extension (patient records) in that directory. It is enforced programmatically that only the files with “.xml” extension will be taken (ref. Class “XmlFilter”)

Class “LogControl” : public

Class “LogControl” provides the logging facilities. Method “init” initialize the logging facility.

4.3.1.1 Transformation Process Class Diagram Description

The transformation of patient records is a sequential process. Once the “TransformationServer” object grabs the a list of patient records, it will process it one at a time, not concurrently. The initial approach was to process the patient records concurrently, in the intention of improving the performance of the system. However, later on it was realized that if the patient records are processed concurrently, it will cause some problems with the ID Encoding Operation.

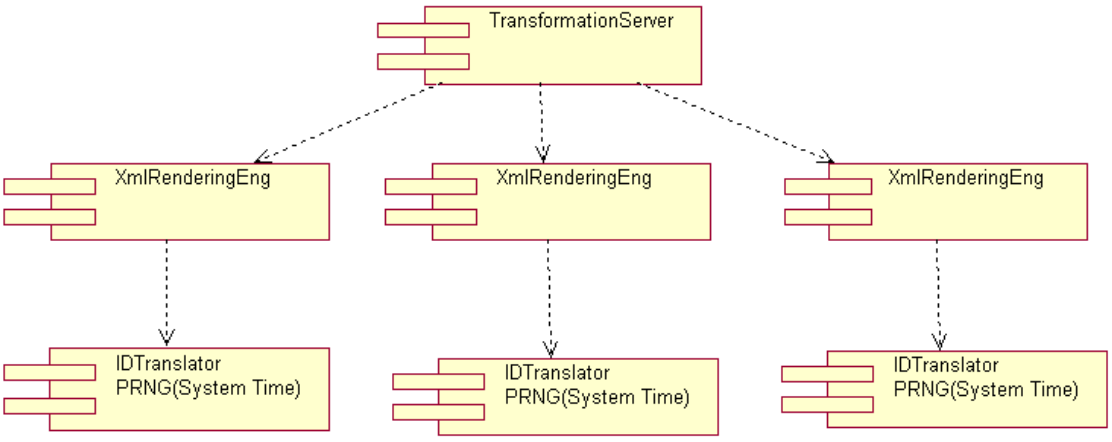

Figure 13 Concurrent Processing of Patient Records

As shown in the above diagram, it is possible for the "System Time" parameter to be the same while method "PRNG" is invoked since multiple concurrent processes are running. Therefore, the Randomized ID generated will also be the same. Even though, there is a check enforced to make sure that the Randomized ID that is going to be inserted into the "IDMapping" table is unique within the table. However, the following could occur, the check for the uniqueness of the Randomized ID can be executed one right after another before any insert Randomized ID tasks was done. So Each "XmlRenderingEng" processes will think the Randomized ID it generated is unique and insert its newly generated Randomized ID into the "IDMapping" table. Then the one to one relationship between the ID and encoded ID won't hold.

Component Interaction

The "rootConfigureProperty" object first reads the "inbox", "outbox" and "archive" location from "RootConfig.xml" and pass those values into the "TransformationSever" object. The "TransformationServer" object is designed to be an active object. Therefore, instead of sitting around waiting for the input patient records,

“TransformationServer” object goes and look for the patient records in the inbox directory. “TransformationServer” object grabs the XML based patient records and keep them in an array, and for each XML document entry in that array, an “XmlRenderingEng” object will be created to process it.

Inside each “XmlRenderingEng” object, the corresponding “EventControl” object will be loaded. And depending on the “Event ID” field of the “Item” object inside the “EventControl” object, Different transformation tasks will be performed on the given entities inside the patient record by loading up “Event ID” classes which implements interface “Transformation” (e.g., IDTranslator).

4.3.2 Design of Event Control Model Configuration

The Event Control Model Configuration GUI developed should have a consistent look and feel across platforms. The components of the GUI should be consistent with other applications running on the same platform, given that the platform in question has a standardized set of GUI components.

4.3.2.1 Design of Command Handling

Event Control Model Configuration GUI is event-driven. User interaction with the GUI results in events being generated to inform the application of user actions. Clicking a button, closing a window results in appropriate event being sent to the application.

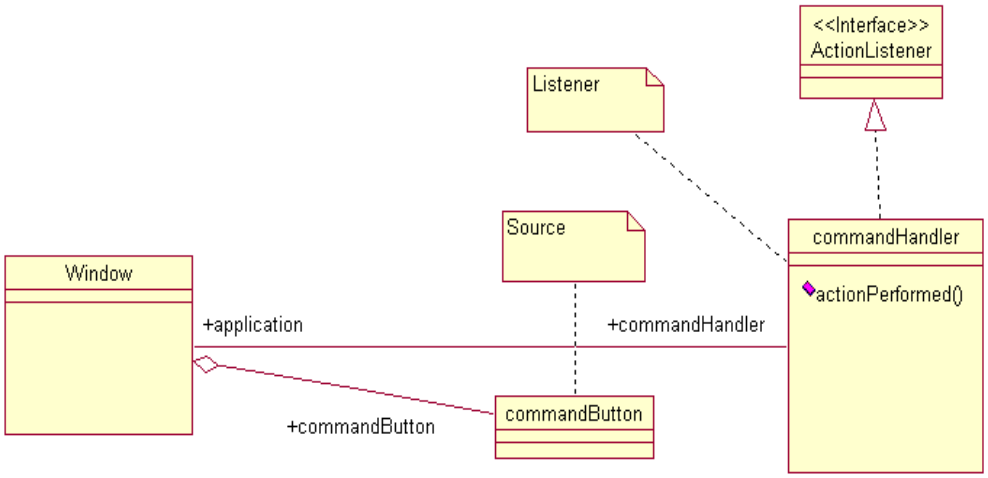

Figure 14 Event Delegation

The design of event handling feature is based on event delegation model. In the event delegation model, an event source informs event listeners about events when these occur, and supplies the necessary information about these events. In other words, an event listener, which is interested in receiving events, is informed by an event source when certain types of events occur, so that it can take appropriate action.

4.3.2.2 Event Control Model Class Diagram

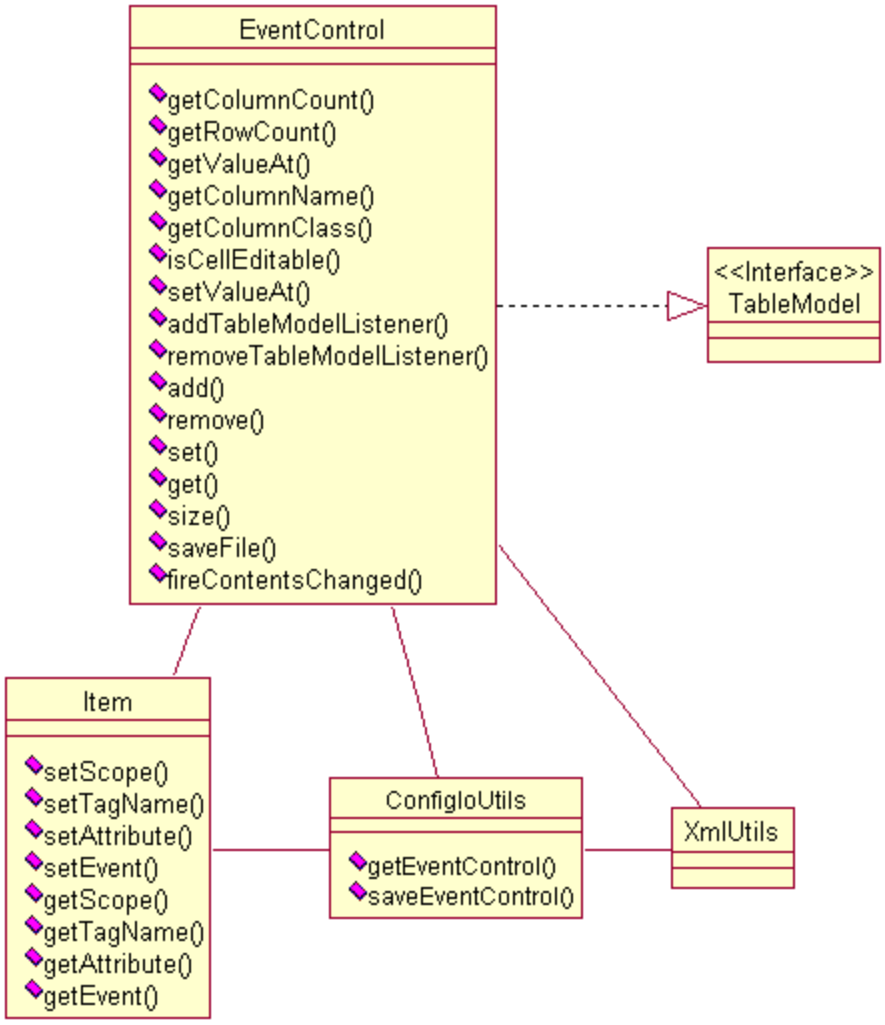

Figure 15 Class Diagram of Event Control Model

Class "EventControl" : public, implements interface "TableModel"

In class "EventControl" the presentation of the Event Control Model is a table-structured data set. However, the underlying Event Control Model data structure is a List of "Item" objects. The separation between presentation of the Event Control Model and the actual physical data structure give user the freedom to tailor the presentation of the Event Control Model according to his/her preference. Most of the methods provided in class "Event Control" dealing with the visual presentation of the Event Control Model.

Methods such as “add”, “remove”, “set”, “get”, “size” dealing with the manipulation of the physical data structure of Event Control Model.

Class “Item” : public

Each “Item” object will be an entry of the Event Control Model. A set of methods are provided in class “Item” to access and modify “Item” object.

Class “ConfigIoUtils” : public

Class “ConfigIoUtils” is a utility class. It provides the means for reading and writing of the Event Control Model. A default constructor is provided.

Methods:

EventControl getEventControl(String file) : public, static

Method “getEventControl” takes the document type specific event control model configuration file name as the parameter and returns the corresponding Event Control Model.

void saveEventControl(java.util.List data) : public, static

Method “saveEventControl” takes the Event Control Model as the parameter and saves the contents of the Event Control Model into the corresponding event control model configuration file.

4.3.2.3 “EventControlFrame” Class Diagram

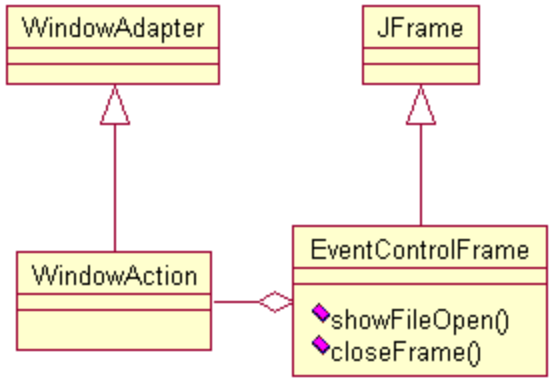

Figure 16 Class Diagram of “EventControlFrame”

Class “EventControlFrame” : public, extends JFrame

Class “EventControlFrame” is the root graphic component of the Event Control Model Configuration GUI. Other graphic components are all added upon “EventControlFrame” object.

Constructor:

A default constructor is provided which initializes the Event Control Model Configuration GUI.

Methods:

void showFileOpen() : public

Method “showFileOpen()” brings up the file dialog box.

void closeFrame() : public

Method “closeFrame” close and exit the Event Control Frame.

Class “WindowAction” : private, extends WindowAdapter

Class “windowAction” is a private inner class of class “EventControlFrame”. It handles window-generated events.

4.3.2.4 “EventControlOpen” Class Diagram

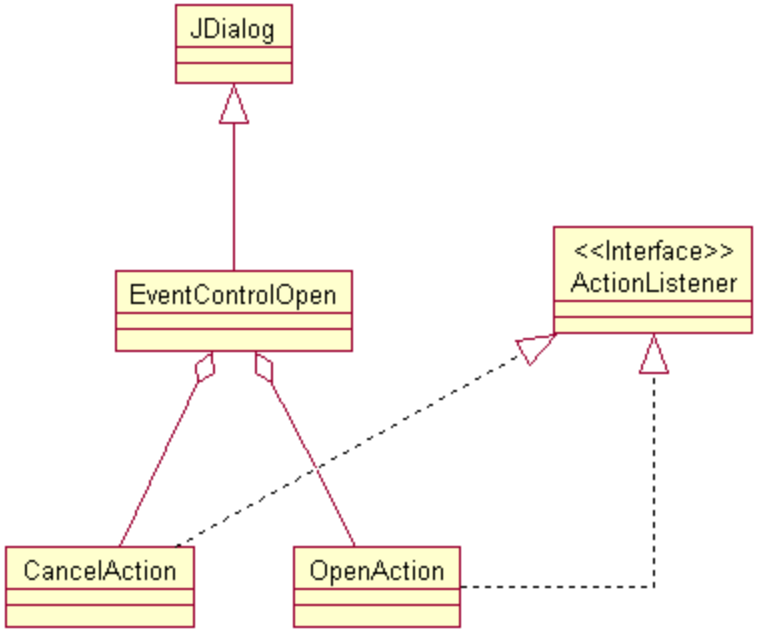

Figure 17 Class Diagram of “EventControlOpen”

Class “EventControlOpen” (extends class “JDialog”) is a graphic component, which is a dialog box that asks for user’s input of Event Control Model configuration file name. Class “CancelAction” and class “OpenAction” are both protected inner class of class “EventControlOpen” and both of the class implements interface “ActionListerner”. Class “CancelAction” is the class responsible for events generated by the “Cancel” button on the dialog box. Class “OpenAction” is the class responsible for events generated by the “Open” button on the dialog box.

4.3.2.5 "EditDialog" Class Diagram

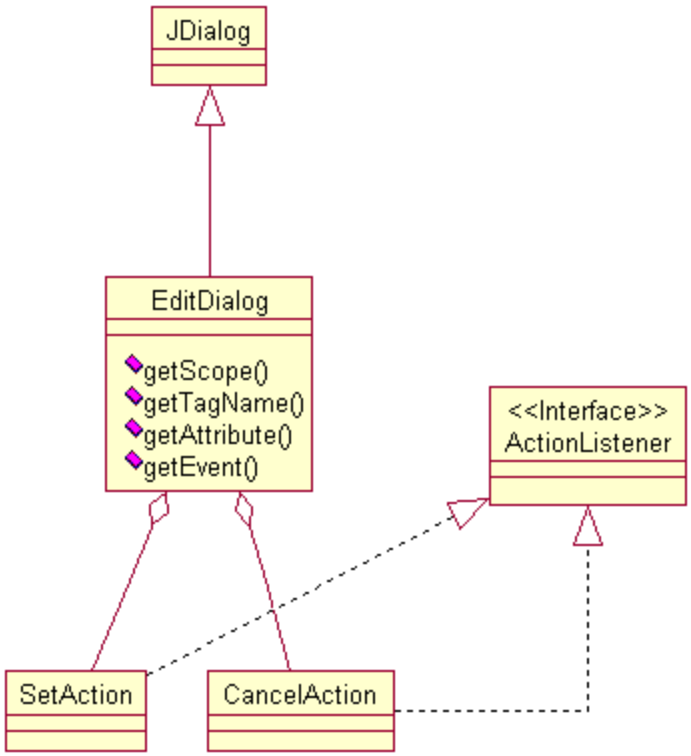

Figure 18 Class Diagram of "EditDialog"

Class "EditDialog" (extends class "JDialog") is a graphic component, which is a dialog box that asks the user's input of the Event Control Model's entry information. A set of operations such as "getScope", "getTagName", etc,. are provided to retrieve the entry's information. Class "SetAction" and class "CancelAction" are both protected inner class of class "EditDialog", and both of these classes implement interface "ActionListener". Class "SetAction" is the class responsible for events generated by the "Set" button on the dialog boxand class "CancelAction" is the class responsible for events generated by the "Cancel" button on the dialog box.

4.3.2.6 "EventControlPanel" Class Diagram

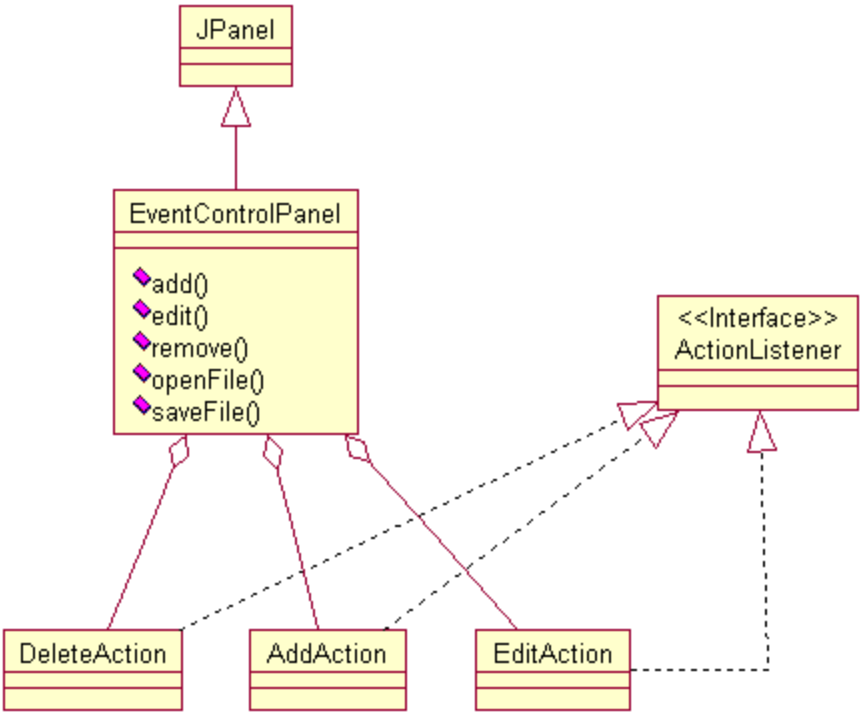

Figure 19 Class Diagram of "EventControlPanel"

Class "EventControlPanel" is a JPanel component (extends class "JPanel"), which is a panel with "Add", "Edit" and "Delete" buttons on it. Its primary responsibility is to add, edit and delete Event Control Model's entry. Methods "add", "edit" and "remove" are called after the entry information is entered in the dialog box (ref. Section 4.3.2.5). These methods will modify the existing Event Control Model in memory based on the information give in the dialog box. Class "DeleteAction", class "AddAction" and class "EditAction" are all protected inner class of class "EventControlPanel" and they all implement interface "ActionListener". Class "DeleteAction" is the class responsible for events generated by the "Delete" Button. Class "AddAction" is the class responsible for

events generated by the “Add” Button. Class “EditAction” is the class responsible for events generated by the “Edit” Button on the panel.

4.3.2.7 “EventControlToolBar” Class Diagram

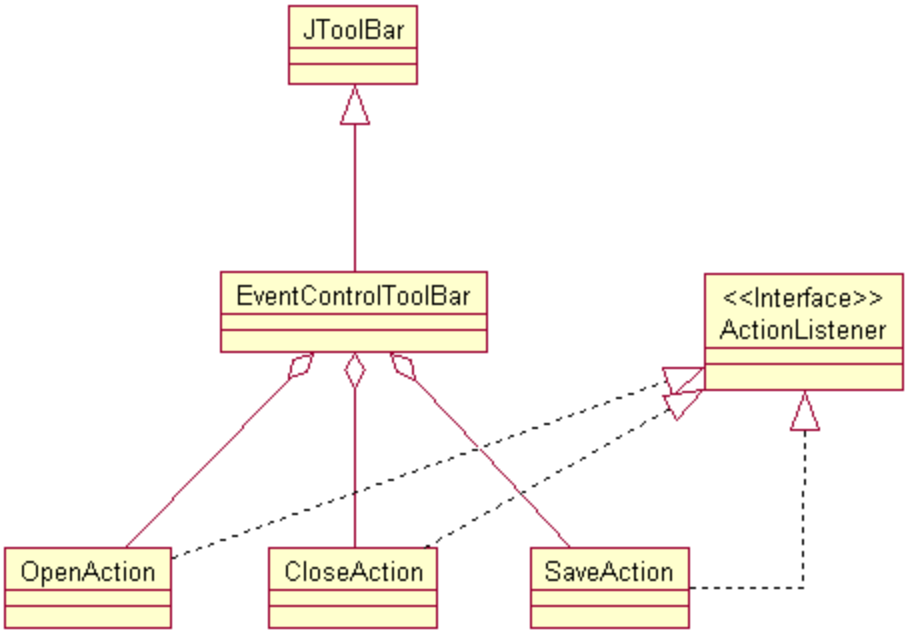

Figure 20 Class Diagram of “EventControlToolBar”

Class “EventControlToolBar” is a JToolBar component (extends class “JToolBar”), which is a ToolBar has “Open File”, “Save File” and “Close Frame” Button on it. Class “OpenAction”, class “CloseAction” and class “SaveAction” are all protected inner class of class “EventControlToolBar” and they implement interface “ActionListener”. Class “OpenAction” is the class responsible for events generated by the button “Open File”. Class “CloseAction” is the class responsible for events generated by the button “Close Frame”. Class “SaveAction” is the class responsible for events generated by button “Save File”.

4.3.2.8 Class Diagram Description

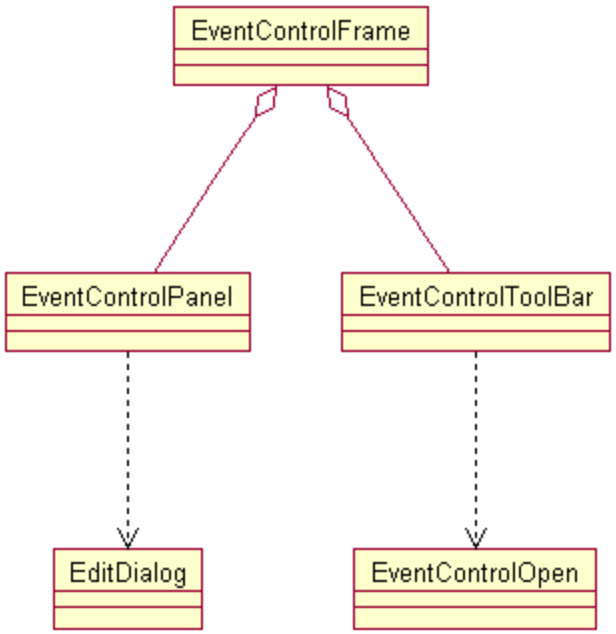

Figure 21 Class Diagram of GUI Core

Class “EventControlFrame” is the root component of the GUI, and it is physically constructed by component “EventControlPanel” and “EventControlTooBar”. Both “EventControlPanel” object and “EventcontrolToolBar” object depends on the services provided by the “EditDialog” object and “EventControlOpen” object.

5 IMPLEMENTATION

The Patient Data Transformation System is developed on Sun Solaris using JDK 1.2, Oracle 8i database, JDBC thin driver, Sun Project X XML parser, Log4J API and XML. The Patient Data Transformation System's performance directly depends on the services offered by these tools. The following sections discuss the criteria for choosing these tools.

5.1 Implementation Decisions

5.1.1 Choice of Implementation Language

Java language as defined in Sun Microsystems Java Development Kit (JDK 1.2) was chosen as the implementation language for the following reasons:

- Java provides a very easy and extensive development API.
- Java enforces modular development thereby makes the future enhancements and modification simpler.
- Any application built in Java is portable across multiple platforms.
- The automatic garbage collection and memory management facilities enable the development of a robust application.
- The Swing API in Java language provides a set of light weight graphic components for GUI development which has the ability to masquerade as native components of a range of platforms, and allows new look & feel schemes to be designed. The look and feel can be changed dynamically.

5.1.2 Choice of the XML Parser

Since all the patient records are XML based, and a lot of the process time in the transformation system will be spent on the parsing and manipulation of the patient

records. Therefore, a right XML parser will make a big difference in the system performance. The Sun Project X parser was chosen as the XML parser, however, it was a tough decision.

A comparative test was done among the Sun Project X parser, IBM XML parser and OpenXml parser.

Testing Methodology:

The parsers were tested by generating different size of XML documents that contains an internal DTD, elements, attributes, entities, #PCDATA sections and CDATA sections.

Test Limitations:

The test results and testing methods outlined here only compare the read performance of these XML parsers. That is, the testing only measures the time it takes to read an XML document and convert it into a (DOM 1.0) document object. It does not say anything about the performance of these parsers for modifying the (DOM) document object using the DOM API. These tests show the read-only performance of the parsers. Also, these tests have nothing to do with the SAX performance of these parsers, because we are only testing DOM performance.

Testing Environment:

The following hardware configuration was used

- Single processor machine, PentiumII/233, 256MB PC100 SDRAM, ASUS P2B motherboard, IDE harddrive, WinNT4.0 SP3

- Dual processor machine, PentiumII/400, 256MB PC100 SDRAM, ASUS P2B-DS motherboard, Ultra Wide SCSI II harddrive, WinNT4.0 SP4.

Test Results:

The following table contains test data for the JDK 1.2 Classic VM on a Dual processor machine.

Input XML file size	Sun (in seconds)	IBM (in seconds)	OpenXML (in seconds)
100KB	1.15	1.45	1.65
500KB	2.06	1.85	4.29
1MB	2.75	2.45	6.59
2MB	4.11	3.53	11.21
3MB	5.54	4.60	15.54
4MB	6.93	5.82	20.21
5MB	8.31	6.82	24.93
7MB	11.11	9.51	34.00
10MB	15.21	12.78	48.12
15MB	21.93	18.65	84.18
20MB	29.26	25.04	not tested
25MB	35.93	OutOfMemoryException	not tested
30MB	49.67	OutOfMemoryException	not tested
35MB	66.35	OutOfMemoryException	not tested
40MB	74.12	OutOfMemoryException	not tested

Figure 22 XML Parser Comparative Testing 1

The following table contains test data for JDK 1.2 classic VM on a single processor machine.

Input XML file size	Sun (in seconds)	IBM (in seconds)	OpenXML (in seconds)
100KB	1.78	1.74	1.82
500KB	5.66	2.35	5.42
1MB	10.03	3.11	9.11
2MB	13.98	4.82	14.03
3MB	16.43	6.30	18.67
4MB	18.94	8.06	23.11
5MB	20.96	9.39	27.69
7MB	25.62	12.42	36.72
10MB	32.30	17.55	49.59
15MB	45.39	24.73	92.99
20MB	54.31	32.47	not tested
25MB	66.87	OutOfMemoryException	not tested
30MB	76.33	OutOfMemoryException	not tested
35MB	156.02	OutOfMemoryException	not tested
40MB	171.74	OutOfMemoryException	not tested

Figure 23 XML Parser Comparative Test 2

Analysis:

As shown in both of tables, IBM parser performs the best. The Sun parser however can deal with the largest files in similar memory configurations. The OpenXML parser comes in at third place for speed and file size handling. However, after a detailed analysis done with IBM parser, it was found out that IBM parser doesn't support the String representations of method getAttributes() in DOM API. Since getAttributes() method will be used heavily in the Transformation System, the Sun Project X XML parser was chosen so the implementation of the system would become easier considering the performance

difference is very slim between the Sun Parser and the IBM parser if the document size is under 1MB which is the case with patient documents.

5.1.3 Choice of the Database

First of all, the database chosen has to be a secure database with sophisticated access control procedure since vital mapping information between the ID and its encoded ID will be stored in that database. And because of the implementation language was Java, the database must have Java Database Connectivity (JDBC) drivers available. Hence, the Oracle ver 8.1.5 was chosen as the database for the Patient Data Transformation System.

5.1.4 Choice of the JDBC driver

A JDBC driver is the set of classes that implement the JDBC interfaces for a particular database. There are four different types of JDBC driver: A Type 1 driver is a JDBC-ODBC bridge driver; this type of driver enables a client to connect to an ODBC database via Java calls and JDBC, neither the database nor middle tier need to be Java compliant. However, ODBC binary code must be installed on each client machine that uses this driver. A Type 2 driver converts JDBC calls into calls for a specific database. This driver is referred to as a "native-API, partly Java driver." As with the Type 1 driver, some binary code may be required on the client machine. A Type 3 driver is a JDBC-Net pure Java driver, which translates JDBC calls into database-independent net protocol. Finally, a Type 4 driver, or the "native protocol, pure Java" driver converts JDBC calls into the network protocol used by the database directly. A Type 4 driver requires no client software.

Oracle provides both Type 2 and Type 4 drivers. All Oracle JDBC drivers support the full JDBC specification, but in addition, they support the extended capabilities of the Oracle database. Oracle's Type 4 JDBC driver, referred to as the Oracle "thin" driver includes its own implementation of a TCP/IP version of Oracle's Net8 written entirely in Java, so it is platform independent, and doesn't require any Oracle software on the client Side. Oracle "thin" driver (Type 4) was chosen as the JDBC driver for the above reason.

6 ANALYSIS AND CONCLUSION

6.1 Analysis of the System

Component Testing

Component testing was done at each stage of the development lifecycle. This implies that classes is built and refined iteratively and therefore tested iteratively. In addition to test the new functionality added to a class during an iteration, the pre-existing functionality is tested to determine that it still performs correctly within its new environment.

System Integration Testing

Several test cases are done after all the components are integrated into the Patient Data Transformation System.

Test Case 1:

Objective: Test the basic operations of the system.

Input: 10 XML based patient records with pre-defined entity relationships and business logic in “inbox”.

Output: 10 transformed anonymous patient records in “outbox”. 10 copies the pre-processed patient records in “Archive”. The output satisfies the system requirements 1, 2 & 3 defined in section 3.1.

Test Case 2:

Objective: Test the logging functionality and Error Handling

Input: 10 XML based patient records with pre-defined entity relationships and business logic. One of these patient records is not a well-formed patient record.

Output: 9 transformed anonymous patient records in "outbox". 10 copies the pre-processed patient records in "Archive". "LOG" file indicated that which patient record transformation process has failed caused by parsing error. The output satisfies the system requirements 4 defined in section 3.1.

Test Case 3:

Objective: portability testing of the system

Testing Environment:

Dell OptiPlex GX1, WinNT.

Ultra Enterprise 450, Sun OS

The Patient Data Transformation System behaved as expected on both of the platforms. No abnormal behavior was discovered. Portability requirement has been satisfied.

Performance Analysis

Testing Environment:

Ultra Enterprise 450 w/ 4 400Mhz UltraSparc 2 processors, 1 gig of Ram

Dell OptiPlex GX1, 300MHZ, 64MB

Testing Result:

Ultra Enterprise 450 w/ 4 400Mhz UltraSparc 2 processors, 1 gig of Ram:

Number of Patient Records	Total Size of Patient Records	Total Process Time
19	154k	57,695ms

Table 3 Utral Enterprise 450 Testing Result

Process time/patient record = Total Process Time / Number of Patient Records = 3036ms

Dell OptiPlex GX1, 300MHZ, 64MB:

Number of Patient Records	Total Size of Patient Records	Total Process Time
19	154k	54,947ms

Table 4 Dell OptiPlex GX1 Testing Result

Process time/patient record = Total Process Time / Number of Patient Records = 2891ms

Analysis:

For an average size PRA (10k), it took both system around 3 seconds to process it. Of course, larger files will take longer time.

6.2 Future Work

More transformation tasks can be added upon the existing system to handle more data manipulation operations, such as noise reduction, find the invariant representation of data, etc. Extensive integration testing and performance analysis strongly recommended to be done in order to analyze system behavior in large and complicated environment.

6.3 Conclusion

This thesis described a Patient Data Transformation System, which provides confidential high quality patient data as the data source for data mining tasks. The objective of the system as stated in the first chapter has been met. The system is user friendly and provides a user interface.

REFERENCE

1. Fayyad, U. “Data Mining and Knowledge Discovery: Making Sense Out of Data”. IEEE Expert Intelligent Systems and their Applications, (11):5, 1996, pp.20-25.
2. Fayyad, U. Piatetsky-Shapiro, G., & Smyth, P. “The KDD Process for extracting Useful Knowledge from Volumes of Data”. Communications of the ACM, (39):11, 1996, pp.27-34.
3. Kimball, R. & Reeves, L. “The Data Warehouse, Lifecycle Toolkit”. Wiley Computer Publishing, 1998.
4. Hagan, K. “Member Service Bulletin” Oregon Association of Hospitals and Health Systems, 2000.
5. MacCraw, G., & Viega, J. “Make your software behave: Playing the numbers, truly secure software needs an accurate random number generator” Reliable Software Technologies, 2000.
6. Charles P. Pfleeger: “Security in Computing Second Edition”. Prentice Hall PTR, 1997.
7. Bresnahan, J. “Health Care Data Mining: A Delicate Operation”. CIO, June 15, 1997, pp. 44-54.
8. Dudeck, J. “XML & Health Care”. XML Europe 2000, June 16, 2000.
9. CareFlow|Net. “CDK Administration Guild”. CareFlow|Net, 1999.
10. World Wide Web Consortium. “Document Object Model Level 1 Specification”. World Wide Web Consortium, 1998.
11. World Wide Web Consortium. “Extensible Markup Language Specification Version 1.0”. World Wide Web Consortium, 2000.
12. Log4J.org. “Logging Package for Java Specification Version 0.8.5b”. Log4J.org, 2000.
13. Sun Microsystems Inc. “JDBC API Specification Version 2.0”. Sun Microsystems, 1999.

14. Mughal, K. & Rasmussen, R. “A Programmer’s Guide to Java Certification”. Addison-Wesley, 1999.

15. Sun Microsystems Inc. “Java 2 SDK Specification”. Sun Microsystems, 2000.