Customized Tool for Data Transfer

Graduate Technical Report

Submitted to the Faculty of
the Department of Computing and Mathematical Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Srikanth Kasha
Spring, 2002

COMMITTEE MEMBERS

R. Stephen Danelly, Ph.D.
Committee Chairperson

David Thomas, Ph.D
Committee Member

Michelle Moore, Ph.D.
Committee Member
Customized Tool for Data Transfer

Graduate Technical Report

Submitted to the Faculty of
the Department of Computing and Mathematical Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Srikanth Kasha
Spring, 2002

COMMITTEE MEMBERS

R. Stephen Dannelly, Ph.D.  ___________________
Committee Chairperson

David Thomas, Ph.D  ___________________
Committee Member

Michelle Moore, Ph.D.  ___________________
Committee Member
ABSTRACT

The *Customized Tool for Data Transfer* (CTDT) is designed and developed to automate the customized periodic transfer of data between two different kinds of databases located in different network platforms. The tool was developed for the Joint Center for Structural Genomics (JCSG) of the San Diego Supercomputer Center (SDSC) at the University of California, San Diego. This application is used in transferring the data collected during the crystallization phase of genome sequencing, on a periodic basis at the Genomics Institute of the Novartis Research Foundation (GNF) to the tracking database which stores the data from all the phases of genome sequencing, Bioinformatics Archival and Retrieval Technology (BART) database of JCSG. This data transfer is implemented in several sequential phases using different modules of this project, which were installed in heterogeneous platforms of the JCSG and GNF local area networks. As the last part of the project, the data was made available to the public via the JCSG website. This tool was designed in compliance with the agreement between JCSG and GNF according to which there should not be any significant risk either to the intellectual property rights of GNF and the regulatory procedures of JCSG at any given time of this data transfer.
# TABLE OF CONTENTS

ABSTRACT ........................................................................................................................................ ii
TABLE OF CONTENTS ...................................................................................................................... iii
LIST OF FIGURES .......................................................................................................................... iv
1. INTRODUCTION AND BACKGROUND ................................................................................. 1
   1.1 Introduction .......................................................................................................................... 1
      1.1.1 What is Genome Sequencing? ....................................................................................... 1
      1.1.2 About the Institute – JCSG ......................................................................................... 1
   1.2 Background and Rationale for CTDT .................................................................................. 4
   1.3 Past Scenario ....................................................................................................................... 5
   1.4 How CTDT fits into the system .......................................................................................... 6
   1.5 Content of this Document ................................................................................................. 6
2. CUSTOMIZED TOOL FOR DATA TRANSFER (CTDT) ......................................................... 7
   2.1 Purpose of the CTDT ......................................................................................................... 7
   2.2 Module-1 ............................................................................................................................ 7
   2.3 Module-2 ............................................................................................................................ 9
   2.4 Module-3 ........................................................................................................................... 10
   2.5 Module-4 ........................................................................................................................... 11
3. SYSTEM DESIGN ....................................................................................................................... 14
   3.1 Design Overview .............................................................................................................. 14
   3.2 Module-1 ............................................................................................................................ 16
      3.2.1 Description .................................................................................................................. 16
      3.2.2 Description of Major Components ............................................................................. 17
   3.3 Module-2 ............................................................................................................................ 19
      3.3.1 Description .................................................................................................................. 19
      3.3.2 Description of Major Components ............................................................................. 19
   3.4 Module-3: .......................................................................................................................... 21
      3.4.1 Description .................................................................................................................. 21
      3.4.2 Description of Major Components ............................................................................. 21
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5.</td>
<td>Module-4</td>
<td>25</td>
</tr>
<tr>
<td>3.5.1.</td>
<td>Description</td>
<td>25</td>
</tr>
<tr>
<td>3.5.2.</td>
<td>Description of Major Components</td>
<td>25</td>
</tr>
<tr>
<td>4.</td>
<td>EVALUATION AND RESULTS</td>
<td>27</td>
</tr>
<tr>
<td>5.</td>
<td>CONCLUSION</td>
<td>29</td>
</tr>
<tr>
<td>6.</td>
<td>FUTURE WORK</td>
<td>31</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>BIBLIOGRAPHY AND REFERENCES</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>APPENDICES</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>Code for Module - 1</td>
<td>36</td>
</tr>
<tr>
<td>2.</td>
<td>Code for Module - 2</td>
<td>37</td>
</tr>
<tr>
<td>3.</td>
<td>Code for Module - 3</td>
<td>40</td>
</tr>
<tr>
<td>4.</td>
<td>Code for Module - 4</td>
<td>43</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1.1 Structure of JCSG .......................................................... 2
Figure 1.2 Sample Tracking Database of JCSG .................................. 3
Figure 2.1 Schematic View of CTDT ................................................... 8
Figure 2.2 A snapshot of the email ...................................................... 9
Figure 2.3 Sample XML file generated by Module-1 ................................ 9
Figure 2.4 The sample snapshot of the desktop ..................................... 10
Figure 2.5 A snapshot of the screen while Module-2 is executing ............. 11
Figure 2.6 A sample of populated BART table ...................................... 12
Figure 2.7 A snapshot of www.jcsg.org/prod/gnfweb.html ................... 12
Figure 2.8 A snapshot of a query output .............................................. 13
Figure 3.1. Context level Data Flow Diagram for CTDT ....................... 15
Figure 3.2. Level-1 Data Flow Diagram for CTDT ............................... 15
Figure 3.3. Level-2 Data Flow Diagram for Tool-1 (Modules 1&2) .......... 17
Figure 3.4. Level-2 Data Flow Diagram for Modules - 3 & 4 ................. 22
Figure 3.5. Schema of BART database [West 2001] ............................. 24
1. INTRODUCTION AND BACKGROUND

1.1 Introduction

This project, Customized Tool for Data Transfer (CTDT), constitutes a significant part of the tracking database Bioinformatics Archival and Retrieval Technology (BART) database. BART stores data from all the phases of the genome sequencing research conducted by the Joint Center for Structural Genomics (JCSG). The CTDT is designed and developed to automate the customized periodic transfer of data between two different kinds of databases, namely Microsoft Access and Oracle, which are located in different network platforms at two different sites. These two sites are the Genomics Institute of Novartis Research Foundation (GNF) and JCSG.

1.1.1 What is Genome Sequencing?

Genome Sequencing is the specification of the order of the nucleotides in an RNA or DNA molecule. These sequences are analysed in the process of investigating a nucleotide or amino acid sequence, employing in silico biology techniques. These studies are used in such fields as cloning of genes, genetic pathology, and human genome exploration. Genome sequencing consists of several sequential phases, namely cloning, expression, purification, crystallization, diffraction, structure modelling, refinement and structure validation.

1.1.2 About the Institute – JCSG

The JCSG is a collaboration between four main institutions, three academic (The Scripps Research Institute (TSRI), the Stanford Linear Accelerator (SLAC), the University of California, San Diego (UCSD)/ the San Diego Supercomputer Center (SDSC)) and one commercial (GNF).
The principal mission of the JCSG is to engineer a high throughput crystallographic pipeline that speeds the production of diffraction quality crystals and the determination and release of the three dimensional structures of genomes for public use [JCSG 2001]. Such a pipeline requires a substantial hardware/software infrastructure to support the tracking of samples, and to track successes and failures at each stage. The records of success and failures make it possible to “learn” what rules govern success and failure at each step along the way (Fig 1.1).

![Figure 1.1 Structure of JCSG](image)

The first step in creating this infrastructure is to create a tracking database that can report the status of samples and their history to operators along the pipeline. This
A database contains the records of both successes and failures for the purpose of data mining after the database is populated. (Fig 1.2)

**Figure 1.2 Sample Tracking Database of JCSG**

Recently, nine high throughput genomics projects have been created by National Institutes of Health (NIH). Of these nine, only one, JCSG has access to robotic machinery to actually enable the kind of high throughput crystallization/structure determination (an important phase in genome sequencing) that one might expect in industry, see Fig. 1.1. The unique benefits of this environment for learning about crystallization are limited only by JCSG's ability to code a sample tracking system that will allow rapid progress tracking through the system, see Fig. 1.2., and immediate feedback on experimental success or failure of individual samples [Miller 2001]. If JCSG can capture this information, in a
year, they can create a body of knowledge that equals that of a crystallographic laboratory over the entire career of its research.

1.2 Background and Rationale for CTDT

The present goal for the tracking team at JCSG is to create a tracking database system that will allow rapid accurate entry of success and failures at each step of a substantial industrial-style pipeline. The tracking database will receive input from the results of various types of experiments, which produce various types of data, and this data will be immediately available to workers at each stage of the pipeline via a web interface. The data will also be made available within the Bio-informatic Core (BIC) of the center for mining (part of JCSG), to learn what correlations exist among the success and failures of the targets. Such systems have been created widely in industry settings, but we know of no real precedent in the current environment, which is a consortium of 4 distinct institutions, one of which is a publicly held company GNF, that has significant intellectual property concerns.

Currently the institution that produces the proteins, purifies them, and crystallizes them is the GNF. GNF keeps its own distinct database that tracks target progress once the target genes are cloned and inserted into Escherichia coli [GNF 2001]. Because there are significant intellectual property issues in interfacing with GNF, it is essential that JCSG be able to extract the data from their database in an automated way that does not present a risk to these intellectual property rights.

The CTDT is a tool that extracts the data from an Access database at GNF, checks the data for integrity, and deposits it in JCSG’s BART database. This process is unusual in that it cannot be done directly, but must be done indirectly using File Transfer Protocol
with maximum possible network security. It is an indirect transfer as the tool transfers the data between two databases at two different sites in an incremental manner on a periodic basis as opposed to the commercially available data migration tools that are used for transferring the total data between databases in one go. The other reasons for this indirect transfer are as following,

- As per JCSG's security policies and regulations GNF is not allowed to directly participate in BART.
- At the same time, according to GNF's company policies it cannot allow any external entity to have the access to its confidential research information except for the crystallization data (the data to be transferred by the tool developed as a part of the CTDT) which will take part in JCSG's tracking database.

There are some commercial tools to do the data transfer from MS Access to Oracle such as Oracle's Migration Work Bench [Oracle 2001], Oracle Migration Assistant for MS Access which is shipped along Oracle SQL*Net (Net 8) product. But the intended data transfer accomplished by the CTDT cannot be achieved with these tools, as the data transfer is not a one-time task, rather it must be done repeatedly every week as the database at GNF gets updated weekly. In addition to this, the required criticality of the data integrity and sensitivity of the data even for very minute errors is accomplished by the CTDT, which are normally overlooked by the available commercial tools.

1.3 Past Scenario

Before installing the CTDT, an individual designated by GNF would collect the data for crystallization status and crystal parameters. These data are input into a
Microsoft Access database on a Microsoft Windows NT Workstation on the GNF intranet. The updates occur approximately weekly and GNF has made these data available on an 'On Request' basis to the Bioinformatics Core of JCSG.

1.4 How CTDT fits into the system

The CTDT fits into the BART tracking database as a data transfer tool and as a data loader that transfers the required data (data about expression, solubility, crystallization conditions, which is an essential part for the tracking database-BART) from GNF database to BART, see Fig. 1.3. The CTDT transfers the crystallization data from GNF's Access database to BART in an indirect way and also publishes the relevant data on the internet through JCSG's website.

1.5 Content of this Document

In order to describe how CTDT was developed and tested, this document first presents an overview of the primary software modules in Chapter Two. Chapter Three describes the technical details and design of the project and Chapter Four explains the results and evaluation. Chapter Five discusses additional work which can improve and extend the functionality of CTDT. Chapter Six briefly summarizes the outcomes of the project.
2. CUSTOMIZED TOOL FOR DATA TRANSFER (CTDT)

2.1 Purpose of the CTDT

The purpose of CTDT is to automate the data transfer from the GNF database, containing crystallization phase data, to JCSG's BART tracking database that stores data for all phases of genome sequencing. This tool complies with the agreement between JCSG and GNF that JCSG be able to extract only the agreed data from the GNF database in an automated way that does not present any risk to GNF's intellectual property rights. Therefore, the design of CTDT concentrated on developing a tool that can extract the data from GNF and deposits it in BART database in a customized way.

Fig 2.1 represents the flow of information between the major system components. The CTDT is basically divided into four modules. The first two were installed at GNF on a Windows 2000 workstation while the third and fourth were installed at JCSG on a UNIX server.

2.2 Module-1

Periodically staff at GNF and JCSG meet to decide on what data to be transferred from GNF to BART. As a result of these agreements, GNF personnel create queries to generate the data to be transferred.

Module-1 connects to an Access database, which has data to be transferred to BART, and retrieves the needed data from the database through multiple embedded SQL queries. Then it transforms the data into XML format and writes the outputs of different queries into different XML files. Then it sends all the files via email to an authorized user designated by GNF. The recipient of the email verifies the XML files for integrity of the data by opening those files either from email directly or by saving to some location.
depending on the Mail Application he/she uses. A snapshot of the email the user at GNF receives is shown in Fig. 2.2.

The present email recipient is using Microsoft Outlook, which has the capability of opening XML documents directly. The Module 1 script runs weekly (every Friday evening at 5.00 pm) by the Windows Schedule Tasks tool. The sample XML file generated by the Module-1 is shown in Fig. 2.3.
<DBI driver="ac">
<br />
<RESULTSET statement="SELECT ID, [GNF Crystal Plate ID] as PlateID, [Well ID] as WellID, Comment FROM [xtal hits gnf]">
<br />
<ROW>
<br />
<ID>152</ID>
<br />
<PlateID>SeMet5_4_01H1-0916</PlateID>
<br />
<WellID>C1</WellID>
<br />
<Comment>micro crystals, same as native</Comment>
<br />
</ROW>
<br />
<ROW>
<br />
<ID>632</ID>
<br />
<PlateID>W1-2532</PlateID>
<br />
<WellID>F11</WellID>
<br />
<Comment>small single crystals, fine screen</Comment>
<br />
</ROW>
<br />
</RESULTSET>
<br />
</DBI>

Figure 2.3 Sample XML file generated by Module-1

2.3 Module-2

This module has been converted into an executable and linked to a desktop icon. The recipient of the email will execute this module by clicking the desktop icon on his
terminal, which will run the script on the Windows 2000 network. This module takes all the XML files created/sent by the previous module and compresses each file and encrypts them using 448-bit encryption and then sends them to ftp.jcsg.org by File Transfer Protocol. The only user intervention is to click the icon on the desktop to invoke the executable script. This is done approximately once per week. The sample snapshot of the desktop having the icon for this executable is shown in Fig. 2.4.

![Figure. 2.4 The sample snapshot of the desktop](image)

2.4 Module-3

This module is another perl script, but runs on a UNIX server at the JCSG site. This module does not involve any human intervention. This module runs weekly (every Saturday morning at 10.00 am) by the Cron tool on a UNIX server. This module downloads all the compressed and encrypted XML files transferred from the GNF site to the JCSG site and then it decrypts and decompresses them. It connects to the BART
database and takes the XML formatted data from the files and harvests them into respective mapped fields and tables. A sample of populated table is shown in Fig. 2.6.

Figure. 2.5 A snapshot of the screen while Module-2 is executing

2.5 Module-4

This module creates a dynamic World Wide Web interface using a Perl/CGI script to the data transferred to BART from GNF. Users can access the web page from any Internet browser (www.jcsg.org/prod/gnfweb.html). The user can either run SELECT queries on this data and get the output on a web page in a tabular format, or they can look at the complete dataset transferred from GNF. This process will also allow a user at JCSG to use an easy to use browser to verify the data uploaded in BART. A snap shot of the web page is shown in Fig. 2.7. A sample output of a query is shown in Fig. 2.8.
Figure 2.6 A sample of populated BART table

**THE DATA TRANSFERRED FROM GNF**

Please enter your query here (SELECT statements only) here

If you want to see the Complete Data transferred from GNF,
Click below

Figure 2.7 A snap shot of [www.jcsg.org/prod/gnfweb.html](http://www.jcsg.org/prod/gnfweb.html)
Your SQL Query was: `select id, plateid, commnt from gnfdata where wellid LIKE 'A6%'

<table>
<thead>
<tr>
<th>ID</th>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>201</td>
<td>SeMet5_4_01H1-1724</td>
<td>micro crystals</td>
</tr>
<tr>
<td>203</td>
<td>SeMet5_4_01H1-3340</td>
<td>round shaped mini crystals</td>
</tr>
<tr>
<td>206</td>
<td>SeMet5_4_01H2-2532</td>
<td>micro crystals</td>
</tr>
<tr>
<td>210</td>
<td>SeMet5_4_01W1-0108</td>
<td>crappy micro crystals</td>
</tr>
<tr>
<td>218</td>
<td>SeMet5_4_01W1-3340</td>
<td>micro crystals</td>
</tr>
<tr>
<td>542</td>
<td>H1-3340</td>
<td>round shaped mini crystals</td>
</tr>
<tr>
<td>545</td>
<td>H2-2532</td>
<td>micro crystals</td>
</tr>
<tr>
<td>549</td>
<td>W1-0108</td>
<td>crappy micro crystals</td>
</tr>
<tr>
<td>555</td>
<td>W1-3340</td>
<td>micro crystals</td>
</tr>
<tr>
<td>561</td>
<td>W2-0916</td>
<td>micro crystals</td>
</tr>
<tr>
<td>564</td>
<td>W2-2532</td>
<td>micro crystals</td>
</tr>
</tbody>
</table>

Figure 2.8 A snap shot of a query output
3. SYSTEM DESIGN

3.1. Design Overview

The CTDT automated the process of data transfer from GNF database to BART totally except that of the email recipient at GNF intervening this transfer binding to GNF’s policies for any kind of data going out from GNF (Fig. 2.1). The CTDT was developed using Perl, CGI/Perl scripts and XML. The reason for choosing Perl language for developing the CTDT is it's efficiency and ease of use in handling variables (data structures) of both XML and relational databases. [Wall 1996] [Schwartz 1997]. It was installed on a Microsoft Windows NT Workstation located in GNF's intranet. This tool extracts the required data from GNF's database, and parses the data into an XML document that is formatted in a way to harvest the data into BART in an easy and efficient manner [Shanmugasundaram 2000] probing integrity/internal consistency of GNF data. The reason for using XML is this is the most recommended standard [Dick 2000] [Shanmugasundaram 2000] for representing data for data processing [Florescu 1999] as the container of the variables from relational database [Deutsch 1999]. After creating this XML document, it will be transferred automatically to the JCSG's authorized personnel's home directory or any other designated directory to which that person has access. FTP will be used as a protocol to transfer the XML file after compression and encryption on a predefined periodic frequency. The encryption and compression will help in easier and faster transfer of the file in a secured manner.
The CTDT is developed modularly to accomplish the task of data transfer from GNF database to BART. Various modules are distinctive in terms of their purposes and also the platforms (i.e. operating systems, environment, etc) they are installed and run on. (Fig. 3.1 & 3.2).

**Figure 3.1. Context level Data Flow Diagram for CTDT**

**Figure 3.2. Level-1 Data Flow Diagram for CTDT**

The tool installed at GNF has two modules in it. Module-1 does the extraction of the required data from the Access database. Module-2 does the compression, encryption and transfer of the data to JCSG’s FTP site. The tool installed at JCSG also has two
modules in it. Module-3 retrieves the XML documents transferred from GNF site, decrypt and decompress them, and then parses those XML files into BART. Module-4 provides an interface for this data on World Wide Web.

3.2. Module-1

3.2.1. Description

This module is installed on a Microsoft Windows NT Workstation located in GNF’s intranet. It connects to an Access 2000 database at GNF and extracts the data from the database using static queries. Then it transforms the extracted data into XML format and writes the output of different queries into different XML files on the drive where the Module-1 script is run. Then it sends copies of all the files to an authorized user designated by GNF by email. The recipient of the email verifies the files for integrity by opening those files either from email directly or by saving to some location depending on his/her Mail Application. The present email recipient is using Microsoft Outlook, which has the capability of opening XML documents directly. The Module 1 script runs weekly (every Friday evening at 5.00 pm) by the Windows Schedule Tasks tool. For any reason, if the task does not run, another script (erronotify.pl), which runs on every Monday, notifies the user at GNF about the error through email and advises him/her to run Module-1 again by clicking the desktop icon, which is linked to the wrapped up executable Module-1. All the errors are logged in a text file (gnferrorlog.txt) in the same directory where the script will be run. See Fig. 3.3.

Prerequisites for this module to run are ODBC Data Source Names (DSN) for the databases from which it is going to extract the data, a Perl interpreter, and various Perl modules (DBI, DBD::ODBC, DBIx::XML_RDB, Mail::Sender).
The first two processes (functions) from Fig. 3.3. represent Module-1 and the next two represent Module-2.

---

**Figure 3.3. Level-2 Data Flow Diagram for Tool-1 (Modules 1&2)**

### 3.2.2. Description of Major Components

#### 3.2.2.1. Systems (equipment, languages, software packages) used in this module

- A Windows NT Workstation on a PC or similar machine.
- MS Access-2000
- Active Perl v.5.6.1. This will be installed on NT machine from Active Perl's web site [Activestate 2001].
- DBI [1.14], DBD::ODBC [0.28], DBIx::XML_RDB, Mail::Sender modules of Active Perl. These modules are installed from Symbolstone's web site [Symbolstone 2001] and Comprehensive Perl Archive Network (CPAN) web site.
- ODBC Data Source Names for the databases to be connected.
- Windows Schedule Tasks tool.
3.2.2.2. Systems needed for normal use of the CTDT products once completed

Same as the above section.

3.2.2.3. Database table and schemas

Schema of GNF's MS Access database

3.2.2.4. Input, processing, and output of programs and functions

Function-1# data_extractor()

- Purpose: This function connects the module to the GNF's Access database through DBIx::XML_RDB module of the Perl language. It accepts SQL queries, database name and DSN from the variables embedded in the script (this allows the changing the database name or the queries or DSN improving the portability of this tool), and then extracts the appropriate data from the appropriate database. Then it parses the results into XML format.

- Inputs: names of tables and fields (in an SQL query), DSNs for the databases.

- Outputs: Transformed XML data (Query results) and an Email with XML files as attachments.

Function-2# email_creator()

- Purpose: This function creates an email and sends those files to a user at GNF.

- Inputs: XML files and Email address of GNF user.

- Outputs: sends out email with XML files as attachments.

3.2.2.5. Protocols and interfaces between the components

SMTP, DBIx::XML_RDB, Mail::Sender, Windows-ODBC drivers

3.2.2.6. File formats

Perl files (.pl) and XML files.
3.3 Module-2

3.3.1. Description

This module was installed on a Windows machine. It has been converted into an executable and linked to a desktop icon. The recipient of the email executes this module by clicking the desktop icon on his terminal, which runs the script over the Windows 2000 network. This module takes all the XML files created/sent by the previous module and compresses each file using Compress::Zlib module. Then it encrypts the compressed files by 448-bit encryption using Blowfish algorithm. Here Crypt::Blowfish module is used to implement the algorithm. Compression followed by encryption makes the files more immune to sniffing than the other way. It then sends the files to ftp.jcs.org by File Transfer Protocol using Net::FTP module. The only user intervention here is to click the icon on the desktop to invoke the executable script. See Fig. 3.3. for data flow in and out of Module-2. The last two processes represent this module. It also creates an error log file if some errors occur during the process. See Fig. 3.3.

3.3.2. Description of Major Components

3.3.2.1. Systems (equipment, languages, software packages) used in this module

- A Windows NT Workstation on a PC or similar machine.
- Active Perl v.5.6.1. This will be installed on NT machine from Active Perl's web site [Activestate 2001].
- Net::FTP, Compress::Zlib and Crypt::Blowfish modules of Active Perl. These modules are installed from CPAN web site.

3.3.2.2. Systems needed for normal use of the CTDT products once completed

Same as the above section.
3.3.2.3. Database table and schemas

- Schema of GNF's MS Access database

3.3.2.4. Input, processing, and output of programs and functions

This module is mainly divided into two key functions as listed under,

Function-1# compr_encryptor()

- Purpose: This function compresses the XML file and then encrypts the file using Blowfish encryption algorithm for the safer transfer of the file (data) across the networks.
- Inputs: XML files.
- Outputs: Compressed and Encrypted XML files.

Function-2# transferrer()

- Purpose: This function will transfer the compressed and encrypted XML files from the GNF site to the authorized directory on ftp.jcsng.org. The transfer will be done by FTP after compression and encryption.
- Inputs: Compressed and Encrypted XML files.
- Outputs: Transferred data file.

3.3.2.5. Protocols and interfaces between the components

Net::FTP, Compress::Zlib, Crypt::Blowfish

3.3.2.6. File formats

Perl files (.pl) and encrypted and compressed XML files.
3.4 Module-3:

3.4.1. Description

This tool is installed and runs on a UNIX server at JCSG. It is scheduled to run every Saturday at 10 AM. The scheduling is accomplished by using the CRON utility on the Unix server. It moves the transferred GNF files by Module-2 from the JCSG’s FTP site to the folder where this module is run. Then it decompresses the files, decrypts them, connects to BART and then parses the data into proper tables in BART. It also creates an error log file if some errors occur during the process. If for some reason the files are not deposited at JCSG’s FTP site before Saturday 10 AM, this module generates an email for a GNF representative and a JCSG representative notifying them about the failure of file transfer. See Figure 3.4 for the Data Flow Diagram for the Modules 3 & 4. The first two processes represent Module-3.

3.4.2. Description of Major Components

3.4.2.1. Systems (equipment, languages, software packages) used in this module

- SUN sparc station (server) running UNIX.
- Perl interpreter for UNIX.
- DBI [1.14], DBD::Oracle, DBIx::XML_RDB, XML::XPath, Crypt::Blowfish, Compress::ZLib modules of Perl.
- Oracle 8i database.
- CRON utility of UNIX.
3.4.2.2. Systems needed for normal use of the CTDT products once completed

Same as the above section.

3.4.2.3. Database table and schemas

Schema of BART database. (Fig. 3.1) [Bill 2001].

3.4.2.4. Input, processing, and output of programs and functions

This module of the CTDT is mainly divided into 3 key functions. They are listed in the following text.

**Function-1# decrypt_decompress()**

- It decrypts and decompresses the XML files transferred from GNF site using Blowfish algorithm (Crypt::Blowfish module) and Decompression algorithm (Compress::Zlib module) and outputs the XML files for parsing into BART.
- Inputs: Encrypted and Compressed XML files.
- Outputs: Decrypted and decompressed XML files.
Function-2# loader()

- Purpose: This function connects the module to the BART database through DBD::Oracle and DBIx::XML_RDB modules of Perl language. It then parses the XML data into variables those can be placed into a SQL INSERT statement and then executes the SQL statement to populate the appropriate fields and tables in BART.

- Inputs: Decrypted and decompressed XML files.

- Outputs: Populated BART database.

3.4.2.5. Protocols and interfaces between the components

DBI [1.14], DBD::Oracle, DBIx::XML_RDB

3.4.2.5. File formats

Perl files (.pl), decrypted & decompressed XML files and error log files.
Figure 3.5. Schema of BART database [West 2001]
3.5. Module-4

3.5.1. Description

This module provides a dynamic web interface through a CGI/Perl script and a HTML page (http://www.jcsg.org/scripts/prod/gnfweb.html), which retrieves the transferred data from GNF to BART. This allows the user to either query the database by entering a valid SQL SELECT statement and/or provides another option to look at the whole data simply by clicking an appropriate button on the web page. This data will be available to the internal and external sites. If the error occurs at the browser interface, CGI::Carp module will log it into a log file on the web server. See Fig. 3.5 for the Data Flow Diagram for this module. A snap shot of the web page and the transferred data is shown in Fig. 2.7 & 2.8.

3.5.2. Description of Major Components

3.5.2.1. Systems (equipment, languages, software packages) to be used in this module

- SUN sparc station (server) running UNIX and APACHE web server.
- Perl interpreter.
- A Web browser (Internet Explorer or Netscape).
- Oracle 8i database - BART.
- CGI, CGI::Carp modules of Perl.

3.5.2.2. Systems needed for normal use of the CTDT products once completed

A Web browser (Internet Explorer or Netscape).

3.5.2.3. Input, processing, and output of programs and functions
Function-1# Publisher()

- **Purpose:** This function posts the data transferred from GNF from BART whenever a user requests for data from a web browser.
- **Inputs:** A valid SQL `SELECT` statement or a click of a button on web page to retrieve whole data.
- **Outputs:** Results of the query or the whole data

3.5.2.4. **Protocols and interfaces between the components**

Common Gateway Interface (CGI), CGI::Carp modules of Perl and Hyper Text Transfer Protocol (HTTP).

3.5.2.5. **File formats**

CGI/Perl scripts and web pages (.html).
4. EVALUATION AND RESULTS

The CTDT is currently implemented at JCSG and GNF. The modules were tested and evaluated in different stages. The module-1 aims at extracting the specified data from GNF database, parse the data into an XML files, and mail those files to a designated user at GNF for verification. The user verifies the data and then executes the next module (module-2). As module-1 is run under Windows Task Scheduler, if the schedule fails besides the notification by the operating system itself, there is another script installed on the same machine, which informs the user at GNF about the failure of the task, by email. This module is tested fine in real life. There were minor additions to this module to what it was proposed. An extra script is added to take care of scheduling failures. An extra step in the process (mailing the XML files to the user at GNF) was added as per the regulations and policies at GNF. Authorized personnel of GNF evaluated this module.

The module-2 compresses and encrypts the XML files and the transfers the same files to JCSG, is tested for its functionality. The 448-bit encryption was achieved successfully. The compression of the XML files was achieved up to 80%. The encryption and compression part can be evaluated by testing for the complexity of the encrypted file for decrypting and for the degree of compression of the file comparing pre and post compression sizes of the file. The other part of the module, which involves the transfer of an encrypted and compressed file to a specified directory at JCSG’s FTP site, was tested at JCSG. An authorized user at JCSG tested the module successful. There were some minor changes in the implementation of this module from what it was proposed. The
transfer protocol was changed from SSH to regular FTP by the instructions from JCSG. The errors are logged to a text file. This error log file is verified for any errors.

The module-3, which does the decryption and decompression of the XML files transferred from GNF was tested successful as the decompressed and decrypted files were exactly same when compared to the original XML files. There was loss of data during the process. The next part of the module, which parses the data from XML files into BART database, was tested at JCSG. The personnel at JCSG tested this module for its functionality by using some sample files. The database administrators of the BART evaluated the data parsed into BART by checking the accuracy and integrity of the data. The errors at the module execution time are logged to a text file. This error log file is verified for any errors. There were some minor changes in the implementation of this module from what it was proposed. After the module-2 transfers the files to FTP site, instead asking the user to browse for the files, JCSG created a directory exclusively for the files transferred from GNF.

The module-4, which prepares a dynamic web interface to publish the transferred data from GNF, was tested by the internal and external users for its usability. This web page and CGI script was tested fine from Internet browsers (Internet Explorer and Netscape Navigator). The published data was compared with the original data for the accuracy and availability. Here the errors from the browser are logged on to a server file by the web server itself using the CGI::Carp module of Perl.
5. CONCLUSION

The CTDT was conceived and developed for transferring the data collected during the crystallization phase of genome sequencing, on a periodic basis at the Genomics Institute of the Novartis Research Foundation (GNF) to the tracking database which stores the data from all the phases of genome sequencing, Bioinformatics Archival and Retrieval Technology (BART) database of JCSG. This data transfer was implemented in several sequential phases using different modules of this project, which were installed in heterogeneous platforms of the JCSG and GNF local area networks. As the last part of the project, the data was made available to the public via the JCSG web site.

The CTDT has accomplished the following tasks.

- Automated transfer of the data between two sites.
- Eliminated the risks concerning intellectual property rights of GNF and data security issues for GNF and JCSG.
- Extraction of required data from GNF database and parse it into an XML files.
- Uses efficient algorithms for encryption-decryption and compression-decompression of the XML data files.
- Secured and efficient transfer of the data (compressed and encrypted XML files) to appropriate place (directory) at JCSG.
- Easy harvest of data from decrypted and decompressed XML files into BART probing consistency and integrity of the data.
• Creation of error log files during the data transfer, in the event of any errors occur.

• Providing a dynamic web interface for the transferred data on the Internet via JCSG website.

• Saving the manpower for both GNF and JCSG besides avoiding the humanitarian errors likely to be introduced during the manual data transfer.
6. FUTURE WORK

The CTDT does the transfer between an Access database and an Oracle database BART, which is a tracking database that collects from different phases of genome sequencing. There are an increasing number of tracking databases especially in areas of scientific research. Most of them are usually on heterogeneous platforms. They all need a transfer tool like CTDT to transfer the data in a customized and tailored way for each organization. So these kinds of tools can be developed between other commercial relational databases like DB2, Sybase, MS SQL server etc. It can also be extended to text databases.
ACKNOWLEDGEMENTS

This project grew out of a series of dialogues and emails with Dr. Mark Miller and Mr. Bill West at Joint Center for Structural Genomics (JCSG)/ San Diego Supercomputer Center (SDSC). I thank them for their co-operation and support throughout my project.

I am also indebted to Dr. Stephen Dannelly who inspired me and encouraged me towards this project and for providing decisive and energetic support during the write-up stage, clearing the path towards project.

Many thanks to Dr. David Thomas and Dr. Michelle Moore whose editing suggestions and precise sense of language contributed to the final copy.

I would especially like to thank Dr. Paul Orser, David Andrus, Gale Stuart, Joyce Cloud, Kathy Winston of Office of Planning and Institutional Effectiveness, and other friendly staff at TAMUCC whose friendship and support have made it more than a temporary place of study for students from all around the world. I am very grateful for my years at the TAMUCC.

I am forever grateful to my parents, Rukmini and Laxminarayana, whose foresight and values paved the way for a privileged education, for their counsel and unconditional support at each turn of the road.

My wife Suman's patience, encouragement and love were essential for my joining and smooth sailing through the project. My brothers, sisters and other family members buffered my ups and downs with unquestioning support all through my life.
Last but not the least, I wish I could mention each of my friends individually, but will throw a large party instead.


[Miller 2001] Miller, M. Joint Center for Structural Genomics. Email from mmiller@sdsc.edu. (received on Sep. 06, 2001).


APPENDICES

1. Code for Module - 1

```
# module1.pl
#Author: - srikanth kasha
#this script is for connecting to Access db on local Win-machine at GNF and for creating
and writing the extracted data
#from queries to XML files and then sending the XML files to a user at GNF via email
for verification and confirmation
#to send the files to JCSG. It will also store the email attached files on the local machine.
use DBIx::XML_RDB;
use Mail::Sender;
#use warnings;
#use strict;

our $database = 'ac';
our $driver = "ODBC";
our $attr = {PrintError=>0,RaiseError=>1};

my %queries=(
    #this hash provides the user the capacity of entering any number of queries
    'JCSG_Crystal_Hit_Query'=>"SELECT PID, [Plate ID] as PlateID, [Well ID] as WellID, Temperature FROM [Xtal Plate Definitions GNF]",
);
&data_extractor($database,$driver);
&email_creator();

#this function enables the user to connect to any number of databases
sub data_extractor(){
    my $dbx = DBIx::XML_RDB->new($_[0], $_[1])|| &errorlog($DBI::errstr);
    my $key;
    my $value;
    my $filename;
    while(($key,$value)=each(%queries)){
        $dbx->DoSql($value);
        $filename=$key.".xml ";
        print $filename;
        open(XML, ">$filename") or die "couldnot open file for writing : $! \n";
        print XML $dbx->GetData;
        close XML or die "Error closing the file: $! \n";
    }
}

sub email_creator(){
    our $sender;
```
$sender = new Mail::Sender {smtp => 'sdsc.edu', from => 'kasha@sdsc.edu'};
$sender->MailFile({to => 'kasha@sdsc.edu', subject => 'JCSG files', msg =>
    "These are the files to be sent to JCSG. \nAfter verifying, please click
the icon on your desktop for executing the script (batch file) to FTP the
files to JCSG. \nThanks", file => 'JCSG_Crystal_Hit_Query.xml'}) or
die "$Mail::Sender::Error\n";
$sender->Close;
}

sub errorlog()
{
    open(F,">>gnferrorslog.txt");
    print F $_[0], `date`;  
    close(F);
    die "$\n";
}

 Jazeera Code for Module - 2
**********************************************************************************
#module2.pl
#Author-srikanth kasha
#Module to be installed at GNF site
#for connecting to Access db on a local NT machine, query the db, write the results to an
XML document, compress it and encrypt it
#and send it to JCSG site via FTP
use strict;
use warnings;
use Compress::Zlib;
use Crypt::Blowfish;
use Mail::Sender;
use Net::FTP;

our $encrypted;
our $x;
our
@queries=("JCSG_Crystal_Hit_Query.xml","JCSG_Transfer_to_SSRL.xml","JCSG_Set
up.xml","JCSG_Expressed.xml","JCSG_Cloned.xml");
foreach (@queries)
{
    our $compr=&compr_encryptor($_);
    &transferrer($compr);
}
foreach (@queries)
{
    `move $_ moved_gnf` || &errorlog($!);
}
sub compr_encryptor {
    #compression
    my $fn=$_[0];
    if (open(FILE,$fn)){
        undef $/;
        my $wf= <FILE>;
        my $len1= length($wf);
        print "length of the uncompressed XML file is : $len1 \n";
        print "compressing the file....\n";
        $x=compress($wf);
        my $len2= length($x);
        print "length of the compressed XML file is : $len2 \n";
        close(FILE);
    } else {
        &notify_errors($fn);
    }
    #writing the compressed file to another file in binary mode

    our $compr="compressed_".$fn;
    open(FILE1, ">$compr");
    binmode FILE1;
    print FILE1 "$x";
    close(FILE1);

    #encryption
    my $text = $x;
    my $key = pack("H56",
                  "higuys123ihavestartedworkingatjcsagsdscon14thjanuaryoftheyear2002");
    $encrypted = encrypt($key,$text) || die "Failed trying to encrypt the file...: $! \n";
    $compr="Encrypted_".$compr;
    open(FILE2,">$compr");
    binmode FILE2;
    print FILE2 "$encrypted";
    close(FILE2);
    print "448 bit (56 byte) encryption successful...\n";
    our $BLOCKSIZE = 8;

    #function for encryption
    sub encrypt {
        my ($key,$dat) = @_;  
        my $enc = "
        my $cipher = new Crypt::Blowfish $key;
        #as the Crypt::Blowfish algorithm restricts to use <=8 bytes per block
        return $cipher->encrypt($dat); 
    }
}
our $BLOCKSIZE = 8;
# for the tail of the file i.e. last read block ... if it is less than eight bytes pad
# it up to eight bytes
if ($BLOCKSIZE > 0) {
    my $l_dat = length($dat);
    for (my $i=0; $i < $l_dat; $i+=$BLOCKSIZE) {
        my $tmp = substr($dat, $i, $BLOCKSIZE);
        my $tmp2 = sprintf("%-" . $BLOCKSIZE . "s", $tmp);
        $enc .= $cipher->encrypt($tmp2);
    }
} # for encrypting the rest of the file
else {
    $enc .= $cipher->encrypt($dat);
}
return $enc;
}

sub transferrer()
{
    ## ftp
    our $ftp;

    our $compr1=$_[0];
    $ftp = Net::FTP->new("ftp.jcsg.org", Debug => 0);
    # logging on to ftp.jcsg.org as anonymous to put the file there
    $ftp->login("anonymous",'kasha@sdsc.edu');
    # getting into the authorized folder to store the file
    $ftp->cwd("/pub/incoming/exchange");
    # for binary transfer of the file
    $ftp->binary();
    my $we =$ftp->pwd();
    print $compr1;
    $ftp->put($compr1) or die "Error FTPing the file: $! \n";
    $ftp->quit;
    my $len3= length($encrypted);
    print "Successfully FTPed the encrypted file of length $len3 \n";
    print "\n$we \n"
}

sub errorlog()
{
    open(F,">>gnferrorslog.txt");
    print F $_[0];
    close(F);
    die "$_ \n";
}
sub notify_errors {
    our $sender;
    our $f = $_[0];
    ref ($sender = new Mail::Sender({from => 'kasha@sdsc.edu', smtp => 'sdsc.edu'})) or die "$Mail::Sender::Error\n";
    ref ($sender->MailMsg({to => 's0k41676@penguin.tamucc.edu', subject => 'Error while transferring files to JCSG', msg => "Errors occurred while running the weekly task transferring the files to JCSG. The file: $f is not transferred. So please run the --- mailer task--- manually by clicking the specified icon on your desktop. \nThanks"))) and print "Mail sent successfully."
    ) or die "$Mail::Sender::Error\n";
}
__END__

**********************************************************************
3. Code for Module - 3
**********************************************************************
#!/usr/local/bin/perl
#Module3.pl
#Author - srikanth kasha
#Module to be installed at JCSG site
#this module will get the file from ftp.jcsg.org and decrypt it and decompress it then parse that XML file and store it in
#appropriate tables in BART.
use Mail::Sender;
use Compress::Zlib;
use DBI;
use XML::XPath;
$XML::XPath::Debug = 1;

use lib "/sdsc/local/generic/apps/perl/lib/site_perl/";
use lib "/sdsc/local/generic/apps/perl/lib/site_perl/Crypt";
use Crypt::Blowfish;

$ENV{"ORACLE_HOME"} = "/usr/local/apps/oracle/home";
$ENV{"ORACLE_PATH"} = "/usr/local/apps/oracle/home/bin";
$ENV{"ORACLE_SID"} = "BART";
$ENV{"TWO_TASK"} = "BART";

$user="xxxxxxxx";
$pass="xxxxxxxx";
my $database = 'bart';
#for multiple files
@queries=("Encrypted_compressed_JCSG_Crystal_Hit_Query.xml");
chdir("gnfxmlfiles");
foreach (@queries){
  `mv /misc/jcsg_ftp/pub/incoming/exchange/$_ .`;
  if(open(FILE,$_)){
    $decrypt=&decrypt_decompress($_);
    &loader($decrypt);
  }
  else {
    &notify_errors($_);
  }
}
chdir();

sub decrypt_decompress(){
  $BLOCKSIZE = 8;
  $file=$_[0];
  open(FILE, $file);
  undef $/;
  $encrypted= <FILE>;
  my $key = pack ("H56",
                   "higuys123ihavestartedworkingatjcsgdscon14thjanuaryoftheyear2002");
  my $decrypted = decrypt($key,$encrypted);

  sub decrypt {
    my ($key,$dat) = @_; 
    my $dec = ";
    my $cipher = new Crypt::Blowfish $key;
    $BLOCKSIZE = 8;

    if ($BLOCKSIZE > 0) {
      my $l_dat = length($dat);
      for (my $i=0; $i < $l_dat;$i+=$BLOCKSIZE) {
        my $tmp = substr($dat,$i,$BLOCKSIZE);
        $dec .= $cipher->decrypt($tmp);
      }
    } else {
      $dec .= $cipher->decrypt($dat);
    }
    $dec =~ s/ \s+//; #remove trailing spaces
    return $dec;
  }

}
print "uncompressing the file...\n";
$uc=uncompress($decrypted);
my @sf=split('_', $file);
$filename=$sf[2]."_".$sf[3]."_".$sf[4]."_".$sf[5];
open(FILE1,">$filename");
print FILE1 "$uc";
print $filename;
close(FILE1);
return $filename;
}

sub loader{
    $x=$_[0];
    my $dbh = DBI -> connect('dbi:Oracle:bart', $user, $pass) || die $DBI::errstr;
    my $xp = XML::XPath -> new(filename => '$x')|| die "could not find your file\n";
    my $sth = $dbh -> prepare(qq{INSERT INTO GNFDATA VALUES(?, ?, ?, ?)});
    # loop through the records
    foreach my $row ($xp -> findnodes('/DBI/RESULTSET/ROW')) {
        my $ID = $row -> find('PID') -> string_value;
        my $PlateID = $row -> find('PlateID') -> string_value;
        my $WellID = $row -> find('WellID') -> string_value;
        my $Comment = $row -> find('Temperature') -> string_value;

        # insert into the db (using placeholders)
        $sth -> execute($ID, $PlateID, $WellID, $Comment) || die $DBI::errstr;
    }
    $dbh -> disconnect;
    print "Successfully loaded the tables in BART!!! \n";
}

sub errorlog{
    open(F,">>gnferrorslog.txt");
    print F $_[0];
    close(F);
    die "$\n";
}

sub notify_errors{
    $f=$_[0];
    ref ($sender = new Mail::Sender({from => 'kasha@sdsc.edu',smtp => 'sdsc.edu'})) or die "$Mail::Sender::Error\n";
    (ref ($sender->MailMsg({to =>'s0k41676@penguin.tamucc.edu', subject => 'Some errors occurred in GNF file transfer',
    msg => "Errors occurred while retrieving the files from GNF. the file:  $f is not transferred. So please send GNF a mail informing the transfer error.\nThanks"})) and print "Mail sent succesfully.";
4. Code for Module - 4

#!/usr/local/bin/perl
#gnfweb.cgi
#Author: Srikanth Kasha
# Get the modules path for JCSGINITS
use lib "/misc/www/projects/jcsg/modules";
# Determine which environment is being used Development, Testing or Production
use JCSGINITS;
&JCSGINITS::set_env_vars;
&JCSGINITS::where_am_i;
$scripts_dir = "/scripts/" . JCSGINITS::dir_loc;
use CGI qw/:standard :html3 :netscape/;
use CGI::Carp;
use DBI;
use JCSGLOGONOBJECT;
use JCSGUTILS;

$logon_obj = new JCSGLOGONOBJECT();
$logon_obj->logon_oracle();
# Maximum length of the document.
my $max_len = 2000000;
$logon_obj->{"dbh"}->{LongReadLen} = $max_len;
$logon_obj->{"dbh"}->{LongTruncOk} = 1;

$dataIn = new CGI;
$dataIn->header();
##$requestType = $dataIn->param('requestType');
$sql = $dataIn->param("sql");

if (($sql eq "") || ($sql !~ /^SELECT/i)) {
    &printSearchForm();
    exit;
} else{
    $dataObject = executeSQLStatement($sql);
    @dbRows = &getDBRows($dataObject);
if ($sql =~ /^SELECT/i)  {
    print qq!
<HTML>
<HEAD>
<TITLE>SQL results</TITLE>
</HEAD>
<BODY BGCOLOR = "FFFFFF" TEXT = "000000">
<CENTER>
<p><TH><i>Your SQL Query was</i> : <b> $sql</b><br></TH></p>
<TABLE BORDER = "1">
    foreach $rowReference (@dbRows)
    {
        foreach $columnReference (@$rowReference){
            print qq!<TR>!
            foreach $column (@$columnReference)
            {
                print qq!<TD>$column</TD> \n!
            }
            print qq!</TR>!
        }
    }
    print qq!</TABLE>
    </CENTER>
</BODY>
</HTML>!
exit;
} else
{
    print "Your SQL Query has been processed, please hit the back button and submit a SELECT to see the changes!";
}
$logon_obj->kill_db_handle();

sub executeSQLStatement{
    my ($sql) = shift;
    $dataObject = $logon_obj->{dbh}->prepare($sql) || push(@error, $logon_obj->{dbh}->errstr);
    $dataObject->execute() || push(@error, $dataObject->errstr);
    return $dataObject;
}

sub getDBRows{
    my ($dataObject) = shift;
}
return $dataObject->fetchAll_arrayref();

})

sub printSearchForm {
  print qq!
  <HTML>
  <HEAD>
  <TITLE>SQL Query for GNF data</TITLE>
  </HEAD>
  <BODY BGCOLOR = "FFFFFF" TEXT = "000000">
  <FORM METHOD = "POST" ACTION = "gnfweb.cgi">
  <TABLE BORDER = "1">
  <TR>
  <TH>There were syntax errors in the SQL query entered. Please enter your query again</TH>
  <TD><INPUT TYPE = "TEXT" SIZE = "40" NAME = "sql"></TD>
  <TD><INPUT TYPE = "SUBMIT" NAME = "requestType" VALUE = "Submit SQL"></TD>
  </TR>
  </TABLE>
  </FORM>
  </BODY>
  </HTML>
  !;
}