Application of real time database to LAMOST control system

Lingzhe Xu [1], Xinqi Xu

National Astronomical Observatories / Nanjing Institute of Astronomical Optics & Technology, Chinese Academy of Sciences

188 Bancang Street, Nanjing 210042, P.R.China

ABSTRACT

The QNX based real time database is one of main features for Large sky Area Multi-Object fiber Spectroscopic Telescope's (LAMOST) control system, which serves as a storage and platform for data flow, recording and updating timely various status of moving components in the telescope structure as well as environmental parameters around it. The database joins harmonically in the administration of the Telescope Control System (TCS). The paper presents methodology and technique tips in designing the EMPRESS database GUI software package, such as the dynamic creation of control widgets, dynamic query and share memory. The seamless connection between EMPRESS and the graphical development tool of QNX's Photon Application Builder (PhAB) has been realized, and so have the Windows look and feel yet under Unix-like operating system. In particular, the real time feature of the database is analyzed that satisfies the needs of the control system.

Keywords: Astronomical telescope，real time, database, GUI, dynamic query, share memory, LAMOST

1. INTRODUCTION

LAMOST is one of a few large scientific and engineering projects in the national Ninth Five-Year Plan in China. When the project is completed the telescope will boast its most wide field of view and most efficient spectroscopic observation in all the astronomical optics telescopes of 4-meter-class and above in the world

The TCS of LAMOST has adopted QNX OS as its high level software development platform. QNX is one of a few excellent and highly regarded real time OSs in the world. The Real Time Database (RTD) was further developed on the platform with EMPRESS tool provided by QSSL, a Canadian software company. The application of RTD to LAMOST conformed to the tendency for contemporary astronomical telescopes in the world, yet was a real debut for astronomical telescopes ever built in China.

During the R&D stage of RTD we encountered problems with EMPRESS because it did not facilitate a GUI, although PhAB, one of the QNX's user tools, did provide a GUI yet lacking enough functions. On the other hand, an indisputable trend was that because of the Windows's domination people were getting used with Window's look and feel environment. Moreover, most users of our RTD will be astronomers, who are primarily interested in statistical reports with data and graphs while less skilful in manipulating data in non-GUI environment. Therefore, we determined to develop RTD with Window's look and feel, particularly the functions of dynamic table and graph creation, to give the user a friendlier interface.

Another distinguishing feature of RTD is its real time nature, which is required for the online quality analysis, historic recording of various kinds of the telescope's status, instruments' diagnosis, meteorological conditions and station environment monitoring, etc. These are all means of enhancing the telescope's service rating and gaining scientific output. We incorporated these time dependent factors into the consideration of RTD design.

[1] Correspondence: Email: xqxu@nairc.ac.cn; Telephone & Fax: 86 25 8540 5562

Painstaking effort led to a successful development of the package that contained 13000 lines of C listings. The package was demonstrated in the institute and received positive response.

2. SYSTEM PLATFORM

The system platform has temporarily been built with a four-PC LAN, one of which is COMPAQ AP500 and the other three are NEC PIII500. All the PCs are set up with QNX4.25 OS, and the EMPRESS v8.60 trial version is installed on one of the NECs as the database server. This configuration is just for our lab test. Since the network is hierarchical and extensible it is easy to hook up more PCs for LAMOST reality.

QNX as a distributed real time OS is the forerunner of the microkernel, which consists of a small kernel in charge of a group of cooperation processes. QNX OS is ideal for real time applications. It provides multitasking, priority-driven preemptive scheduling, and fast context switching, all essential ingredients of a real time system. When QNX is used to serve a distribute application, it uses its built-in fleet protocol so as to make the network resources transparent from any node-user's perspective.

EMPRESS database is one of the few databases that can support QNX4.25. EMPRESS v8.60 supports the standard distributed relation database, using fleet protocol under QNX and working with ODBC. The RTD for LAMOST was developed under QNX OS with EMPRESS DBMS.

3. USER INTERFACE STYLE

The design realized the Windows' style under QNX OS. Such as mouse manipulation, menu operation, table dragging, multi-resolution selection, hot key and automatic creation of statistic graphs, etc. All these database operation conveniences with Windows' feel and look are realized in RTD by means a large number of C codes running in the background, which cost painstaking effort to develop and debug.

4. DATABASE PLATFORM ARCHITECTURE

Since the EMPRESS has no GUI tools and the PhAB is not able to use the functions of EMPRESS we adopted the database platform architecture as shown in figure 1, which makes a smooth connection between the EMPRESS and

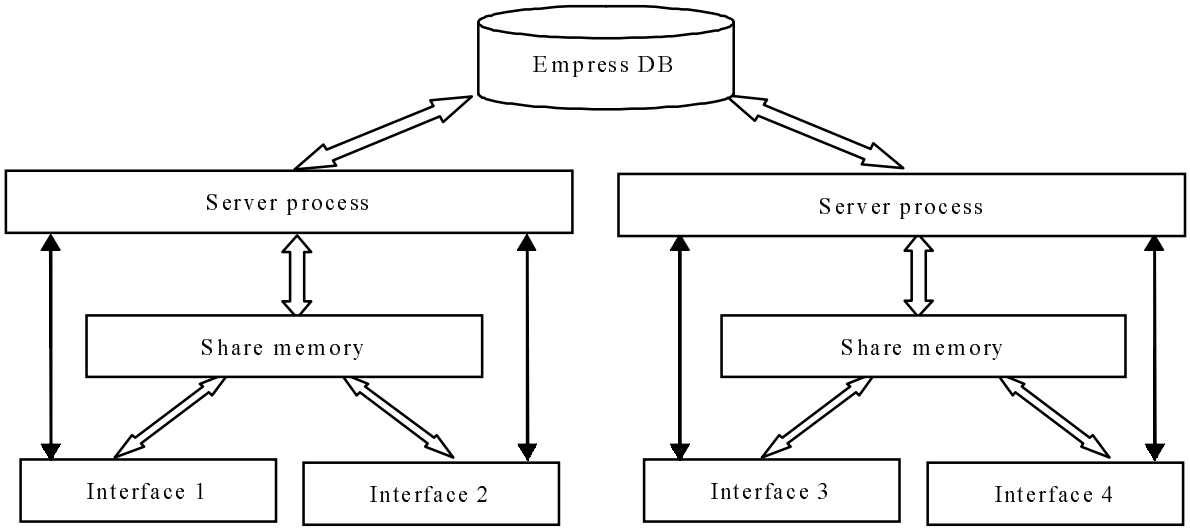

Figure 1 Architecture of the database platform

PhAB. The interface process was developed under PhAB using Watcom C code, and the server process was developed under QNX shell using Watcom C too. They communicate through the share memory. For example, the interface process accepts a read command input by the user, and then sends a message to the server process. The server process operates the database accordingly and writes a data to the share memory. Finally the interface process reads the data from the share memory. Thus the read cycle is completed.

It is worthy of mention that parallel channels exist, meaning a servo process running on each distributed node and with a share memory each between the server process and its corresponding interfaces. Thus the parallel feature makes it possible for simultaneous data read or write depending on the requirements of the input commands. Besides, figure 1 also shows that a server process can provide services for several interface processes at the same node.

5. FUNCTION OF THE APPLICATION PROGRAMMING

The main functions of the application are Create Table, Alter Table, Drop Table, Create Index, Drop Index, Execute SQL, Create Form with Column Selected, Adjust Form Column, Adjust Form Font, Adjust Form Color, Adjust Form Size, Adjust Form Width, Choose Query Condition, Choose Query Sequence, Create Graph with Column Selected, Set Graph X-Y Coordinates, Set Graph Title, Set Graph Size, Insert Data, Delete Data. These functions accommodate the user with major application features of the DBMS. And the user is able to create customized forms and graphs by means of the application GUI and to save them, as well as to conduct data analysis.

6. TECHNICAL REALIZATION

6.1 Database service process

The Database Service Process (DSP) gets a message from the interface process, and then interacts with the database. The DSP processes the message and abstracts the SQL command form it. By analyzing the key words in the command the DSP determines the type of SQL command. For executive type of SQL such as insert, delete, update, etc. the DSP is implemented and directly returns the status to the interface process. On the other hand, if the SOL is a select type we must adopt a dynamical SQL technology to realize the select request. The adoption of so called dynamical technology is simply because that the number of columns and the types of columns are unknown before the query is conducted. The dynamical SQL declares a data structure as bellow and reads/writes data from/to the database.

```
EXEC SQL BEGIN DECLARE SECTION;
        char     sql_str[1024];          /* string to contain SQL statement      */
        char     dname[33][25];          /* for storing item names               */
        int      dtype[25];              /* for storing data type of items       */
        int      da_num;                 /* number of active descriptor areas    */
        int      area;                   /* descriptor item area                 */
        char     *pstr;                  /* for getting string data              */
        long     vlong;                  /* for getting integer data             */
        double vdouble;                  /* for getting float data               */
        int      i;
        typedef struct
        {
            long size;
            char data[1];
        } bulk;
        bulk     *pbulk;                 /* for getting bulk data                */
        short    ctrl;                   /* control variable for null checking   */
 EXEC SQL END DECLARE SECTION;
```

6.2 InterProcess Communication (IPC)

There are a number of different processes being involved in the communications, such as the communication between the service process and interface process, the interaction between the service process and the share memory and the interaction between interface process and share memory. In the development of the database application package we employed both synchronous communication and asynchronous communication to complete the data transactions.

6.2.1 Message Passing

The communication between the interface process and the service process is carried out via messages using C language functions, Send() and Receive(). In QNX, a message is packet of bytes that's synchronously transmitted from one process to another. Figure 2 outlines a simple sequence of events in which two processes, Process A and Process B, use Send(), Receive(),and Reply() to communicate with each other. Initially Process A sends a message to Process B by issuing a Send() request to the Microkernel. At this point, Process A becomes SEND-blocked until Process B issues a Receive() to receive the message. Then Process B issues a Receive() and receives Process A's waiting message .Process A changes to a REPLY-blocked state. Since a message was waiting, Process B doesn't block. Finally Process B completes the processing associated with the message it received from Process A and issues a Reply(). The reply message is copied to Process A, which is made ready to run. A Reply() doesn't block, so Process B is also ready to run. Who runs depends on the relative priorities of Process A and Process B.

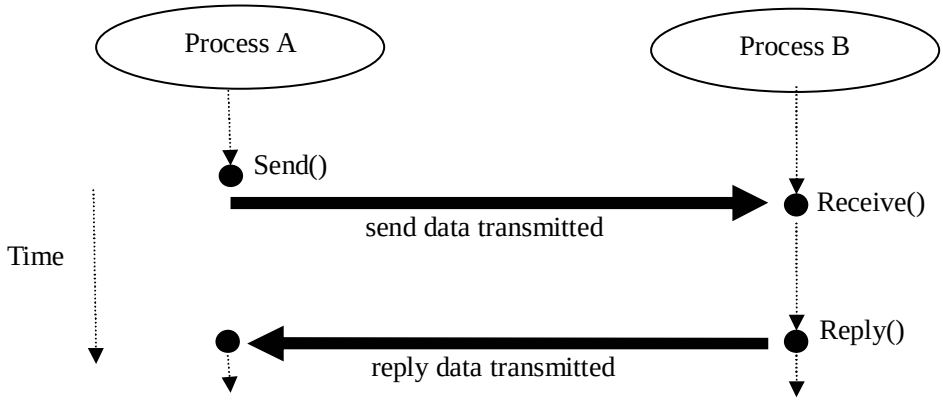

Figure 2 Communication between Process A & Process B via message

6.2.2 Share Memory

The database server process gets the data from database via share memory. The share memory provides a fastest data buffer for IPC. Once the share memory buffer is mapped onto the process-space the communication among corresponding processes will no longer involve the kernel. However, it is vital to establish synchronization for one process to write data to the buffer and another to read the data from the buffer. A number of such synchronization techniques exist, for example, mutually exclusive lock, conditional variable, read-write lock, record lock, semaphore and so on. We have adopted the semaphore in our programming. The server uses a semaphore to get the right to access the share memory, then maps an address pointing to the share memory, and finally writes data to the address. This is done, the server sets a semaphore to inform the client, and the client then gets data from the share memory.

Four main functions, Shm_open, Mmap, Sem_post and Sem_wait, are involved in the share memory operation. The shm_open function designates a name argument to create a new share memory or to open an existing share memory. The Mmap function maps the share memory to the calling-process address. The Sem_pos function and Sem_wait function are used to control the semaphore. The Sem_post increases the semaphore and the Sem_wait decreases the semaphore. When the semaphore is zero, the process, which gets the semaphore, will be blocked. In our practice, the share memory is divided into N parts. Two semaphores are set, one is read-semaphore and the other is write-semaphore. The read-semaphore is initialized 0, and the write-semaphore is initialized N. Each time the server writes a part of memory, write = write –1 and read =read + 1, then the pointer points to the next part of memory. Each time the interface process reads a part of memory, write = write +1 and read = read –1, then the pointer points to the next part of memory. When the n parts

of memory are written if write > 0 then the pointer points to the first part of memory. When the N parts of memory are read if read > 0 then the pointer points to the first part of memory. When write = 0 the server process is blocked, and when read = 0 the interface process is blocked. The semaphore implementation has realized the mutually exclusion on the share memory operation, increased the throughput and eliminated memory overflow.

6.3 Interface programming

6.3.1 Dynamical control widget and combo control widget

The user interface was developed under PhAB environment. Because the PhAB lacked powerful graphic programming tools, which made it difficult to develop a complicated GUI. Therefore we had to develop the application by using a series of programming techniques. For the sake of programming flexibility and averting the restriction of the inefficient PhAB's graphic tools we used dynamical control widgets and further built them into sophisticated combo widgets. For example, the dynamically created tables are made up of three control widgets Ptlist, PtDiider and PtButton. Every head entry of the table column is PtButton control widget. The tables are created dynamically, which is a smart way to deal with the un-prediction of grammatical generated user statement.

During the package development for a frequently called control widget the best way to implement was un-doubtfully to utilized control-widget store, which provided the possibility of once-created and multi-called. The call-back function could be preprogrammed and immediately become effective after the calling.

We used the function of ApCreateWidget to create widget by copying a widget from the PhAB widget database

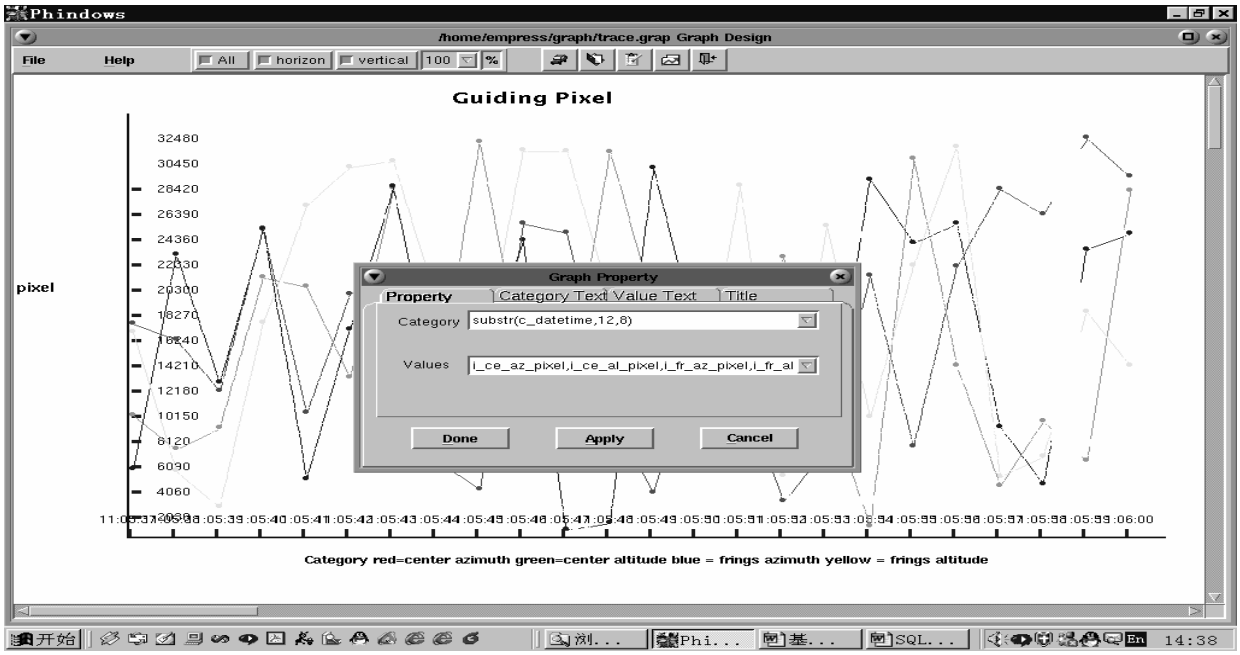

Figure 3 Guiding error is being generated by user's input preferences

Figure 3 shows the created report graph incorporating some dynamical widgets. The broken lines on the graph demonstrate the guiding error in units of CCD pixels. The parameters that determine how the graph looks could be dynamically created by user's input preferences.

6.3.2 Dynamical control widget management

Having been created, the dynamical widget will require some system resource. On the other hand, when the template, who has called the dynamical widget, is closed the program needs to destroy the dynamical widget to free the resource.

We use the chain list to manage the dynamical control in our program. Chain list is a structure which can be distributed dynamically without requiring continuous storage locations. That means the data can be stored in discrete locations. In addition for inserting and deleting data in the chain list it is not necessary to move almost a half of all the data around as in the case of the vector storage. In our programming the chain list was used in large scale. The structure below functions to store one column for a report form.

```
struct ST_DESIGN{
    PtWidget_t widget ;/*the handle of widget*/
   Char item_name[NAMESIZE];/*the attrib of the table*/
        char text[NAMESIZE];/*the name of the attrib*/
        PhArea_t area;/*the size of the widget*/
     PgColor_t font_color;/*the color of the widget*/
    PgColor_t fill_color;/*the fill color of the widget*/
          char font[NAMESIZE];/*the font of the widget*/
     struct ST_DESIGN next;/*the point of the list*/
};
```

6.3.3 Storage of data structure

Having been designed, the user report form or graph needs to be saved. We use the stream file to save the form or graph. The advantage of such a strategy is that it can save the data structure directly without the user's conversion. We illustrate the main functions involved in the programming below.

The fopen() function opens the file whose name is the string pointed to by filename, and associates a stream with it.

The fread() function reads nelem elements of elsize bytes each from the file specified by fp into the buffer specified by buf.

The fwrite() function writes nelem elements of elsize bytes each to the file specified by fp.

The fclose() function closes the file fp.

7. DISCUSSION ON THE DATABASE REAL TIME PERFORMANCE

Modern large astronomical telescopes are often required to provide a means of online observational data quality analysis, and LAMOST is no exception. Besides large number of control related data are involved in LAMOST control system's routine operation. Data sending, receiving, analyzing and processing go on all the time. These operations need to be conducted real timely within a fraction of each servo cycle. The TCS of LAMOST incorporates a number of such cycles to fulfill its observation mission, for example the mount servo cycle, field rotation servo cycle, guiding servo cycle and active force correction cycle of the Schmidt mirror plate, etc. In particular the tracking process is conducted rigorously based on the target passage that is extremely time-critical. We foresee that during the commissioning stage of the telescope it will absolutely be necessary to diagnose the mount tracking behavior against the readings from both azimuth and altitude encoders. All these call for a real time database to cope with.

To some extend, the real time performance of the database depends on the OS platform. In addition to the CPU's cycle frequency the interrupt latency, the scheduling priorities all contribute to the real time performance. The real time OS differs from non-real time OS in dealing with time crucial tasks. With real time OS the user is able to opt for the priority level for particular process so as to guarantee the spent time within the requirement. QNX OS platform serves this purpose well. Its multitasking, priority-driven preemptive scheduling, and fast context switching as well as built-in fleet protocol are all real time associated essentials. For processor of 166 MHz Pentium under QNX OS the interrupt latency could reach 3.3 microseconds, let alone for future much more fast processors.

The package we developed inherited the real time essentiality plus some other measures stated below, thereby the package fits real time requirement well.

The architecture of database structure plays an important role in building a real time database. Thoughtful storage arrangement of all these control related date is one of priorities. Bearing this in mind we put the tables most frequently accessed along with its process on the same node so as to reduce the data traffic greatly. In terms of data access efficiency it is naturally much better for local data than for remote data. In case of local date the access efficiency is basically limited by the CPU's frequency. For example, for LAMOST mount servo suppose to get the readings of azimuth encoder and altitude encoder every 20ms so as to produce a record. Plus some other related information the length of the record might reach 100 bytes. For examining our package to see if it could fulfill such a task within the time tick we conducted the test to insert a series of records of 266 bytes length each, repeatedly for 1000, 3000 and 5000 times respectively, and record the whole time interval and get the average time spent for each record. The following two tables show the test results in units of microseconds. The test was done on a Pentium 500 MHz with 512M memory and a Pentium 233 MHz with 192M memory respectively.

Pentium 500 MHz with 512M memory	Average time spent on inserting each record of 266 bytes in units of microseconds									
Insert a series of records of 266 bytes length each for 1000 times	12139	12129	12129	12149	12139	12129	12109	12129	12119	12119
Insert a series of records of 266 bytes length each for 3000 times	12125	12115	12122	12115	12122	12115	12119	12115	12125	12112
Insert a series of records of 266 bytes length each for 5000 times	12117	12119	12127	12119	12119	12119	12121	12127	12119	12125

Pentium 233 MHz with 192M memory	Average time spent on inserting each record of 266 bytes in units of microseconds									
Insert a series of records of 266 bytes length each for 1000 times	18158	18178	18148	18158	18258	18118	18138	18128	18128	18108
Insert a series of records of 266 bytes length each for 3000 times	18122	18178	18122	18108	18098	18118	18122	18125	18118	18118
Insert a series of records of 266 bytes length each for 5000 times	18152	18152	18150	18146	18152	18150	18156	18160	18140	18216

From the above two tables it is noted that the CPU's cycle frequency has an impact on the inserting time. Yet even with the slower PC of Pentium 233 MHz with memory of 192M the average time around 18ms spent on inserting each record of 266 bytes still meets the requirement, let alone for future PC with higher cycle frequency.

8. THE CONCLUSION

The research work passed a critical review among the experts from Beijing, Nanjing and Hefei at the evaluation meeting on July 2, 2002 organized by the LAMOST Engineering Headquarters. A number of advanced techniques have been adopted in the design such as dynamic creation of control widgets, dynamic query and share memory. What is more, the package provides the user with a friendly and powerful function of dynamic creation of database tables. Last but not least the real time performance of the RTD proved satisfactory for LAMOST specification.

REFERENCES

1. Xinqi Xu, "Control system and technical requirements - preliminary design", LAMOST Technical Report, 1998.
2. Xinqi Xu, "Preliminary Exploration of QNX Real Time OS Based Application to Large Astronomical Telescopes", *Astronomical Instruments and Technology*, 1999.
3. Xinqi Xu, "The Control System of LAMOST Telescope", *ACTA ASTROPHYSICA SINICA*, **Vol. 20 Supplement**, , P.43-52, 2000.
4. Lingzhe Xu, Xinqi Xu, "Outlook of Distributed Database Application to LAMOST Control System", LAMOST Internal Technical Report, 2001.
5. Xinqi Xu, Jun Zhou, Lingzhe Xu, "Software simulation of the LAMOST control system", **SPIE Volume 4757**, P.145-153, 2002.
6. Xinqi Xu, Lingzhe Xu, Gangping Jin, "Overview of LAMOST control system", **SPIE Volume 4837**, 2002.