Enabling schools and public access to the Liverpool Robotic Telescope

I.A.Steele, A.M. Newsam, C.J. Mottram, P.McNerney

Astrophysics Research Institute, Liverpool John Moores University, CH41 1LD, U.K.

ABSTRACT

Five per-cent of the observing time on the Liverpool Telescope (a 2-m robotic telescope sited in La Palma) will be set aside for public understanding of science. Schools access will be via. a queue scheduling mechanism, and public access via live Planetarium shows. We describe the development and performance of a generic Java message passing system to allow communication between the processes implementing the robotic control of the telescope and the remote processes that will be run at the Planetarium. We also describe an adaptive data compression algorithm to allow transfer of data back from the telescope in near real time and our software for Planetarium access which allows staff at the Planetarium to implement their own control system and display software. Finally we describe our hierarchical web-based system for schools to input observation requests and the image processing software we have developed to allow them to make quantitative measurements of the resulting data.

Keywords: robotic telescopes, Internet, education, public access

1. INTRODUCTION

The Liverpool Telescope (LT) will be a 2.0-m fully robotic telescope situated on the island of La Palma in the Spanish Canary Islands. By "fully robotic" we mean that the telescope is capable of autonomous operation, unsupervised during the night either at the telescope site or in the UK. Such a mode of operation allows a number of unique science programmes to be implemented, such as long term monitoring and fast response to targets of opportunity that are organizationally virtually impossible with a conventionally scheduled and operated telescope.[1]

Five per-cent of the time on the LT is reserved for Public Understanding of Science and Technology (PUST). This is a mixture of schools and general public access with the overall general objectives of supporting the teaching of science in schools and raising public awareness of and support for astronomy, and by extension other basic scientific research. We describe the objectives of each programme in more detail in Section 2. The methods of operation for each audience we have decided on are different (queue scheduled access for schools, live planetarium shows for the public), however they both must interface with the robotic control systems of the telescope, and both have a requirement to transport the resulting data back to the UK in a timely manner. In Section 3 we therefore we give a brief overview of the software architecture we have put in place to allow robotic operation of the Liverpool Telescope, and in Section 4 discuss the Java message passing system we have developed to allow easy and reliable communication between the processes that make up the robotic systems. In Section 5 we describe our solution to the limited Internet bandwidth between the UK and La Palma and in Section 6 we describe the software we have developed to support schools in both specifying observations, and making measurements on those observations once received. In Section 7 we describe the software we have developed to allow Planetarium staff to control the telescope and retrieve data. Finally in Section 8 we give our conclusions and make recommendations for future similar projects based on our experience.

2. OBJECTIVES FOR PUBLIC AND SCHOOL ACCESS

2.1. Public Access

Public access to the Liverpool Telescope will be via. the Planetarium and Space Gallery of Liverpool Museum. This is one of the principal museums within the National Museums and Galleries on Merseyside umbrella and houses strong collections in Archaeology, Natural History and Physical Sciences. It also houses a 66-seat planetarium which attracts 44,000 visitors per year, of whom 30,000 are members of the public and 14,000 are in pre-booked school

E-mail: ias@astro.livjm.ac.uk

groups. Over 1200 shows are given every year, making it one of the busiest for its size in the world. The Zeiss ZKP1 star projector is supplemented by the use of video and slide projection equipment.

Our objectives for public access to the Liverpool Telescope via. the Museum are as follows:

1. To bring an element of excitement and achievement to public audiences in taking an active part in high quality astronomical observations.
2. To bring together the research, teaching and public faces of astronomy locally and regionally, and to provide avenues for the public to develop further interest and expertise.
3. To introduce members of the public to aspects of active astronomical research such as observing techniques, experiment design and image processing
4. To maximize public awareness of the LT.
5. To increase the number of visitors to Liverpool Museum and Planetarium, including groups that would not normally visit the facility.

2.2. Schools Access

Unlike public access, where the emphasis is mainly on generating and holding public interest, for schools use our objectives are more focussed towards scientific experiment:

1. To support the teaching astronomy content in the UK National Curriculum (ages 11-16) and A Level / Scottish Higher physics (ages 17-18).
2. To support the teaching of "scientific investigations" in schools (the process of planning, prediction, experiment, analysis, comparison with theory and refinement of theory).
3. To provide an inspirational real-world resource for teaching in other subjects (e.g. Mathematics, Languages, Geography)
4. To provide a real-world example of the application of control and communication technology.
5. To show children the excitement of scientific observation and discovery and promote science as a career option.
6. To provide this resource to as many schools and children as practical across Merseyside and the UK.

3. SOFTWARE ARCHITECTURE

The basic software architecture for robotic control of the Liverpool Telescope is shown in Figure 1. The usual method for professional astronomers to interact with the Liverpool Telescope will be via. the "phase II database". This interaction is via an observing tool which allows observers to completely specify the parameters (e.g. coordinates, instrument configurations, exposure times etc.) of an observation. An object oriented model of this database was presented in Ref. 2. The observing tool is run as a local client on the observer's computer, and communicates with the phase II database via. the Internet. The phase II database is implemented as a server with priority queue access. As well as remote observers, the other client of the database server is the Robotic Control System (RCS). This is responsible for a number of tasks, including:

- start up and shut down procedures
- calibration procedures
- requesting the next group of observations from the phase II database based on a scheduling algorithm[3]
- logging
- high level error recovery procedures

To carry out these tasks the RCS must interface with the Telescope Control System (TCS), which has responsibility for telescope pointing and tracking, and the Instrument Support System (ISS), which provides a generic interface to astronomical instrumentation (both imaging and spectroscopic) that may be mounted on the telescope.

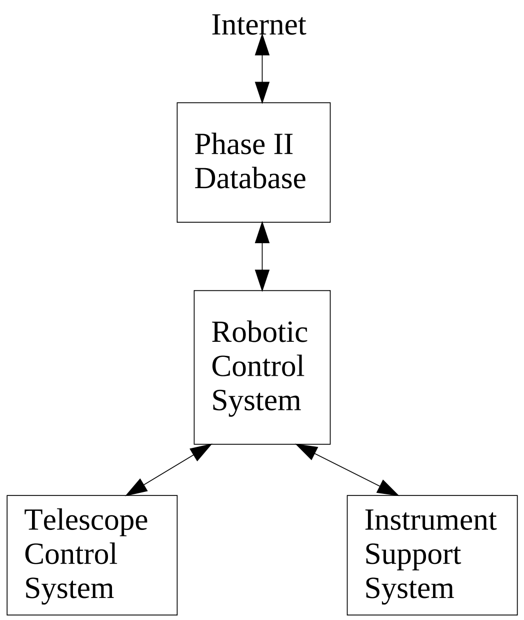

Figure 1. Software Architecture at Telescope Site

4. JAVA MESSAGE SYSTEM

4.1. Design Options

All of the systems described in the previous section are implemented as Java applications, apart from the TCS, for which we have developed a Java layer on top of the underlying FORTRAN code. There is a need to pass control and status information between these processes, which can be executing on different machines both at the telescope and remotely. We considered several different mechanisms to pass this information at the design stage:

- **ASCII text over TCP/IP sockets**. This is the simplest interface. It could be written in either C or Java and would probably be the most efficient communication methods with the minimum of overhead. However there would be a significant overhead in abstracting the information out of objects into memory to send across the network. Also, when the contents of the objects change then the code to abstract the information out of the objects would also have to change: two pieces of code (at least) would have to change when the contents of a message changes. It would be easy for the two pieces of software not to be up to date leading to errors.
- **Remote Method Invocation(RMI)** RMI is the Java version of rpc's (remote procedure calls), where you effectively call a remote method on another machine. The problem with this method of communication is that it limits the information that can be returned from the remote method, and it does not allow a two way communication dialogue to be set up easily.
- **Object Serialization over TCP/IP** This is another Java mechanism for sending Java objects over any stream interface (a socket connection being a stream interface). Transferring an object simply involves one line of code as does receiving an object. Serialization creates a tree of all objects referenced by the root object, simplifying the data transmission process. However, care has to be taken that the classes at both ends of the connection are the same version otherwise serialization can fail (this is probably a good thing).

We have selected Object Serialization as our mechanism for object transfer. The simple socket connection will involve development of a lot of support code, which will be difficult to keep up to date. RMI is fine for simple communication, but will soon become too complicated when two processes need interactive two way communication.

4.2. Protocol

The message passing system is a simple protocol, consisting of three types of messages sent over the communication media:

- A *COMMAND* message, the class of which determines the action to be taken by the receiving server, and which contains fields holding the necessary parameters for the action.
- An *ACK* (acknowledgment) message, which informs the client the command has been received and a time estimate for completion.
- A *DONE* completion message, which informs the client the command has been completed, and contains fields with information on the success (or not) of the action and the results produced.

Because all these types of messages are themselves classes, they can be sub-classed. This allows us to organize the different messages that need to be sent hierarchically, so that messages with similar parameters can inherit the same fields from common super-classes. One of the main benefits of object-orientation is used to organize and simplify our message passing.

Figure 2 shows how a transaction takes place within the system:

As can be seen from the figure, the basic transaction between the client and server consists of:

- The client sends the server a command message.
- The server sends the client one or more acknowledgment messages.
- Upon command completion, the server sends the client a done message.

The purpose of the ACK messages is to allow the client to implement a timeout check on the connection (i.e. while waiting for a DONE message) that has a duration that is specific to a command and its associated parameters (e.g. a slew between targets in opposite parts of the sky should have a longer timeout than a slight "tweak" to the telescope pointing). A timeout is also possible when waiting for the ACK message as we expect this response to be virtually instantaneous as it is simply a calculation of expected time to complete. A timeout value of 10 seconds is always (more than) sufficient for such a calculation.

We found it was useful to allow the server to send more than one ACKnowledgment back to the client. This simplifies the case when the server has to perform several sub-commands to complete a client command, as the server can issue an acknowledgment for each sub-command.

4.3. Implementation

We have implemented our Java Message System in a package (ngat.net) with three basic classes that may be subclassed to create specific messaging systems between two processes:

- TCPServer. This is the server class. It runs in it's own thread. It creates a SocketServer to listen for connections, and when it accepts connections it spawns a TCPServerConnectionThread to deal with it. Real servers would sub-class this class to spawn a sub-class of TCPServerConnectionThread (which provides only the base functionality).
- TCPServerConnectionThread. This class deals with one message transaction from the server end. It reads the command in, sends an acknowledgement, processes the command (an empty implementation in the base class) and then sends a done message to terminate the transaction. This thread would be sub-classed to provide an implementation of the commands. The thread then terminates.
- TCPClientConnectionThread. This class provides the client end of the protocol. It is again a thread, so that once a command has been sent to the server the client can continue it's own operations. For instance, a client can send an ABORT message to the server to terminate a command that seems to be taking the server too long. The thread sends the command, get the acknowledgement and then gets the done message from the server. The thread then terminates.

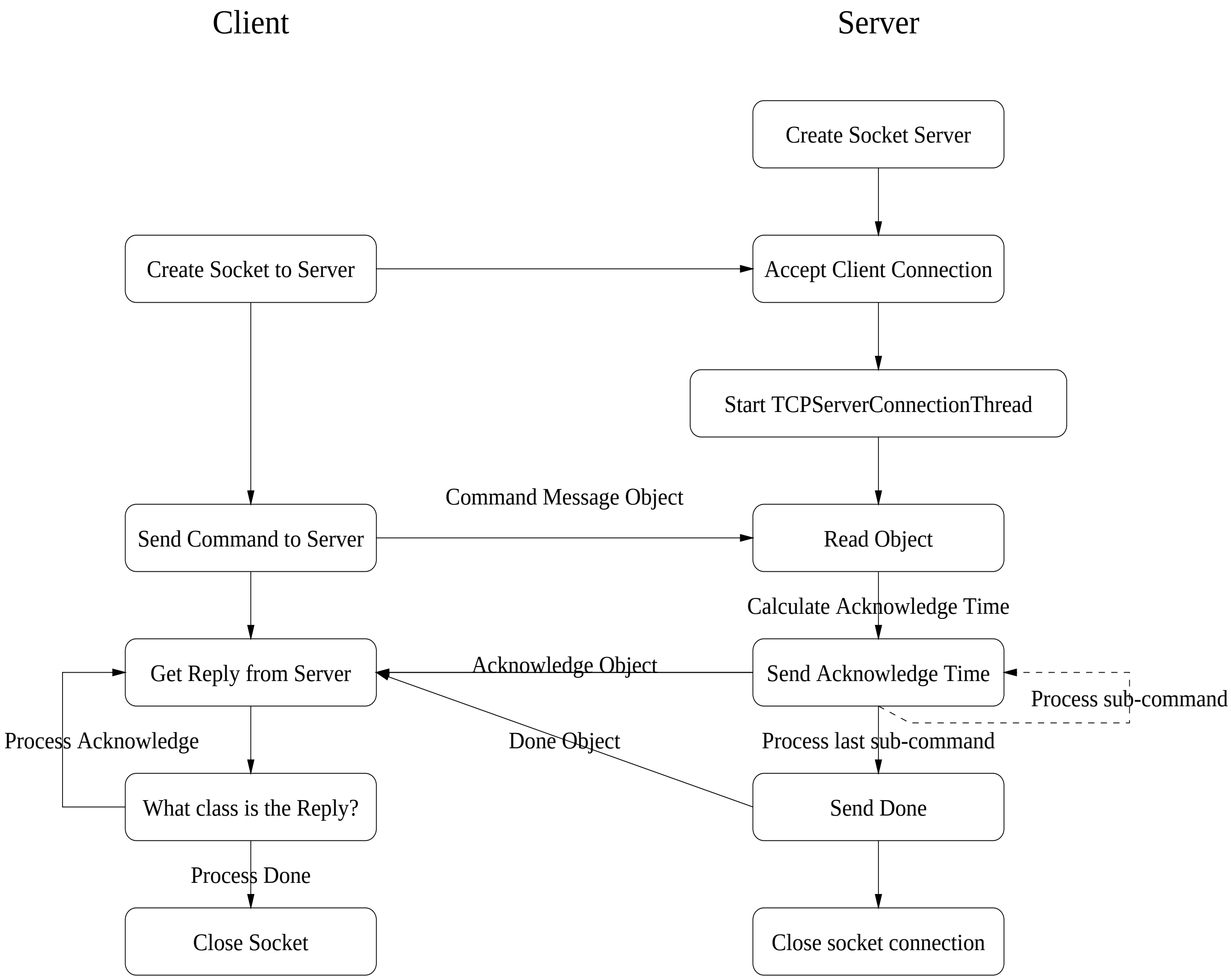

Figure 2. ngat.net example multiple acknowledgment transaction.

A mechanism to improve performance is buffering of the input and output streams that the messages are sent over. Java already provides a mechanism for implementing buffered input and output streams. This takes the form of two classes, BufferedInputStream and BufferedOutputStream. These wrap round the input/output stream obtained from the socket connection, and buffer input/output from the ObjectInput/OutputStream. It was relatively simple to add this functionality to the TCPServerConnectionThread and TCPClientConnectionThread thread classes although our initial attempts at buffering the streams caused a problem to occur. Both Client and Server threads hung when the ObjectInputStream was created for a message transaction. This was because the ObjectOutputStream needs to send the ObjectInputStream header information as the ObjectInputStream is constructed. However, the header was small enough to fit in the buffer and so was not sent to the socket layer, and the ObjectInputStream was left waiting for the header. A *flush* method call after the ObjectOutputStream constructor solved this problem.

In order to test the effects of this enhancement and the performance of JMS we implemented test clients and servers. Two machines were used for the test: ltobs5 and ltccd1. ltobs5 is a Linux PC, and ltccd1 is a Sun Solaris machine. This allowed a comparison of the relative performance of the two sets of hardware All the tests were performed using JDK 1.1. Each test was performed by timing the time taken for 100 message transactions (command/acknowledge/done objects) to be sent over the network and repeated at least twice. Three messages were used:

- CONFIG - This was the biggest message we have so far devised to be sent across the network, at 3827 bytes. * It contains an instrument configuration complete with a proposal object, although the current CONFIG object no longer contains this(causing the size to drop to ≈672 bytes).
- GET_STATUS - A much smaller message at 257 bytes, used to request the instrument's current status. The GET_STATUS_DONE message contains a Hashtable with a single element.
- SET_LOGGING - Another small message at 266 bytes, this one with no DONE parameters.

The results of the test are shown in Tables 1 and 2

Message Object	Time to complete Test (ms)	Messages Per Second(m.p.s)
CONFIG	18250	5
GET_STATUS	5268	18
SET_LOGGING	5172	19

Table 1. JMS Test results (no buffers)

Message Object	Time to complete Test (ms)	Messages Per Second(m.p.s)
CONFIG	12262	8
GET_STATUS	3213	31
SET_LOGGING	2827	35

Table 2. JMS test results (2048 byte server/client buffer, No Thread Pools.)

In this case using a buffer of size < 2 Kbytes caused a doubling of message throughput, and this improvement was incorporated into the final design. Interestingly buffer sizes greater than 2 Kbytes caused a performance degradation, highlighting the need for comprehensive testing when implementing such systems.

More generally, these results are fast enough to enable us to use Object Serialization over TCP/IP sockets successfully. We need a performance of at least 2-3 m.p.s. for sending messages between these processes. As these processes have overall control over the system they do not need sub-second response time. Neither hardware system seemed to offer a performance advantage over the other. The system seems to scale quite well with regard to message size. The m.p.s. of the largest message under test is a little slow, but this is the largest object to be sent over the network to date.

We note here that both the TCPServerConnectionThread and TCPClientConnectionThread threads only run for one message transaction. After this they stop, and another thread of this class has to be created for the next message to be sent. At first sight this appears inefficient, and a revised implementation was done using a Thread Pool to which connections could be attached in order to remove the overhead of thread creation. However it was found (Table 3) that this (more complex) implementation actually reduced performance and was so dropped.

*Note these message sizes are only approximate, based on the size of the serialized object on disk. Presumably each file contains the ObjectOutputStream header bytes in as well, which will only get sent once over the network (i.e. The size includes header bytes which will not be sent every time the message is sent over the network). To determine accurately how much data was sent over the socket some other mechanism would need to be used (which would probably have a heavy impact on performance).

Message Object	Time to complete Test (ms)	Messages Per Second(m.p.s)
CONFIG	14231	7
GET_STATUS	8772	11
SET_LOGGING	9464	12

Table 3. JMS test results (2048 byte server/client buffer, Server and Client Thread Pools.)

5. DATA TRANSFER BACK TO THE UK

By long term (> 2 years) monitoring of the Internet link between the UK and La Palma,[4] we have found that we can expect a minimum transfer rate of 25 KBytes/sec in the early evening. The telescope is currently equipped with a 2048x2048 pixel 16-bit CCD. A typical image is therefore 8MB. For Planetarium operation we have set a target transfer time of 10 seconds. This implies a compression ratio of 30:1 is required.

As we wish the transferred data to be scientifically useful to schools and the planetarium, it is necessary that our transfer is *quantitatively* accurate. However general image compression schemes (e.g. JPEG, GIF etc.) are designed to produce images that are *qualitatively* accurate ("look good"). We have therefore decided to develop a dedicated compression scheme for astronomical images which exploits their structure. In this section we will give a very brief overview of the compression scheme we use. Further details will be presented in Ref. 5.

We find we can classify astronomical images into two basic types ("star-fields" and "low-contrast/extended"). The first stage of our compression is therefore to determine the image type. This is done by an analysis of the image histogram. After the image type is determined, a compression scheme especially devised for the image type is applied.

"Star fields" contain large areas of background which do not contain useful information. Superimposed on this background are the objects of interest. For such images it is the retention of photometric and astrometric accuracy for these objects that is important, and our algorithm is designed to take advantage of this by transmitting the information about the objects in a lossless fashion while encoding the background in a lossy manner.

"Low-contrast/extended" images contain a wealth of low contrast features. An example of such images are those of planetary surfaces. For such images there may not be large areas of background. In this case, it is the retention of morphological detail rather than absolute photometric accuracy that is generally critical. For such images we therefore use a different algorithm, which is based on wavelet transforms.

Overall we find using prototypes of the compression schemes outlined here we can achieve compression ratios greater than the 30:1 required. We are therefore confident that we can achieve the necessary real-time performance for Planetarium operation.

6. SCHOOLS OPERATION

Schools will operate the Liverpool Telescope in the same manner as professional astronomers, that is by having details of their observations stored in the phase II database for queue scheduling at a later date. In this mode of operation there will typically be a delay of up to one week between the pupil requesting data and receiving it. We do not see this as a problem, but rather an advantage, as it re-enforces the concept of observing as a scientific experiment, where one must carefully plan what to do, wait for results and then evaluate them. To support teachers in using the Liverpool Telescope we have produced a large amount of curriculum material, including suggested experiments, worksheets and guides to the telescope operation and image processing. This has been developed in a series of working parties consisting of a mixture of staff from the Astrophysics Research Institute and local teachers. The involvement of teachers in this process has been crucial in specifying, ensuring the applicability of, and testing the materials produced, and in encouraging teachers that using real data from the telescope is a worthwhile educational exercise.

While the schools observation requests will be stored in the same database as those of professional astronomers, the phase II observing tool has been developed for professional use. It is therefore not appropriate in terms of user interface and functionality for schools to input details of their observations. In addition the requirement of downloading and installing the a Java application plus a Java Virtual Machine (JVM) on the observers computer is not convenient for schools use.

In order to address these problems we have devised an alternative web based mechanism for entering details into the phase II database. This uses a hierarchical set of web forms to collect details of the required observations. The forms are served from the Liverpool Telescope web server in the UK (http://telescope.livjm.ac.uk/) and are driven by a Java servlet. The servlet converts the form data to the phase II database objects for transmission to the phase II database at the telescope via. our message passing system described in Section 4.

Much effort has gone into the design of the web form system for schools use. The majority of school teachers have little or no astronomy training. Therefore concepts such as Right Ascension and Declination have little meaning to them. We have therefore derived a hierarchy for requesting observations based on the type of experiment the school wishes to perform and the nature of the objects required for that experiment. Figure 3 shows the preliminary hierarchy we have devised. By navigating this hierarchy via. a series of web pages, the school user will be able to arrive at the details of the source to be observed. A similar hierarchy of questions (e.g. do you wish to construct a multi-colour image?) will then lead the user to define the instrument configuration and exposure time necessary to perform the observation.

Once observations have been carried out, the resulting data will be transfered back to the UK web site using the compression scheme outlined in Section 5. The compressed data can then be downloaded from the UK web site to the school via the Internet. Decompression of the data will occur at the school as part of the image analysis package LTImage that we provide to the schools on CD ROM. This has been design to allow useful *scientific* data to be extracted from the supplied images. We note here that the standard professional astronomical packages (e.g. IRAF) are not suitable for this task as they are (a) too complex, (b) generally require a Unix-like operating system, and (c) often require relatively powerful hardware compared to that available in the typical UK school classroom. Amateur packages tend to be more usable, but they often manipulate the images in "non-standard" ways in order to achieve visually impressive, rather than quantitatively accurate results. Finally general image processing packages (e.g. Photoshop, Paintshop Pro) tend to lack the facilities to make quantitative measurements of image features.

LTImage (see Figures 4 & 5) has been designed to run on the typical machines found in UK secondary schools. It therefore does not require high specification hardware, and in fact runs with reasonable performance on 486 class machines with 4MB RAM and a 1MB video card. Similarly it runs under all varieties of Microsoft Windows from 3.1, through 95 and 98 to NT. In addition a Linux version is available. It provides a large degree of functionality, including:

- Multi Level User Interface (teacher can select Basic, Advanced or Expert Mode)
- Colour table manipulation
- Variable Zoom on main and side windows
- Image statics, including photometry of individual objects
- Ability to measure distances and angles between image features
- Ability to load up to 4 images into internal buffers for blink comparison
- Ability to combine 3 images into a true-colour image

7. PLANETARIUM OPERATING SYSTEM

Unlike Schools Access, access via. the Planetarium will be real-time. In other words, during a show given by a demonstrator at the Planetarium, it will be possible to request live images from the telescope and have them projected in real time on the Planetarium Dome. To achieve this we do not use the phase II database at the telescope to hold details of the requested observations. However simply allowing direct access to the telescope and instrument control user interfaces is also not a viable option, as these are unnecessarily complex for a non-professional user and do not provide any automation of common procedures. Such automation is provided by the Robotic Control System, and our chosen solution takes advantage of this. Our solution was also driven by a requirement to allow

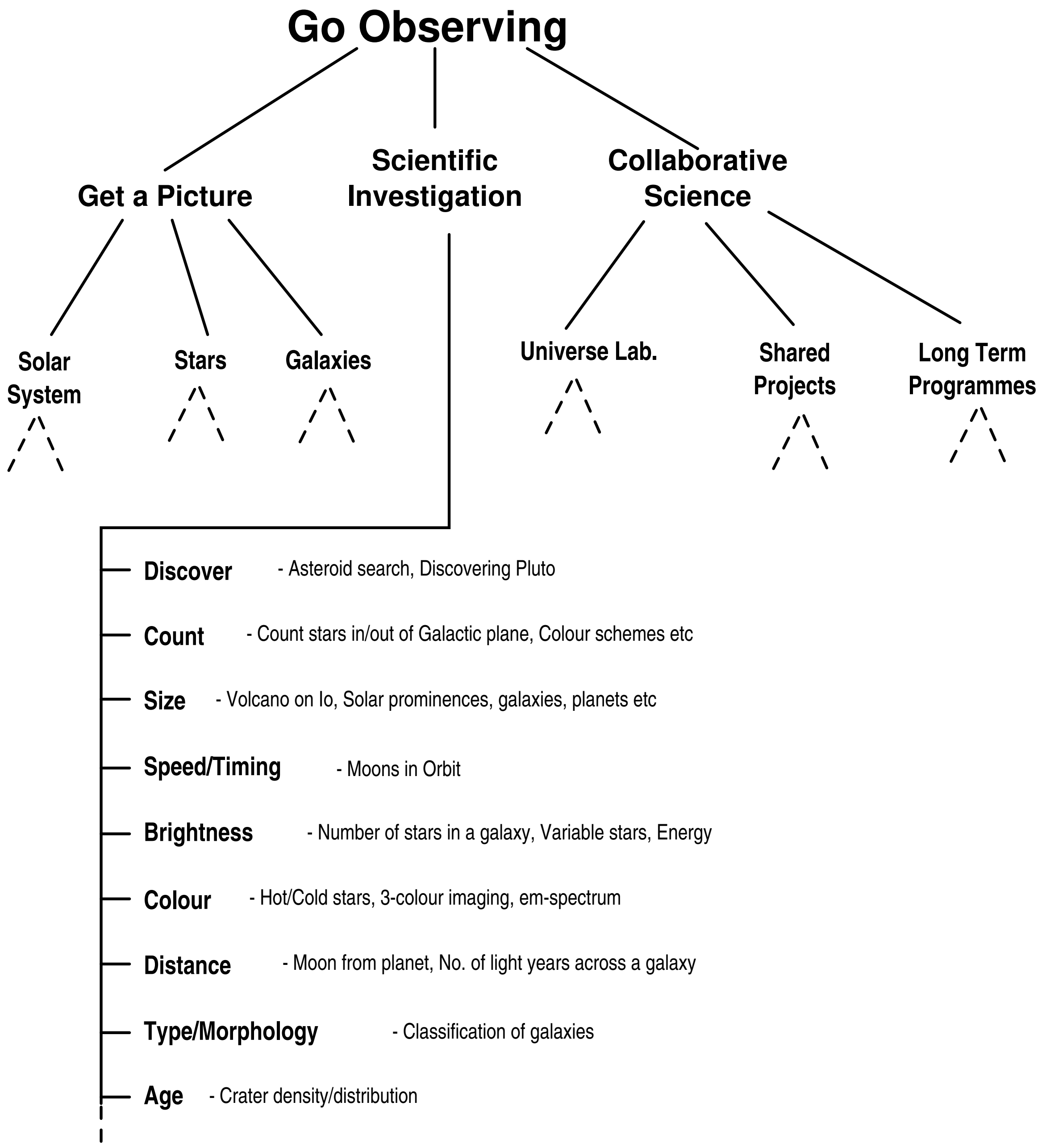

Figure 3. Hierarchy of experiments on schools web site

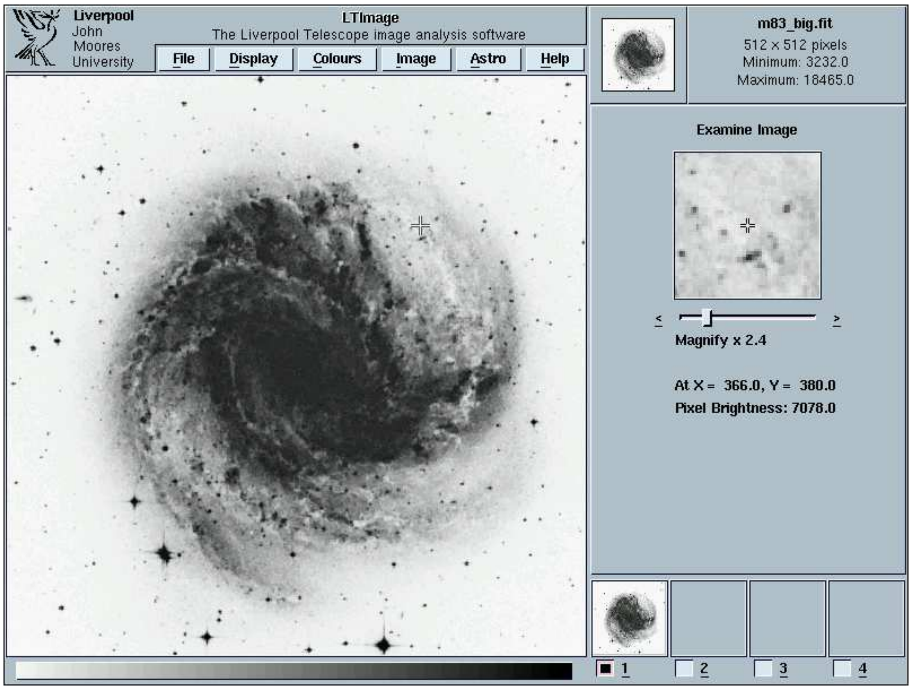

Figure 4. LTImage example screenshot I

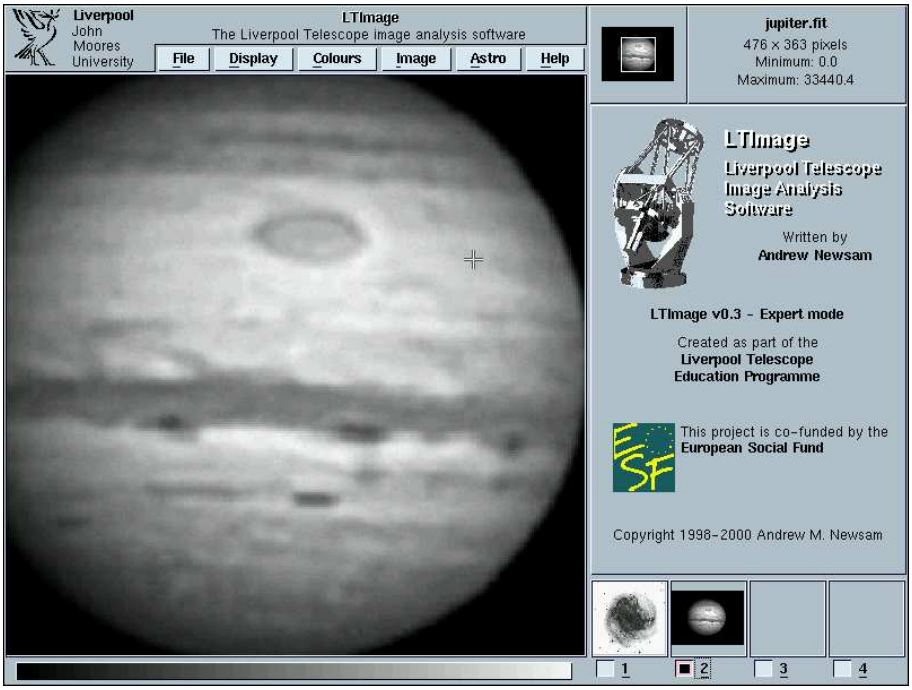

Figure 5. LTImage example screenshot II

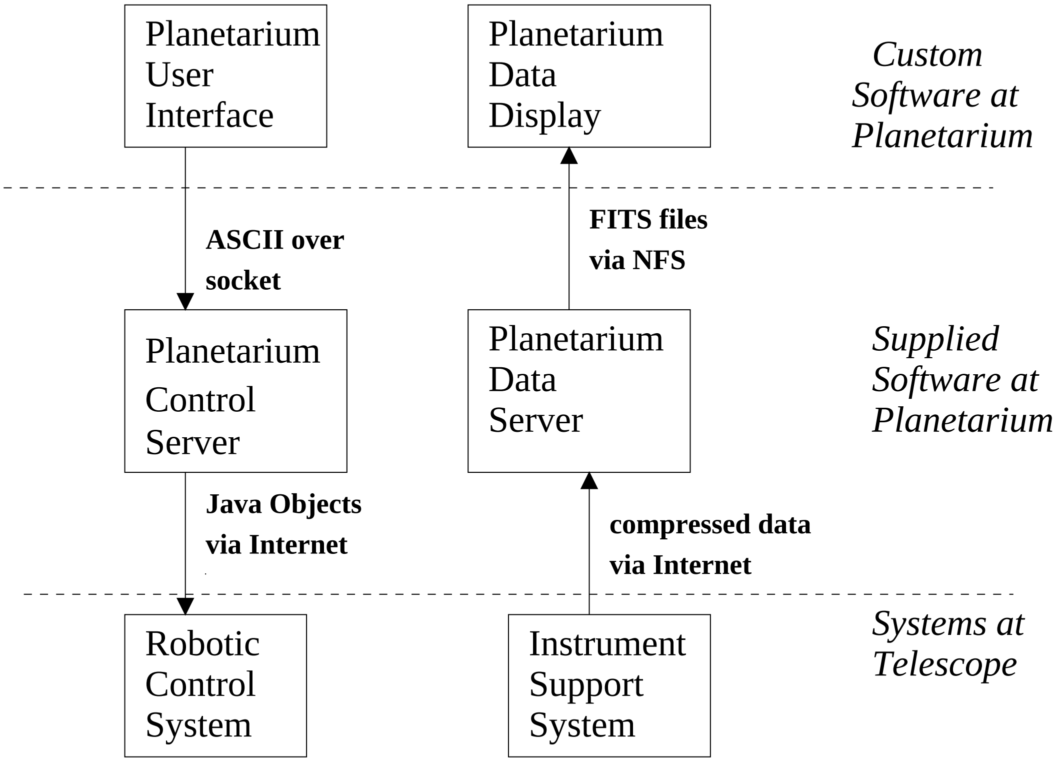

Figure 6. Software Architecture for Planetarium Mode Operation

Planetarium staff to develop their own control and display applications, as the operating environment (based around Windows and Visual BASIC) of the current Planetarium software was one in which JMU staff had little experience.

In essence our solution simply comprises instructing the RCS to request details of the next observation not from the phase II database at the telescope site, but from the Planetarium (Figure 6). We have implemented a Planetarium Control Server (PCS) that mimics the interface of the phase II database. This runs on a PC located at the Planetarium and holds its own internal queue of observations requested by the Planetarium User Interface (PUI). In order to allow Planetarium staff to design an appropriate (Visual BASIC) interface for their demonstrators we do not provide the PUI software to the Planetarium, but rather have implemented a simple, ASCII string based, socket interface to the local queue held by the Planetarium Control Server. This interface has only 3 basic commands (GOTO a position, SELECT a filter, EXPOSE for a time). These are packaged into the appropriate phase II-style Java objects by the PCS which holds them ready to be requested by the RCS.

Once data is acquired, it is transferred back to the Planetarium Data Server (PDS) over the Internet using the compression algorithm outlined in Section 5. The Planetarium Data Server decompresses the data and makes the resulting images available as a simple FITS file via. NFS to the Planetarium Data Display. Again the implementation of the hardware (data projector etc.) and software of this system is left to the Planetarium technical staff.

8. CONCLUSION

In this paper we have described some of the mechanisms we have put in place to allow schools and public access to the Liverpool Telescope on La Palma. In many ways this is a more complex problem than that of allowing professional access to a robotic telescope. We have evolved different mechanisms to support this access. For access by the general public we will provide live “shows” where the telescope is controlled in real time from the Planetarium of Liverpool Museum in the UK. For access by schools we will use a queue scheduled mode, with a delay of up to one week between requesting and receiving data.

The implementation of both of these modes has been described. In order to allow communication between different software processes both at the telescope and in the UK we have derived a Java message passing system that aims to be robust yet flexible and simple to adapt to new message types. In order to allow data transfer back to the UK we have devised an adaptive compression algorithm that evaluates the image type to apply the most appropriate compression scheme. The compression ratio achieved ($> 30 : 1$) is sufficient that transfer of 2048 pixel square CCD images may be transferred back to the UK via the minimum expected Internet bandwidth to La Palma in 10 seconds. We have devised a software architecture for Planetarium access that allows the staff of the Planetarium to implement their own user Interface and data projection system simply. For schools access we have developed a hierarchical web based system (built on Java servlet technology) to guide object selection and instrument configuration based on a series of questions. We have also produced image display software for schools (LTImage) that incorporates facilities for students to carry out simple quantitative analysis of data from the telescope, and supported this with accompanying curriculum material.

We have a number of recommendations for groups considering similar projects in the future. It is essential to work closely with the end users of the system to ensure that what you deliver meets their objectives. It is also important to realize that real-time access is not always a necessity (as we have done with our schools access programme, where through discussions with teachers it became apparent that the more structured approach that queue scheduling offered actually fitted their educational objectives better than real-time control). Where real-time access is necessary, our tests reported elsewhere in these proceedings[4] have shown that (even in Europe!) sufficient connectivity can usually (9 out of 10 days) be relied upon to transfer images back within 10 seconds if a compression ratio of 30:1 can be achieved. Our experience shows that bespoke algorithms for such data can be derived. Finally our experience of object oriented analysis and design techniques has been very positive in allowing the design of a message passing system that is common across the various telescope systems.

ACKNOWLEDGMENTS

The development of the Liverpool Telescope and its associated educational programmes is part-funded by the European Union and the UK Particle Physics and Astronomy Research Council (PPARC).

REFERENCES

1. I.A.Steele, "The Liverpool Telescope", in *The Future of the Isaac Newton Group*, N. Walton ed., New Astronomy Reviews, in press, 2000.
2. I.A.Steele, "An object model of the Liverpool Telescope", in *Telescope Control Systems III*, Proc SPIE, **3551**, H. Lewis ed., 343, 1998.
3. I.A.Steele, D. Carter, "Control Software and Scheduling of the Liverpool Robotic Telescope" in *Telescope Control Systems II*, Proc. SPIE, **3112**, H. Lewis ed., 222, 1997.
4. P. McNerney, I.A.Steele, "Internet Quality of Service between the UK and La Palma 1998 - 1999", this volume, 2000
5. P. McNerney, PhD Thesis, Liverpool John Moores University, 2000