Chapter 3

A Distributed Framework for Internet Telerobotics

Barney Dalton

Abstract

This paper describes a framework that can be used to control devices over the Internet. It is written using Java and is therefore portable to any platform that can run the Java 1.2 runtime environment. The framework provides a number of services that are necessary for Internet control, these include; connectivity, user authentication, addressing, resource allocation, time delay management, and ordered request delivery. These services are provided through the abstraction of shared contexts that include sessions, channels and tokens. The framework has been applied to the control of an industrial robot. Users download a Java applet that shows images of the robot workspace, commands can be generated to perform 'pick and place' tasks with objects in the robot workspace. The client interface uses an Augmented Reality cursor for command generation. As the client is downloaded as an applet, code size must be kept to a minimum, therefore the framework is designed to be as lightweight as possible. The project is based on the assumption that Java is a workable multiplatform language. Although in theory this is the case, there are unresolved issues of memory use, reliability, and consistency with some Java virtual machine implementations.

3.1 Introduction

This paper describes a framework that has evolved out of the UWA telerobot project first started by Ken Taylor[15,16,14]. The framework has been designed to minimise the work required to connect a robot or other device to the Internet, while providing the basic functionality needed for a collaborative distributed system.

Most of the early World Wide Web controlled Robots[14,5] used the Common Gateway Interface(CGI)[2] to interface between web browsers and the physical device being controlled. CGI processes are launched by a web browser in response to certain HTTP requests, and the CGI result is sent as the HTTP response. HTTP is a stateless request response protocol, meaning it is very simple to implement but can have shortcomings. As it is stateless, state must be managed by the browser and the CGI process. This is normally achieved by passing extra identification information with each request, known as cookies[7]. The request response paradigm means that after a client request has been processed by a server, there is no way for the server to contact the client. The client must always initiate contact. This can be a problem when the client is interested in the state of a non stationary remote process, to receive constant updates the client must poll the server at regular intervals. Polling is inefficient as requests must be made even when there are no changes and the server must handle these requests using resources to process them. Added to this, new information is only received with each poll, not as it becomes available.

The WWW has undergone major changes since the first online robots in 1994, and there are now new techniques available for robot control. The introduction of Java is probably the most significant as it allows code to be executed on the client side, whereas previously all code had to be in the CGI process. It is being used by a number of web telerobotics projects including the PUMA paint robot [12] and the NASA Pathfinder interface [1]. Using Java applets, a client can provide both a more sophisticated interface and use its own protocol to communicate with the server. The use of a different communication protocol means that the limitations of HTTP mentioned above can be overcome. With a constant connection between client and server, both sides can communicate immediately when new information needs to be transmitted. For telerobotics, this ability for server initiated conversation is important as changes in the state of the robot and workspace needs to be uploaded to the client with minimal delay.

With each client running a Java applet and communicating with a server, the system can be termed distributed. There are many issues associated with the design and implementation of a distributed system. The next section identifies the issues most relevant to distributed control of physical devices. Most of these issues are independent of the physical application, and can therefore be addressed by a framework that can then be customised for specific applications. Section 2.1 presents a framework that has been implemented in Java to address these issues. The following sections discuss its application to the control of an Industrial robot arm.

3.2 Design

An Internet controlled telerobotic system is likely to consist of a number of physical devices such robot(s), cameras, sensors, and other actuators. Software to control these devices may require startup time that should be minimised by using permanently running servers. It is also likely that some devices may require different hardware and software configurations. The architecture of such a system is therefore likely to consist of a number of server processes that may be distributed across different machines. Added to this, the clients themselves will be in many different locations. The servers and clients are all part of the system and may all need to communicate with each other. In this context both clients and servers are referred to as peers.

Writing servers can be complicated and time consuming. Not only does the software to perform the control and sensing need to be designed, but a communication technique and protocol must also be chosen. This should enable communication between some or all of the peers in the system. The number, type, and location of peers may change over time. As communication may come from a number of peers, they need to be identified and authenticated. If the device being controlled cannot be shared freely then its allocation to different peers needs to be managed. These and other requirements are similar for all servers and clients, and can therefore be abstracted and provided as a framework.

Some of the issues and requirements that a distributed robotics framework must address include:

- Connectivity
- Authentication
- Addressing
- Resource allocation
- Time delay management
- Request Delivery

Connectivity is the most basic requirement. For parts of a system to communicate with each other, they must be connected in some way. Potentially each peer may want to communicate with any number of other peers that are part of the system.

Authentication is essential to differentiate requests from different parts of the system. Both, initial authentication and ongoing authentication may be required. Authentication typically consists of a challenge and response, where the actual response is compared to a expected response. Once authenticated a user's details can be retrieved. This may include information such as personal preferences, authority level, group membership, and personal information. Here the term user may apply equally well to a software server or agent, or to a human user.

Addressing provides a mechanism by which one peer of the system can communicate directly with another peer by specifying the address of the recipient. The recipient my be a single peer, a list of peers, or all known peers.

Resource allocation is required for any resource that cannot be shared freely amongst all peers. This may be data files, device settings such as camera image size and quality, or physical devices themselves such as a robot.

Time delay management is necessary for any control application. Currently the largest portion of the delay is caused by data travelling over a network. This is predetermined by the network path connecting parts of the system. However, correct choice of protocols can help to minimise delay at the possible cost of reliability. Even though the delay cannot necessarily be controlled it can at least be measured, providing estimates of information age, and therefore allowing predictions of the current state to be made.

For some devices, the order in which requests occur is important. These are normally commands that have some temporal or spatial relationship to the state of the device, or its environment. For example, a relative command 'move left' is issued on the assumption that the device is in a particular place. Any prior movement will render this command incorrect.

Reliable delivery and speed often have to be traded off against one another. To ensure reliable delivery extra information must be passed backwards and forwards, at the cost of speed. For important messages, reliable delivery must be chosen over speed, but there may be messages that do not require reliability, and can therefore benefit from the increased speed gained from less handshaking.

3.2.1 Software Architecture and Design

The system architecture is based on Message Oriented Middleware (MOM) [9,11]. Peers connect to each other through a central router known as the

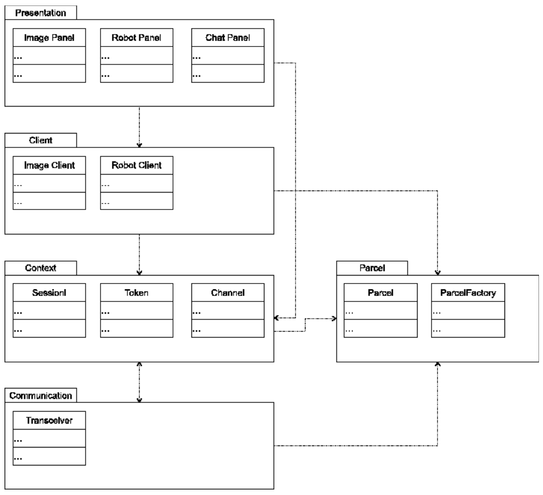


Figure 3.1: The Layers of the MOM architecture for a robot client, shown using UML package notation. The presentation layer provides the user interface. The middle layers are explained in more detail later in this section. The bottom transceiver layer sends and receives objects over the network. Communication between the different layers uses parcels.

MOM. This requires only *n* connections for *n* peers and also fits well with the Java Applet security requirement that applets may only connect to the server from which they were served.

The architecture, as shown in figure 3.1, is built up of a number of layers. The bottom layer is responsible for communication objects between connected processes and is known as the transceiver layer. Above the transceiver layer is the context layer. Contexts represent shared entities to which all peers may have access. Applications communicate by joining contexts and then sending/ listening to messages through these contexts. Above the context layer are the client and presentation layers. All communication within the framework is encapsulated in objects known as parcels.

Parcels

Parcels consist of a list of keys and values. The key-value pairs are termed slots. The keys are strings, whereas their associated values may be any object type. Some example of slot types include, address, sender, timestamp , priority, family, name, id, reference id, context and content.

An address contains a list of peers to which the message is directed. If the address is omitted then a broadcast is assumed. The sender slot identifies the creator of the parcel. Timestamps measure time delay, by indicating when a parcel was created. A timestamp may be significantly earlier than the time that the parcel is actually sent, as in the case where the system is busy and the parcel has to wait in queues. The priority of a parcel is an integer value expressing the urgency with which it should be delivered, higher values are given higher priority. If the priority slot is missing, the parcel is assigned a default priority. The family and name slots indicates the class to which the parcel belongs. The framework handles different types of parcels in different ways. The id and reference id are used for identifying parcels; a reply to a previous parcel would include the original parcel id as the reference id. The context slot identifies the context of the parcel (contexts are explained in detail in section 2.1). The content of the parcel is application specific and may contain any object, although in practice this must be of a type expected by the receiving peer.

An example parcel for logging into the system, converted to XML by the XML Object transceiver (see next section for explanation) is shown in figure 2. The parcel is one used for logging into the system. The content is a name value list of name, password and nickname.

Transceiver Layer

The transceiver layer sends and receives objects through a connection with another transceiver. The other transceiver may be in the same process, a different process, or on a separate machine connected via the network. If a pair of transceivers are connected over a network, then they must use a protocol to exchange objects.

Currently two implementations of network transceivers have been completed. One uses XML [17] as its communication protocol, and the other delegates the task to Java's Remote Method Invocation (RMI)[10].

The XML based transceiver is most useful for Java applets and cross language use, and was developed as a result of problems with RMI support in standard web browsers. Internet Explorer does not support RMI by default, and although Netscape Navigator supports RMI, there are problems with using RMI callbacks (see section 3.2 for discussion). XML appears to be becoming a well known standard for cross platform exchange of data, and therefore was chosen over a more concise custom language. To send an object as XML, the transceiver must know how to convert it into an XML representation. This is

```
Content-Type: multipart/related; boundary=xxxxBOUNDARYxxxx

xxxxBOUNDARYxxxx
Content-Type: text/xml
Content-ID: object

<?xml version=’’1.0’’ ?>
<parcel type=’’parcel’’>
  <properties type=’’hashtable’’>
  <key type=’’string’’>Family</key><value type=’’string’’>Request</value>
  <key type=’’string’’>Content</key>
    <value type=’’hashtable’’>
      <key type=’’string’’>Username</key><value type=’’string’’>barney</value>
      <key type=’’string’’>Password</key><value type=’’string’’>secret</value>
      <key type=’’string’’>Nickname</key><value type=’’string’’>b</value>
    </value>
  <key type=’’string’’>Name</key><value type=’’string’’>Login</value>
  <key type=’’string’’>Time</key><value type=’’long’’>946198913679</value>
  <key type=’’string’’>ID</key><value type=’’int’’>1</value>
  </properties>
</parcel>

xxxxBOUNDARYxxxx
```

Figure 3.2: A parcel converted to XML. This example is used for logging into the MOM at the start of a session. If the content was binary then this would be included as a second part of the MIME parcel. .

achieved by applications registering an XML writer that will perform this task. The XML writer may in turn recursively call other writers to serialize contents of the object. Each field of an object, is written as an XML tag consisting of the name of the field with an attribute representing its type. XML is only designed for exchange of textual data, and does not allow for the inclusion of binary data. To include binary data from an object, such as an array of bytes, the multipart/related MIME[8] type is used, binary parts are replaced by a cid tag in XML and the binary part is included as a separate attachment. An example of a serialized parcel for logging into the system is shown in figure 3.2.

There a number of disadvantages to the use of XML. Firstly, new data types must have a defined XML translation, which needs to be registered with the MOM. Secondly, XML is extremely verbose and can therefore increase bandwidth requirements. This could be reduced by compressing the data before sending, but at the cost of extra processing time. However, XML is fast becoming an accepted standard and there are many class libraries for parsing and generating it. It also has the advantage that it is human readable which makes it easier to discover errors.

The RMI implementation is useful for intra Java communication within a local network. Where higher bandwidth is available, there are no firewalls, and there is control over client software. Because all the serializing and deserializing of objects is handled by RMI itself, so no data format needs to be defined. All that is required is that RMI transceivers implement a remote transceiver RMI interface. As objects must be passed in both directions, once a client successfully obtains a reference to a remote transceiver it must register itself with the remote transceiver so that it can be ``called back".

Other implementations of transceivers are possible. An early version[4] used a structured binary protocol to send and receive data over sockets, but was found to be too inflexible to changes in data structure. A CORBA implementation is planned. This would maximise cross platform, and language interoperability. However there are still problems with CORBA support in standard browsers so there is little advantage to be gained at the moment. As discussed in section 3.2, not all clients are able to create socket connections on unprivileged ports. For these clients, the only viable communication technique is to use HTTP tunnelling. An HTTP tunnelling transceiver would communicate with a Java Servlet on the server side to relay parcels to the MOM.

Context Layer

The context layer provides the endpoint for parcels within the framework, therefore all parcels must be related to a particular context. Contexts are addressed in a similar way to URLs. They are organised in a tree structure relative to a single root context. Contexts can be managed, by a manager specified at their creation. The manager is consulted to accept or reject all requests to the context. All contexts support the same basic actions, which are join and leave. To join a context a peer must have joined its parent first.

There are three main types of context: Session, Token, and Channel. Sessions are like directories and are placeholders for a group of related contexts. Tokens are used for resource management and can be grabbed and released by peers. Access to Tokens is determined by their manager which is specified when they are created. Channels are used by peers to exchange messages. Any message sent to a channel is forwarded to all peers in the address field, or broadcast if the address field is missing. Some contexts in an example configuration are shown in figure 3.3.

To join a context a peer first join the context's parent session. Therefore a group of contexts can easily be managed by a single session parent, which allows increasingly fine grained access control for each step down the context

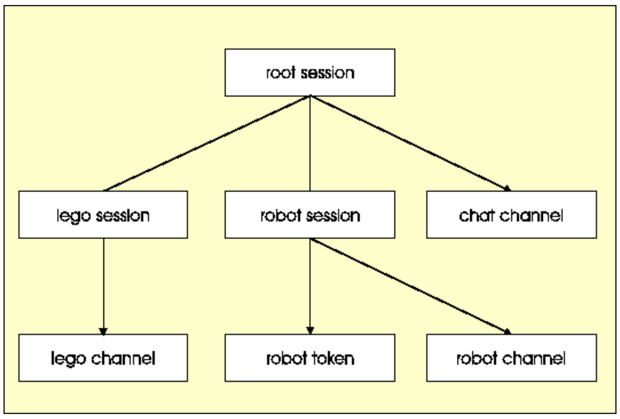


Figure 3.3: Contexts are organised in a tree structure similar to the structure of directories and files. Each type of context has a different purpose. Sessions are used as placeholders for logically related contexts, channels are used by peers to exchange messages, and tokens are used for resource control..

tree.

When the MOM is first started, only the root session exists - a configuration file may specify a manager that restricts creation of particular contexts to certain peers. Peers join the root session and then try to create or join appropriate contexts. For instance, a server that controls a robot might create a robot control token, and a robot channel for exchanging commands and information about the robot.

Tokens play an important role as resource managers in the framework. Only one peer may own a token at any one time. Exactly how the token is managed, is determined by the creating peer. For example, the robot token might be made immediately available if a peer has a higher user level than the current owner, or if the current owner has not renewed the token for a specified maximum time.

Contexts are implemented as master and proxy objects. The master object resides in the MOM, while each peer uses proxy objects that point back to their respective master. All actions performed on proxy objects are relayed to the master object in the MOM. For instance, a ``send message'' action invoked by one peer on its chat channel context is first packaged as a parcel and sent via the peer's object transceiver to the MOM, where it is received by an object transceiver and forwarded to the root context. The destination context is examined and the parcel forwarded to the appropriate master context - in this case the chat channel. As the requested operation is a channel send, it is then queued and sent to each of the peers in the parcel's address list. The MOM resends the parcel using the object transceivers for each peer.

Client Layer

To use the framework in an application, references to the proxy objects are used for all operations. Initially, a session root object is created by specifying the address of the MOM server as a URL. Once a session root has been obtained then other contexts can be found using a lookup, or created if they don't exist - and the manager allows it. Peers can then join contexts. Once joined, peers receive parcels for each context. Changes in context state are sent to peers as events. Events are a type of parcel. All contexts produce events when peers join and leave. Channels produce events when a message is received, while tokens produce events when they are grabbed or released. Sessions produce events when child contexts are created or deleted. The management and receipt of context events is usually handled by the client layer. In a simple application, the client layer may be omitted and events handled directly by the presentation layer.

3.2.2 The MOM in Practice

This framework has been used to control a robot arm via Java applets distributed using a web server. The full system consists of the MOM, a robot server, an image server, and any number of Java applets dynamically downloaded by users. On system startup, the servers log into the MOM and create a number of contexts. The applet clients then join and subscribe to these contexts to control and observe the robot.

The robot server creates a robot session, under which there is a robot channel and robot token. The token is created with a manager that will only allow change of ownership if the requesting user has a higher access levels, or if the token ownership has not been renewed for 3 minutes. Robot commands are sent to the server over the robot channel and new robot states are broadcast by the server over the same channel. The server will only accept robot commands from the current owner of the robot token.

The image server creates a camera session, under which it creates a channel for each camera. Commands to change image size and quality are received by the server over these channels. The server also uses these channels to broadcast information about new images. Image information includes the URL of the image, size and calibration matrix (for those cameras that are calibrated). The image server subscribes to the robot token and only allows the robot token owner to change image specifications.

The client applets join all the robot and image contexts as well as an unmanaged chat channel. Using the robot contexts, control of the robot can be gained or released, robot commands can be sent, and command results and robot states can be received. Using the camera contexts new images can be requested and received, and image size and quality can be changed. Finally, the chat channel can be used to talk to other users as all are subscribed to the same unmanaged chat channel.

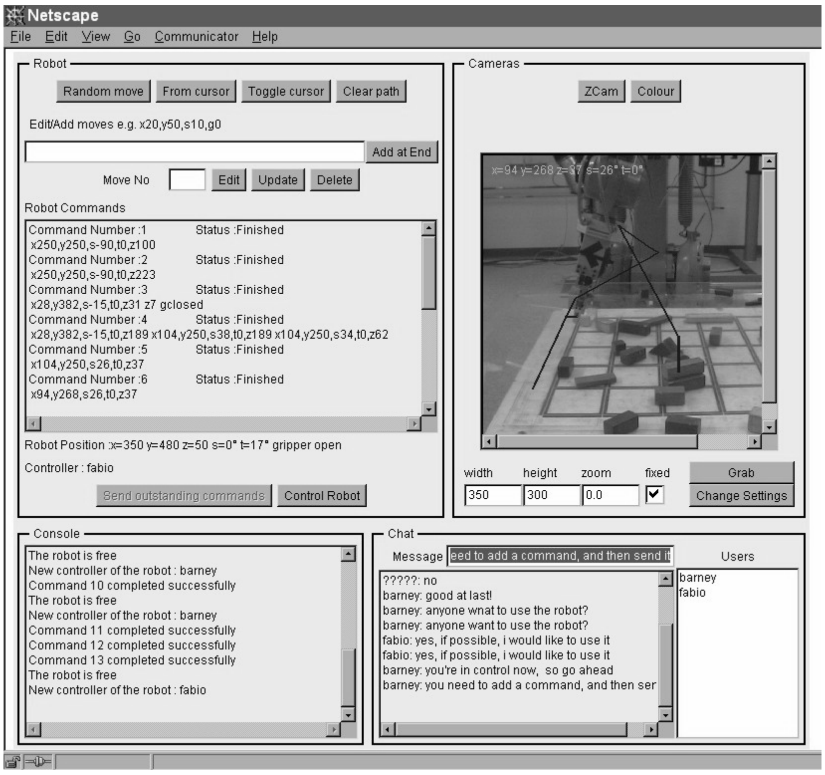
This shows just one example of the framework in action. It could equally well be applied to a mobile robot, or a simpler device such as a pan/tilt camera. For robot control, this is by no means the only configuration. For example, it may be desirable to multitask the robot in different workspaces in this case, a token could be created for each workspace, and the robot server would then use those tokens to control robot access instead of the single token currently used.

It would also be easy to integrate agents into the system that could aid the user in planning tasks, or even supervise robot motion. For example a visual servoing agent could subscribe to the robot and image contexts. The user would specify two objects to align and would give control of the robot token to the visual servo agent. The visual servo agent would then keep moving the robot and analysing new images until the desired state was reached, at which point it would return robot control to the user. The only change to the system would be to include the visual servoing option in the Applet client and, of course, to create the visual servoing agent!

3.2.3 Interface Design

Most development work has been concerned with the core framework, and hence the current user interface is only a rudimentary implementation. The interface uses some, but not all of the facilities made available by the framework. The interface, as shown in figure 3.4, has four main parts, a robot panel, a camera panel, user chat, and console area.

The robot panel lists any robot commands that the user has submitted and allows new ones to be added and edited. Commands can have a number of



Edit/Add moves e.g. x20,y50,s10,g0

Move No

Robot Commands

```
Command Number :1          Status :Finished
 x250,y250,s-90,t0,z100
Command Number :2          Status :Finished
 x250,y250,s-90,t0,z223
Command Number :3          Status :Finished
 x28,y382,s-15,t0,z31 z7 gclosed
Command Number :4          Status :Finished
 x28,y382,s-15,t0,z189 x104,y250,s38,t0,z189 x104,y250,s34,t0,z62
Command Number :5          Status :Finished
 x104,y250,s26,t0,z37
Command Number :6          Status :Finished
 x94,y268,s26,t0,z37
```

Robot Position :x=350 y=480 z=50 s=0° t=17° gripper open

Controller : fabio

x=94 y=268 z=37 s=26° t=0°

width: 350 height: 300 zoom: 0.0 fixed: ✓

```
The robot is free
New controller of the robot : barney
Command 10 completed successfully
The robot is free
New controller of the robot : barney
Command 11 completed successfully
Command 12 completed successfully
Command 13 completed successfully
The robot is free
New controller of the robot : fabio
```

Message: eed to add a command, and then send it

```
?????: no
barney: good at last!
barney: anyone wnat to use the robot?
barney: anyone want to use the robot?
fabio: yes, if possible, i would like to use it
fabio: yes, if possible, i would like to use it
barney: you're in control now,  so go ahead
barney: you need to add a command, and then ser
```

Users: barney, fabio

Figure 3.4: The Applet interface used to move the robot. The interface is divided into four parts. The top left corner refers to the robot and the top right corner shows images taken by the cameras. The console in the bottom left corner is used for returning information to the user and the chat panel in the bottom right enables users to communicate with each other.

states and representations, but currently only commands based on a simple syntax are used. Commands are queued and can then be sent once the user is ready to execute them. The status of the commands can be one of the following: waiting to be sent, sending, executing, finished, or error. If an error occurs during the execution of a command, then the details are made available via the console. Currently there is no way of stopping a set of commands once they have been sent, but this could be implemented using a high priority stop message delivered from user to robot server. As the robot moves, new poses are broadcast over the robot channel. The most recent pose is shown underneath the robot command list, and the path is drawn on the current camera image.

The camera panels provides the user with the latest images of the workspace. Different camera views can be selected using the buttons above the image. Each camera has a corresponding channel context over which new image URLs are broadcast. Commands to change image size and quality are carried over these same channels. For cameras that are calibrated various objects can be overlaid. These include the last *n* positions of the robot and an Augmented Reality cursor for specifying robot pose. It is also possible to overlay wireframe models of the blocks, although this is not active in the current interface.

The console area provides a single visible place to show all error and status messages. It is used as an alternative to the Java console available in most browsers, as most users either do not know how to show the console or, even if they do, rarely look at it. Robot errors and any unexpected communication errors are all displayed in the console. The chat area provides a basic chat interface. A single text field is provided for typing messages, on pressing ``return" messages are broadcast to all users. Received messages are echoed in the text area below.

Robot commands follow a simple scripting syntax, allowing multiple waypoints to be specified in a single command. These commands can be typed directly or generated from an augmented reality cursor displayed on the camera images (see figure 3.5). The cursor maps the 5 degrees of freedom of the robot to the 2 degrees of freedom of the image. Each line in the cursor represents a degree of freedom. By dragging part of the cursor, the user can manipulate each degree of freedom separately. To generate commands from the cursor, the user moves it to the required position and presses the ``from cursor" button. The command is then pasted to the edit window, where the user can make changes before queuing and sending it to the robot. The augmented reality cursor can also be used to measure position and orientation of objects in the image. For more details of the cursor and its application see [3].

3.2.4 Robot Hardware

The robot is an IRB 1400 made by Asea Brown Boveri (ABB). It is an industrial spot welding robot with 6 degrees of freedom. The robot is controlled by an ABB S4 controller, running the ABB Baseware operating system with a RAP module for SLIP communication. The controller also has an additional digital I/O card that is used to control the gripper. The gripper has parallel jaws that are either open or closed.

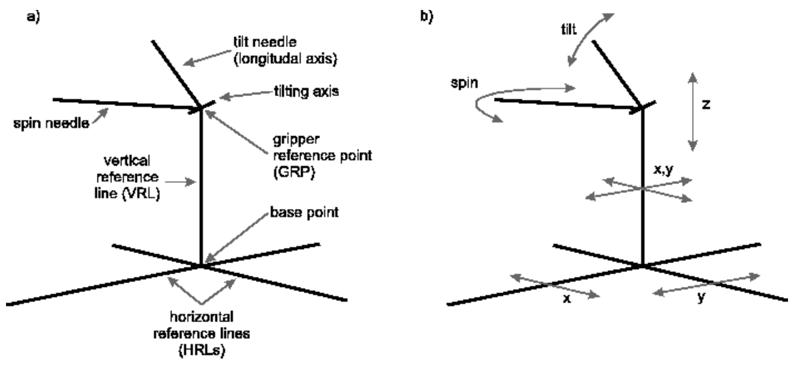

a)
tilt needle (longitudal axis)
tilting axis
spin needle
gripper reference point (GRP)
vertical reference line (VRL)
base point
horizontal reference lines (HRLs)
b)
tilt
spin
z
x,y
x
y

Figure 3.5: The augmented reality cursor used as an alternative way to specify robot commands. Each element of the cursor can be used to specify a different degree of freedom.

Detection of contact and collisions is crude, as the only method available is the detection of a current overload in one of the joint motors. This cutoff value cannot be changed. To minimise collision forces, the robot is operated at 150mm/s, a fraction of its maximum speed.

The robot is restricted to operate in a 600x600x450mm cuboid above the table. Orientation is also restricted to an inverted cone with an apex angle of 90 degrees. The orientation is further limited so that an imaginary line drawn through the gripper jaws is always in a plane parallel to the surface of the table. This intentionally reduces the degrees of freedom to 5 as the sixth is of little use for block manipulation. The two remaining orientations are termed ``spin'' and ``tilt''.

A serial link running the SLIP communication protocol connects the S4 controller to a networked computer. To execute commands on the controller, the ABB protocol RAP must be used. This is implemented in the ABB product RobComm.

3.2.5 Control

The S4 controller operates by running RAPID programs. To move the robot a simple program is executed. It first operates the gripper by switching the value of a digital output and then moves the robot to a target pose. To move the robot in a straight line, the following RAPID command is used:

```
MoveL targetpos,vslow,fine,grippertool\WObj:=table;
```

This command means, move to pose 'targetpos' with velocity 'vslow', with tolerance of final position being 'fine', using tool 'grippertool' and workspace 'table'. The 'targetpos' variable is currently the only variable that is changed for each move. For a more sophisticated application, control of the gripper speed would also be important. Currently, the speed is chosen as a balance between collision forces and path execution time.

At the suggestion of a user, an additional step was added to a basic move. If the gripper is moving down, the horizontal path is performed first, and conversely if the gripper is moving up, the vertical path is performed first. This reduces the chance of accidental collisions while executing the path, as the gripper is kept as far from the table for as long as possible.

During execution of a move there are a huge number of warnings/errors that can occur. They may be from the request itself, from communication problems between the PC and robot controller, or there may be problems associated with the actual physical move. Errors in requests, may be due to workspace boundaries or incorrect syntax. Typical physical problems include singularities, joint limits, and joint torque overloads. If the error occurs during the physical move, then the S4 controller provides an error code and some explanatory text. Often this text is quite technical and of little use to a users with little or no knowledge of robotics. For common errors, this explanation is replaced with a more understandable message.

3.3 The Setup

3.3.1 General Description

Image feedback is provided by a number of Pulnix TM-6CN cameras connected to a single Matrox Meteor II framegrabber. New images are taken whenever a new robot state is received on the robot channel, or when a user explicitly requests that a new image be taken. The position of the cameras has been varied over time: but the current positions are, one camera looking down the X Axis; one looking down the Y Axis; one looking down on the table; and one on the 3rd joint of the robot arm. The X and Y Axis cameras are calibrated. Calibration is performed by recording the position of the robot in camera images for 20 different robot poses, the camera matrix is then obtained by a least squares fit of the data. The calibration process is run as a Java application that joins the

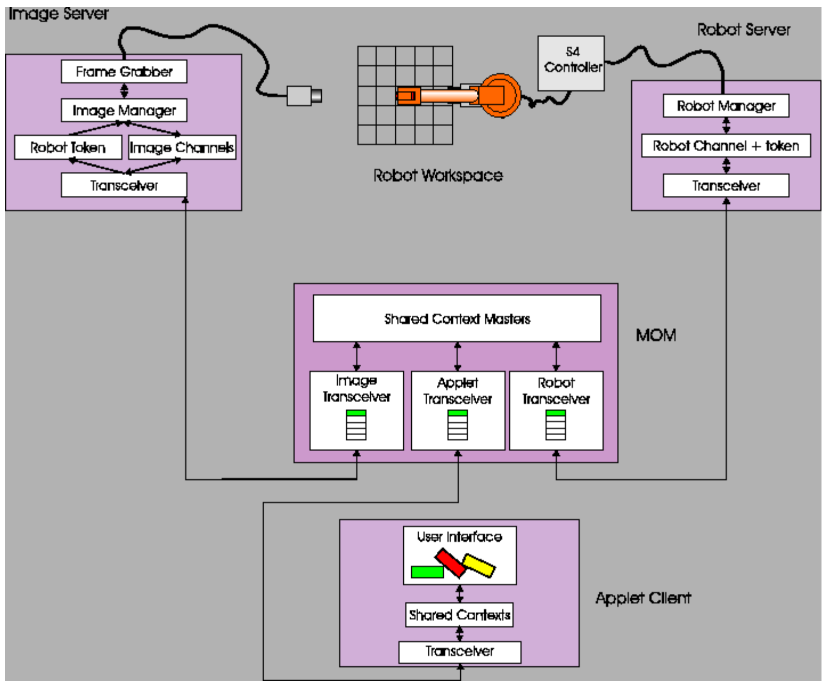


Figure 3.6: The complete system

MOM system, in much the same way as the user interface clients.

The MOM and image server runs on a Pentium II, running windows NT. The only extra card required being the framegrabber. For historical reasons, the robot server runs on a different NT machine, but could probably be run from the same machine as the MOM. The SLIP link is run on a PC running Linux. Figure 3.6 shows the complete system.

3.3.2 Experience and Statistics

Previous experience and statistics with the CGI system have been reported in [15,16,13,14]. The new MOM system has only been tested for short periods of time, as there are still a number of problems to be overcome. However, to check the network and Java capabilities of clients, a number of network tests using Java applets have been performed.

Network Tests

These network tests were designed to answer a number of questions. Firstly would user be able to connect to ports other than HTTP, and secondly what sort of network performance could be expected?

Table 3.1: Connection's made from remote hosts. Just over 13% of clients failed to make socket connection back to the server on an unprivileged port. The reason for UnknownHostException is not understood, as the client would have to have been capable of resolving the server's name to download the test applet in the first place.

Success			1162
Failures	java.net.UnknownHostException:	64	
	java.net.NoRouteToHostException: Operation timed out	56	
	java.net.ConnectException: Connection refused	39	
	java.net.NoRouteToHostException: Host unreachable	12	
	java.net.SocketException:	6	
			177
All			1339

The first question is whether a client can connect from an applet back to an unprivileged port on the server? For many networks, firewalls place restrictions on which ports can be used. If a client cannot connect on other ports, then the only communication technique available is HTTP tunnelling. Table 3.1 shows the results for 1339 unique hosts that downloaded the test applet, of these 87% were able to make a connection. The remaining 13% showed a range of errors, most of which can probably be attributed to firewalls. The cause of the UnknownHostException is not fully understood, as the client was able to resolve the hostname to download the applet in the first place. It has been suggested that this is due to proxy configurations; where all DNS requests are handled by the proxy, straight DNS lookups within the Java VM therefore fail.

Figure 3.7 shows the time taken to establish an initial connection. The median time is 0.6 seconds with a third quartile of 1.4 seconds. This time is relatively insignificant if only performed once, but if reconnection is required for each request, as is the case with HTTP 1.0, then this can start to impact on the overall response time of the system.

To test the average network characteristics once a connection was established the test applet sent and received a 50 packets of a fixed size. The size of packet for each test was chosen at random and ranged from 1 to 10000 bytes. Figure 3.8 shows the results. Each point represents the average round trip time for a particular connection, in all, almost 1700 connections were recorded from

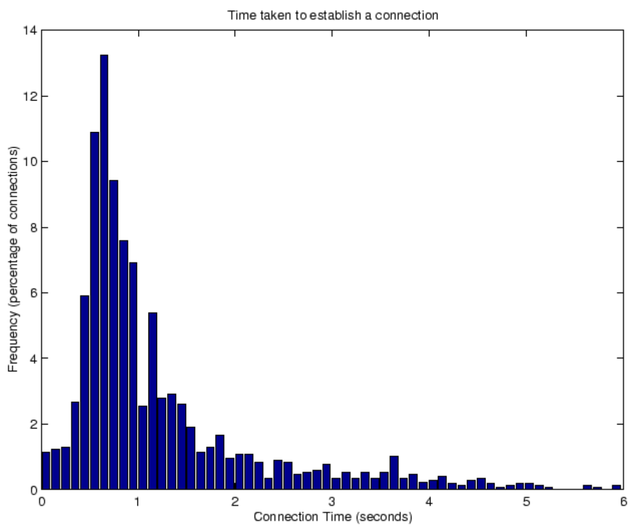


Figure 3.7: Time taken to establish a connection between a client applet and server at UWA. The distribution has a median of 0.6 seconds and a 3rd quartile of 1.4 seconds. Connections should therefore be held open wherever possible to minimise waiting time.

1339 unique addresses. The graph shows that round trip time appears to be independent of message size until the 1000 Byte range. This may be caused by all connections being carried in part by Ethernet which has a maximum transmission unit (MTU) of 1500 Bytes. This suggests that it may be better to send more data less often (as is implemented in TCP via Nagles algorithm) to achieve minimum average delay. Perhaps more importantly it shows that optimising the XML protocol by reducing size is unlikely to have little effect on round trip time provided that total message size is less than 1000 Bytes. These tests give an approximate idea of average network characterises of users. Further tests are being performed to obtain a more precise measure.

The above tests were carried out using TCP. Some tests using UDP were also tried, but there is a bug [6] in the Internet Explorer virtual machine that makes receiving UDP datagrams impossible in untrusted applets.

Results and Experiences

The system has been run and made available for some test periods, these have established that the applet works under windows 95/98/NT, Linux and Mac OS

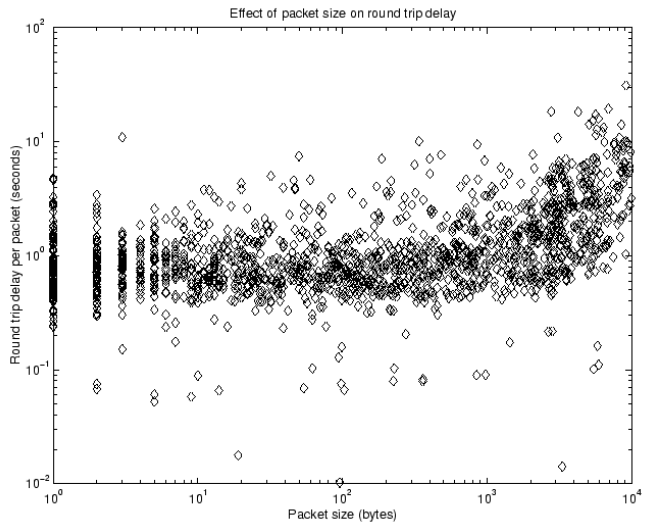


Figure 3.8: The effect of packet size on round trip delay. The overall trend is for round trip time to increase with packet size as would be expected. However there seems to be minimal increase until packet size reaches 1000 bytes

using Internet Explorer or Netscape Navigator versions 4 and higher. Users are able to move the robot and observe the status of commands, both via the command status window, the overlaid robot path, and workspace images. The chat interface also provides a chance for users to collaborate, ask for help and exchange ideas. The collaboration capabilities of the framework have not really been exploited by this first interface, but this is an area where it can provide significant improvements over the CGI version.
Some users report problems, although their platform and browser version correspond to those known to work. These problems are hard to debug as the problem machine is remote and unknown. This has led to the development of a remote logging technique, that sends log information back to the sever using CGI instead of writing to the local error log. It is hoped that this will provide more information on client problems.

Although the idea of running Java applets in a client browser is very attractive, there are many hurdles to be overcome. Developing a reliable applet is significantly harder than making a reliable Java application. The Applet must obey all security restrictions, use the lowest common denominator of Java capabilities, and also be as small as possible to minimise download time. The size of the applet is reduced significantly using the IBM Java tool JAX, the final packaged JAR file being only 130KB, less than a minute download for most users. Another potential problem is that of Java memory use. The image server used 4MB of memory as C++ application, but once a Java front end was added using JNI, the usage increased to 16MB. Java applets can also use a significant amount of memory, that for lower end client machines may render them unusable. Some Java implementations also have stability problems. The Virtual Machines (VMs) shipped with the Netscape and Internet Explorer browsers, although fine for lightweight, periodic use, prove less stable when operated for longer periods, often resulting in the browser, or even the complete machine locking up. A possible solution to this problem is the use of the Java plugin, although this requires a download of the Java runtime environment. The plugin is over 12MB which is a significant time commitment for users with slow modems. Web users are notoriously fickle in there attention span, so any extra requirements such as this are likely to reduce the use of the system.

Many Java applet problems come down to security issues. The problem of receiving UDP packets in Internet Explorer is a security manager issue that can be resolved by signing an applet and asking for higher network rights. Similarly, problems with the use of RMI callbacks in applets are caused by security exceptions, again signing the applet can solve the problem. However up until recently no single standard has existed for signing applets and therefore two separate, inconsistent systems have evolved for each of the major browsers. The Java 2 platform introduces a signing standard, but the default Java VMs of current browsers only support JDK 1.1. Hence to use the new signing standard the Java plugin must be used. Due to these signing complications it is preferable to avoid API calls that require extra security, so that applet can operate untrusted.

3.4 Conclusions and Future Work

The framework has been shown to provide the basic services required for a distributed robotic system operating over the Internet. The services include: connectivity, user authentication, addressing, resource management, time delay estimation, and ordered request delivery. These services are provided through the abstraction of shared contexts such as sessions, channels, and tokens. The configuration and behaviour of these contexts is determined only at run time, and is specified by other parts of the system that connect to the central MOM.

The framework uses a hub based architecture that consists of a central server (the MOM) and any number of peers connected to it. Thus architecture minimises the number of connections in the system, while also adhering to the security restrictions of untrusted Java applets.

Application of the architecture has been successfully demonstrated for controlling a robot over the Internet. The robot, an industrial ABB manipulator, is positioned over a table with a number of cameras distributed round the workspace. Users can use images from the workspace along with an augmented reality cursor to construct commands to send to the robot.

The framework provides a good basis for developing distributed telerobotic applications, in fact there is nothing in the framework that is specific to robotics, and it may well be useful for other distributed applications. This also means that other distributed frameworks not originally intended for use in robotics may be able to be applied to telerobot applications. For example, Sun's Java Shared Data Toolkit provides many of the same features of channels, tokens and sessions. However JSDT requires applets to be signed, and because the the JSDT libraries take up 700K, also add significant size to the downloaded applet. Distributed applications is an area of intense research and there are may be other projects that can provide useful contributions.

Although the system has been tested for short periods, it is yet to reach the stability of the previous CGI system. Therefore, immediate work will concentrate on further testing and bug fixing of the current system. Given that the design is a framework, it should be applicable to other applications. Some applications planned are: the control of a small Lego Mindstorms mobile robot, and a simple pan/tilt camera. It is envisaged that if all these robots were controlled from the same MOM, they could be used together to form a collaborative system with multiple robots and multiple users.

Development of the robot arm system could proceed in many directions although increasing the level of robot commands is a priority. This has been investigated in an interface that allows a user to interactively model objects in the environment. Commands could then be generated in terms of objects instead of position. This could be implemented by introducing a model server and a model channel, where updates and queries to the model would be performed using the model channel. Write access to the model could be restricted to the owner of the robot token or new model tokens could be introduced.

3.5 References

[1] Paul G. Backes, Kam S. Tao, and Gregory K. Tharp. Mars pathfinder mission Internet-based operations using WITS. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 284-291, May 1998.

[2] The common gateway interface. http://www.w3.org/CGI/.

[3] B. Dalton, H. Friz, and K. Taylor. Augmented reality based object modelling in internet robotics. In Matthew R. Stein, editor, *Telemanipulator and Telepresence Technologies V*, volume 3524, pages 210-217, Boston, USA, 1998. SPIE.

[4] B. Dalton and K. Taylor. A framework for internet robotics. In Roland Seigwart, editor, *IROS Workshop Robots on the Web*, pages 15-22, Victoria BC, Canada, 1998.

[5] K. Goldberg, K. Mascha, M. Genter, N. Rothenberg, C. Sutter, and J. Wiegley. Desktop teleoperation via the world wide web. In *Proceedings of IEEE International Conference on Robotics and Automation*, May 1995.

[6] Derek Jamison. Bug: Datagram socket causes security exception. http://support.microsoft.com/support/kb/articles/Q186/0/32.ASP, 1998. Microsoft Knowledge Base no Q186032.

[7] D. Kristol and L. Montulli. HTTP State Management Mechanism. RFC 2109. Technical report, Internet Engineering Task Force (IETF), February 1997. Available at http://www.ietf.org/rfc/rfc2109.txt.

[8] E. Levinson. The MIME Multipart/Related Content-type. RFC 2387. Technical report, Internet Engineering Task Force (IETF), 1998. Avaiable at http://www.ietf.org/rfc/rfc2387.txt.

[9] Robert Orfali, Dan Harkey, and Jeri Edwards. *The essential client/server survival guide*, chapter RPC, Messaging, and Peer-to-Peer. Wiley computer publishing, 1996.

[10] Java remote method invocation. http://java.sun.com/products/jdk/rmi/index.html.

[11] Michael Shoffner. Write your own MOM. http://www.javaworld.com/javaworld/jw-05-1998/jw-05-step.html, May 1998. Internet Publication - Java World.

[12] Matthew R. Stein. Painting on the world wide web: The pumapaint project. In Matthew R. Stein, editor, *Telemanipulator and Telepresence Technologies V*, volume 3524, pages 201-209, Boston, USA, November 1998.

[13] K. Taylor and B. Dalton. Issues in Internet telerobotics. In *International Conference on Field and Service Robotics (FSR 97)*, pages 151-157, Canberra, Australia, December 1997.

[14] K. Taylor and J.Trevelyan. Australia's telerobot on the web. In *26th International Symposium on Industrial Robotics*, pages 39-44, Singapore, October 1995.

[15] Ken Taylor, Barney Dalton, and James Trevelyan. Web-based telerobot-