Acknowledgements

- I would like to thank the National Research Foundation for the bursary for the year 2000.

- Sincere thanks to my mentor, Theo van Niekerk, and co-mentor, Jianan Huang, for their valuable support, motivation and encouragement during the development of this project. Thank you for your consideration and knowledgeable guidance.

- Thanks goes to my family for their support and motivation throughout this project.

- Lastly I would like to thank the Lord Jesus Christ who gave me the strength and dedication to finish the project to the best of my ability.

"The fear of the LORD is the beginning of wisdom, and knowledge of the Holy One is understanding." – Proverbs 9:10

Abstract

Manufacturing processes may be modeled in various ways, including 3D modeling. There is a need to visualise, control and monitor manufacturing processes remotely via the Internet.

Virtual Reality (VR) can be described as the science of integrating man with information. It is based on three distinct environments: three-dimensional, interactive and computer-generated. VR has come to the Internet in the form of VR modeling. The evolution of Web technologies in recent years has enabled the use of VR modeling for visualisation of manufacturing processes. The VR modeling language (VRML), which has become the standard for transmitting 3D virtual worlds across the Internet, can be used to control and monitor manufacturing processes visually.

A 3D model of a manufacturing process, specifically an industrial robot arm, was created for this project. This model was successfully linked to the industrial robot that it represents in order to control and monitor the robot's actions remotely via the Internet using Web technologies. This dissertation proves the viablity of using Virtual Reality to effectively visualise, monitor and control an industrial robot via the Internet. It also describes the methodology that was followed in modeling the industrial robot arm in VRML as well as linking the model to the real world application.

Samevatting

Vervaardigingsprosesse kan op verskeie maniere, insluitende driedimensioneel, voorgestel word. Daar is 'n behoefte om vervaardigingsprosesse op 'n afstand deur die Internet te visualiseer, beheer en monitor.

Virtuele Werklikheid (in Engels as Virtual Reality of VR bekend) kan beskryf word as die wetenskap wat die mens en inligting by mekaar aanpas. Dit is op drie verskillende omgewings gebaseer: drie-dimensioneel, interaktief en rekenaaropgewek. Op die Internet bestaan VR hoofsaaklik in die vorm van VR-voorstelling. Die groei van Web-tegnologieë het oor die afgelope jare die gebruik van VR-voorstellings vir die visualisering van vervaardigingsprosesse moontlik gemaak. VRML (Virtual Reality Modelling Language), wat die standaard vir die stuur van 3D-modelle oor die Internet geword het, kan gebruik word om vervaardigingsprosesse te visualiseer en te beheer.

'n 3D-model van 'n vervaardigingsproses, spesifiek 'n robot-arm, is vir hierdie projek geskep. Die model is suksesvol aan die nywerheidsrobot wat dit voorstel, gekoppel om die robot se aksies oor 'n afstand deur die Internet te visualiseer en te beheer. Hierdie tesis bewys die lewensvatbaarheid van sulke visualisasie en beheer van nywerheidsrobotte deur die Internet. Dit beskryf ook die metodes wat gevolg is om die robotarm in VRML voor te stel en om die model aan die werklike robot applikasie te koppel.

Table of Contents

List of Figures

List of Tables

CHAPTER 1 INTRODUCTION

You end up with a tremendous respect for a human being if you're a roboticist -

Joseph Engelberger, 1985

(McKerrow, 1991, p.1)

Bryson defines Virtual Reality as "the use of computers and human-computer interfaces to create the effect of a three dimensional world containing interactive objects with strong sense of three-dimensional presence" (1996, p.62). Virtual Reality (VR) modeling has not been extensively used for visualisation of manufacturing processes, specifically over the Internet. The evolution of Web technologies such as HTML, JAVA and VRML has paved the way to the use of real-time, interactive, 3D graphics.

Virtual environments (also known as virtual worlds) are non-real (computer generated) environments containing three-dimensional VR models. It is possible to represent complex engineering models and then manipulate and interacting with the models using a standard input and output devices to a PC, i.e. mouse, keyboard and monitor.

1.1. Problem statement

Figure 1.1 gives a visual representation of the relationship between the individual components of the research project within an integrated manufacturing plant. The following problems were identified and need to be addressed before effective modeling of manufacturing processes can be accomplished.

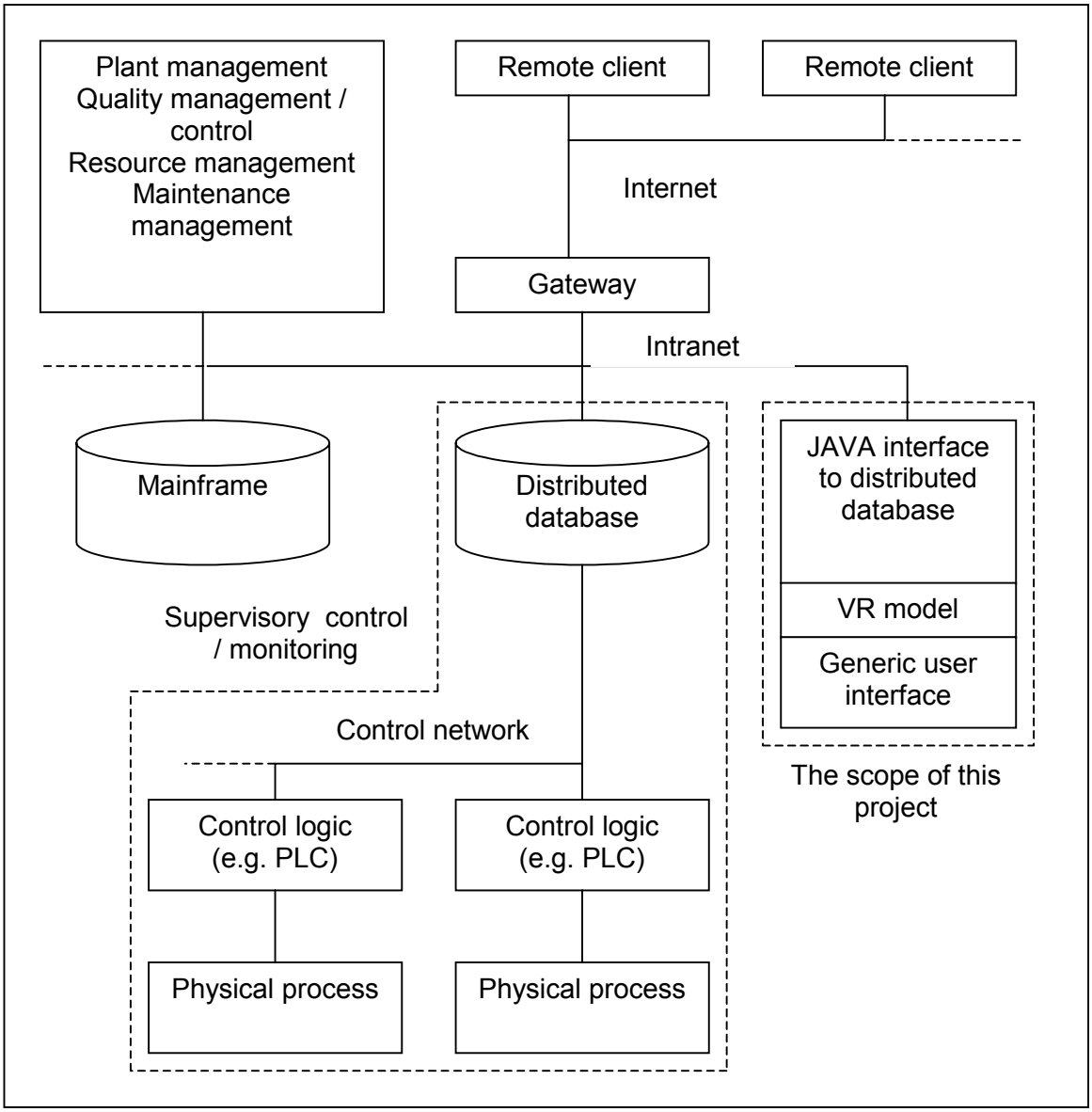


Figure 1.1 A manufacturing information system

- Investigating existing hardware and software tools for VR modeling so as to identify appropriate components needed for use in this research. Currently available VR models must be examined.
- Selecting a generic user interface, which can be used to display VR models of manufacturing processes. Sample VR models of manufacturing processes may then be built. Animation and interactivity of the models, with respect to each other and with respect to user input, must be studied.

- Selecting and implementing an appropriate interface between the VR models and a database. Review different methods of accessing the database, e.g. using the Internet/Intranet.
- Developing a software program that will be used to simulate the manufacturing processes that are represented by the VR models and integrate this with data derived from a database.

1.2. Hypotheses

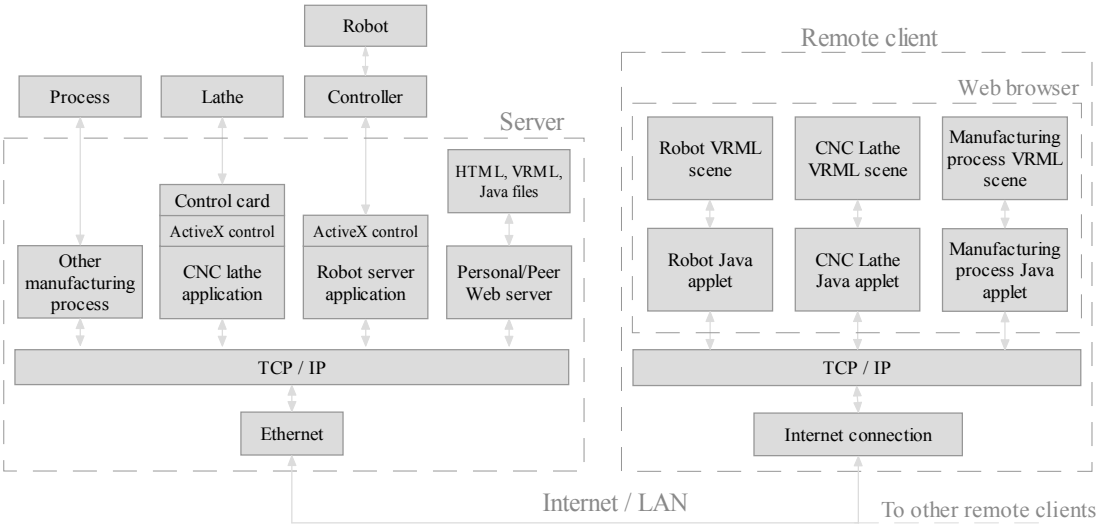
Virtual Reality can effectively be used to visualise, monitor and control an Industrial Robot via the Internet. This research project aims to prove the viability of this by making use of VR models.

1.3. Delimitation of research

The research will:

- not provide an exhaustive library of models but aims to create a model of an Industrial Robot to prove the viability of using VR for visualisation and control of manufacturing processes over the Internet;
- not include a complete study of the real-time control over the Internet;
- is limited to single-user virtual environments in contrast to multi-user virtual environments.

1.4. Outline of the system developed



1.5. Significance of the study

The research will identify the ability to control and visualise manufacturing processes remotely, interactively and in a 3D format. The research will contribute to manufacturing industry by laying a foundation for remote access of manufacturing processes.

1.6. Review of related literature

The aim of the literature study was to establish the relevancy of the research and to determine whether this area has been previously researched, and to what degree. It also aims to compile a background in the area of VR for manufacturing.

“Using VRML” (Matsuba & Roehl, 1996) concentrates on the VRML version 1.0 language details and gives examples of basic worlds. It gives a good background of the evolution of VRML as a language of the Web.

“3D Graphics & VRML 2.0” (Lemay, Couch & Murdock, 1996) focuses on 3D graphics. The second part of the book expands on the VRML version 2.0 specification and its ability to produce models with behaviours, interactivity and added functionality.

“Using JAVA” (Newman, 1996) is a complete reference of the JAVA programming language and includes many examples of creating applications and Web applets.

“Virtual Reality in scientific visualisation” (Bryson, 1996) looks at the use of VR to visualise complex numerical representations of scientific concepts. It also reviews the constraints of using this technology such as computational demands, real-time visualisation demands, data management problems, etc.

“Virtual manufacturing” (Carpenter et al., 1997) gives an overview of the opportunities and potential that VR offers the manufacturing industry.

“Using VRML to access manufacturing data” (Ressler, Wang, Bodarky, Sheppard & Seidman, 1997) describes the use of VRML in the development of a visual interface to manufacturing system. VRML objects in this application are used as an interface to a database.

“The Virtual Reality Modeling Language en Java” (Brutzman, 1998) is a good paper giving a general overview of what VRML is and how it can incorporate Java to enable the creation of powerful, interactive 3D graphics.

The study indicated that the research is relevant. The technologies to be used in the research are upcoming future technologies most of which had their origins in the 1990's to mid 1990's and is still being developed.

1.7. Thesis organisation

The chapters are organised logically, first giving a background on currently relevant issues in Chapter 2. These issues include the Internet, its history, its protocols and its technologies. It also includes industrial robotics, its applications and their methods of monitoring and controlling them. Chapter 3 gives a detailed overview of the virtual reality modeling language (VRML) with respect to its use in this project and the procedures to setup and configure a computer to view VRML files.

Chapter 4 provides an brief overview of the project's components. Chapter 5 goes on to discuss the implementation of the components identified in Chapter 4 in detail. Chapter 6 is a concluding chapter that provides results from testing the system. It also looks at different problems that were encountered with the project.

CHAPTER 2 THE INTERNET, INDUSTRIAL ROBOTS AND VIRTUAL REALITY

This chapter will serve as a background to topics related to this dissertation such as the Internet, the robotics manufacturing system and robotic interfacing software. It will also briefly discuss new technologies and trends related to these areas. These topics need to be discussed to understand the project as a whole and its relevance to the manufacturing industry.

First a brief history of the Internet and its technologies/protocols will be given. Industrial robots will be discussed by taking a look at the history of robotics, present applications of robots, different types of robots and methods of monitoring and control of robots. Web-based technologies as a means of interfacing robots and the Internet will then be discussed. Finally Virtual Reality, with emphasis on its relation to this project, will be discussed.

2.1. The Internet

2.1.1. Internet history

The origins of the Internet can be taken back many decades. Some might argue that the discovery of telegraph in 1836 or the laying of the “Atlantic cable” in 1860’s or the discovery of the telephone by Bell in 1876 was the first step towards a universal network. This may be true in a sense as these lay the foundation for modern day communications.

In 1957, however, the launching of the Russian artificial earth satellite Sputnik provided a turning point for communications (Laursen, 1997). This event urged president Eisenhower of the USA to appoint a presidential assistant for science and led to the formation of a new department within the Department of Defence (DoD) called the Advanced Research Projects Agency (ARPA). The aim was to establish the USA as a leader in science and technology within the military (Marsh, 1998).

In the early 1960's a project inside ARPA focussed on development of time-sharing in the computer process. This project was aimed at providing many people with access to powerful computers simultaneously. The vision of the director at the time was to provide "interactive computing benefiting both research and educational systems as well as the world at large" (Laursen, 1997).

1966 saw a new networking project at ARPA that would connect all researchers via dial-up telephone lines. It became known that similar work was being done in the UK by Donald Davies who came up with the principle of packet switching. The DoD realised the potential of using such a system for its command and control systems. The principle of packet switching is based on the fact that all computers in a network lies on a peer-to-peer level and each one has similar status and data forwarding capabilities (Laursen, 1997).

Data that must to be sent from one computer to another may send it as chunks of the original data known as packets. The packet contains only a few bytes of data as well as information of the source computer, the destination computer and information that would enable the receiving computer to reassemble the data packets to the original

data. This principle was important for the DoD because in the event that computers in-between the source and destination computers were damaged or made disfunctional, it would still be possible for data to be routed between the two.

In 1968 a set of specifications for a packet-switching network was defined and in September 1969 the first network computer was built. Later that year three more network computers were built and this network was called the ARPANET. It was the first long distance computer network. A set of protocols were developed in the same year to ensure that the different types of computers would be able to communicate.

The 1970's were spent evolving the ARPANET. Specifications for FTP (file transfer protocol), TELNET, electronic mail and newsgroups were laid down. The basic protocols used were also developed further as the ARPANET grew. In 1979 the first multi-user interactive sites were established.

In 1982 the Transmission Control Protocol and Internet Protocol (TCP/IP) saw the light as the protocol suite for ARPANET. The main idea was to have an underlying network control protocol (NCP) which could handle (i.e. establish and maintain) connections with others computer(s) thus providing basic communication's services. Other protocols that used the NCP would then provide services at a higher level on top of the NCP (Laursen, 1997).

The Name Server was developed in 1983. This made it easier for people to remember a specific site on the network. The focus also started moving away from one large computer per local area network (LAN) that is connected to the larger

network, to where a whole LAN gets connected to the larger network. In 1984 the domain name system (DNS) was introduced because it was easier to remember a name e.g. http://www.petech.ac.za/ than it is to remember the IP address of 198.54.155.198.

Academic institutions began to realise the importance of networking for research purposes, which prompted the NSF (National Science Foundation) in 1986 to create the NSFNET – a trans-continental network that linked 5 super computer centres to form a high-speed backbone (Laursen, 1997). These centres invited universities and educational institutions to link to them to form a decentralised network for research purposes. The NSFNET was later also used for commercial purposes.

In 1991 Gopher, a text-based user interface to the Internet, was released by Paul Lindner and Mark P. McCahill (Marsh, 1998). In October 1990 Tim Berners-Lee and Robert Cailliau handed in a research proposal entitled "WorldWideWeb: Proposal for a HyperText Project" (Laursen, 1997). The original idea behind the WWW was to "provide a distributed hypermedia system" (Marsh, 1998) for accessing resources (initially only text) on the Internet. Later, in 1993 with the release of MOSAIC, graphics was added to the list of hypermedia. This changed the face of the Internet and the commercial market really started noticing the Internet for what it was.

In 1995 new technologies such as Java, JavaScript, ActiveX and VRML saw the light as emerging WWW technologies and further as "applications" that can be run on remote machines over the Internet. An Internet browser war started between Netscape and Microsoft.

The Internet has grown at a phenomenal rate during the past 10 years with the number of hosts in January 2000 standing at 72 398 092. Figure 2.1 shows a graph of the growth of the number of hosts since 1969.

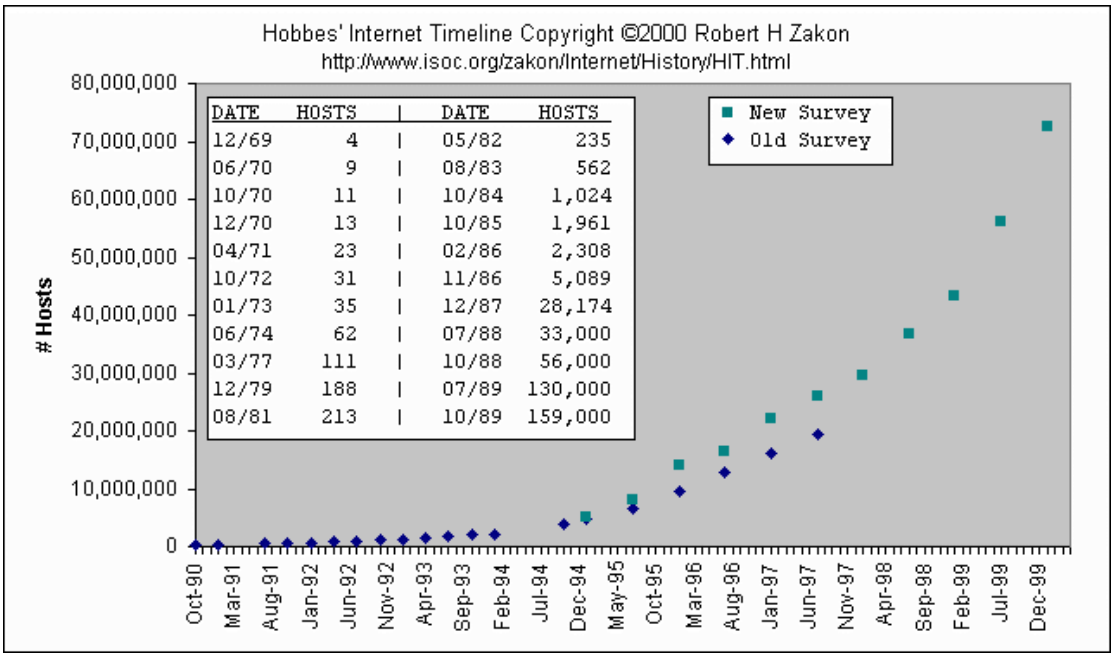


Figure 2.1 Growth of the number of Internet hosts (Zakon, 1999)

Today the Internet is being used for a multitude of applications including online shopping, research and educational purposes, file transfer, public media system, collaborative discussion forums, real-time multimedia (audio and video) and much more. It is foreseen that the growth of the Internet as a global network will continue to expand exponentially as networked, Internet-enabled devices are becoming popular over the next few years.

2.1.2. Internet protocols

Two major network reference models are used today namely the OSI model and the TCP/IP model. A reference model is used to define a set of network layers with each layer providing a specific function and abstraction from the other layers.

The OSI reference model is a general model that contains seven layers and are normally used as a basis for creating a more application-specific model. The problem with the OSI model is that the layers are not evenly matched in terms of work-load where some layer do almost nothing and others do too much.

The TCP/IP reference model, however, were designed with the connection of multiple networks in mind (Tanenbaum, 1996, p.35) and it derives its name from the two major protocols that it is built on. Figure 2.2 shows the OSI and TCP/IP reference models with typical applications for the TCP/IP model. The TCP/IP model is the one of choice that is used in the Internet today.

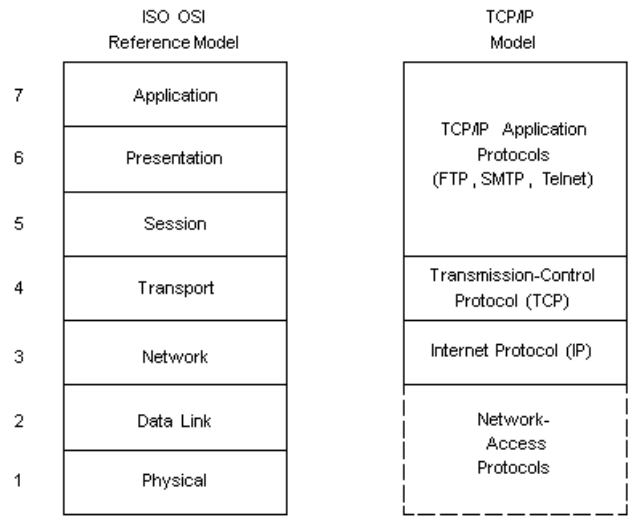


Figure 2.2 OSI and TCP/IP reference models (taken from http://webdocs.sequent.com/docs/tcpoac01/ch_1.htm)

The main protocols that the Internet is built on are the Internet Protocol (IP) and the Transmission Control Protocol (TCP). From the previous section it can be seen that this protocol suite have been used since 1982. They are very well established and an inherit part of the Internet. A brief description will now be given for the prominent Internet and application protocols.

2.1.2.1. Internet protocol

IP is an internet level protocol that provides a connectionless service (or datagram service). A connectionless service offers various advantages to its users (Stallings, 1997, p.535):

- It provides a more flexible system;
- There is no wasting of time and resources for setting up a connection before using it (this is left to upper layers, notably the transport layer).

IP provides very basic service primitives namely send packet and receive packet. It also defines a specific packet format (see also Tanenbaum, 1996, p.413ff; Stallings, 1997).

2.1.2.2. Internet control message protocol

ICMP is an internet level control protocol that should be implemented together with IP to ensure that the IP implementation is standard. ICMP messages are sent between routers, hosts and other hosts and provides information on the network environment (Stallings, 1997, p.546) e.g. when an IP message could not reach its destination because a router's buffering capacity is not sufficient. ICMP actually lies

on the same logical level as IP in the reference models. ICMP is contained in RFC792.

2.1.2.3. Transport control protocol

TCP is a transport level protocol and in contrast to IP, TCP provides a connection-oriented service. TCP provides reliable communication between two processes over reliable and unreliable networks and internets (Stallings, 1997, p.611). As was seen in the reference models, TCP is built on IP.

TCP provides some form of security as well as more complex service primitives to the application layers. Because of its connection oriented nature, TCP requires extra overhead to establish and maintain a connection but the user is ensured of an open data “pipe” once a connection has been established. The TCP specification is contained in RFC793. Typical application protocols that runs over TCP are FTP, HTTP and SMTP.

2.1.2.4. User datagram protocol

UDP is a transport level protocol for delivering packets over IP using a connectionless service. The service is inherently unreliable and no functionality for packet sequencing or error recovery are implemented. UDP is, however, much faster than TCP because of the small amount of overheads required in sending packets. A typical application protocol that uses UDP is SNMP (simple network management protocol).

2.1.2.5. Hypertext transfer protocol

HTTP is an application level protocol and the standard for transferring Web pages over the Internet (Tanenbaum, 1996, p.689). It is a transaction oriented client-server protocol (Stallings, 1997, p.721) that is built on TCP to provide reliable communication. HTTP is inherently stateless in the fact that different transactions are normally used for retrieving Web pages or other files from possible more than one distributed servers.

It is also very flexible because it can handle many file formats (text, images, audio, video, controls). HTTP makes use of an ASCII-based request-reply mechanism to retrieve and send Web pages as well as perform other, more complex, tasks.

2.1.2.6. Simple mail transfer protocol and Multi-purpose Internet mail extension

SMTP is an application level protocol used for transmitting simple 7-bit ACII text messages (electronic mail) from a sender to a receiver. SMTP is built on TCP and uses command-response sequences to transfer the mail messages. SMTP has a few limitations that make it inappropriate as a standard mail protocol (only two are mentioned here):

- The message may not contain executable files, binary objects (images or the like) or non-ASCII text;
- Some servers may reject SMTP messages over a certain size.

These limitations created the driving force in developing MIME. MIME is an extension of SMTP (RFC822) and builds on SMTP. It supports various multimedia formats in the mail message and also supports message fragmentation so that

messages will not be blocked by servers due to size restrictions. MIME is contained in RFC1521 and RFC1522.

2.1.2.7. Other protocols

Network news transfer protocol (NNTP) is a widely used application protocol for transferring news messages from newsgroups across the Internet. Telnet is a session oriented application layer protocol that provides remote terminal log-in capabilities over TCP.

The file transfer protocol (FTP) is just what the name implies a protocol specifically for uploading or downloading of files. It also uses a request-reply mechanism to accomplish file handling. Some servers require user identification and password (i.e. log-in) in order to use FTP eventhough most servers accept anonymous logins. FTP has been around much longer than HTTP but HTTP can also be used to transfer files so the need for FTP servers is not that important anymore.

The Internet is also built on various other protocols that are mainly used as control mechanisms such as address resolution protocol (ARP), reverse ARP and Internet control message protocol (ICMP). These protocols are normally not part of the Internet protocol repetua because they are completely transparent to the user.

2.1.3. Internet languages and technologies

Languages for the Internet have been developed since the inception of the Internet. In recent years, however, the number of Internet languages have grown

considerably. Each and every one of these languages have a unique place and purpose in the Internet environment. This section will discuss some currently available Internet languages and technologies.

2.1.3.1. 2D data representation languages (HTML, XML, XHTML)

The early 1990s saw the transition from text-based Internet documents to that of documents that are hypertext linked and contains images and other media. The hypertext markup language (HTML) was the mechanism that triggered this occurrence.

HTML is the standard document type and the language of choice on the Internet today. It is contained in ISO standard 8879 and its current version is 4.01. It is based on the standard generalised markup language (SGML) that has been defined in the mid 1980s (XTML 1 specification, 2000). SGML is a standard for describing markup languages and is very flexible and feature-rich. It is, however, also very complex. This posed a problem because the idea behind HTML was to create a language that scientists and technical personnel could use to generate simple documents and share it via the Internet. HTML was created from a subset of the features of SGML with hypertext and multimedia capabilities added.

Markup languages defines how documents are formatted. They contain codes (called tags in HTML) that tell the browser (or viewer) how the documents are to be displayed. It normally includes standard tags and attributes for creating hyperlinks, tables, different page layouts, toolbars, lists and many types of graphical user interface (GUI) elements (buttons, etc.). Forms can also be created using the GUI

elements. Many tools are available to create HTML documents although a simple text editor can also be used to create an HTML document.

The extensible markup language (**XML**) have been developed as a "universal format for structured documents and data on the Web" (http://www.w3.org/XML/). It is not so much focused on formatting for technical documentation as SGML. It was created after HTML but is not derived from it, but rather from SGML. The idea of XML was to create a language similar to SGML that has all the power of SGML but is simpler to use.

XML is contained within a text file and also contains tags and attributes like HTML but these are to are only to delimit the data not how to interpret it. XML is stricter in terms of syntax than HTML and does not tolerate syntax errors at all.

The extensible hypertext markup language is a move of HTML to XML. Its current version is 1.0 and it is an effort to "provide richer Web pages on an ever increasing range of browser platforms" (http://www.w3.org/MarkUp/). These platforms include cell phones, television, wireless communicators (handheld devices) and desktops. The reason for this is that it is estimated that in 2002 75% of all Internet document viewing will be on alternate (non-desktop) devices (http://www.w3.org/TR/xhtml1/).

XHTML makes it easier for content providers to provide document for various platforms while ensuring that they know how these documents will be rendered. It thus offers customisable solutions for different environments with differing capabilities (i.e. less memory on handheld devices than on desktops).

XHTML is an extensible language meaning that is may be extended through new markup. This extending capability is realised by means of modularization.

2.1.3.2. 3D data representation languages (VRML, Java3D)

The virtual reality modeling language (**VRML**) is "an open standard for 3D multimedia and shared virtual worlds on the Internet" (VRML Consortium). VRML is one of the very few languages on the Internet that is dedicated to 3D scene description and in this regard is unique to all other languages. VRML is explained in great detail in Chapter 3.

Java3D was developed by various industry leading companies, i.e. Intel, Silicon Graphics, Apple, and Sun. Java3D is a "high-level, platform-independent, 3D graphics programming API" for Java (Java3D FAQ, http://java.sun.com; Java3D community site, http://www.j3d.org/). It is specifically an API (application programmers interface) and not a language.

Java3D and VRML has a number of similarities and differences but they cannot be compared on the same level because they are inherently different. Their similarities go as far as the fact that both of them generate three-dimensional scenegraph content.

VRML is specific to content-developers where the program stays fixed but the content is variable. Java3D is specific to application-developers where the application and the content may be variable. Java3D generates all the scenes

dynamically using programming whereas VRML generates the scenes beforehand and they are only displayed (with certain exceptions as shown in Chapter 3). Because Java3D does all the scene rendering programmatically it is more powerful than VRML but with VRML it is much easier to create scenes.

2.1.3.3. Scripting languages

There are a tremendous number of scripting languages available today. Each of these have specific applications and purposes and some applications seem so specific that it seems only one scripting language would do the job. Most scripting languages, however, seems to have a similar flavour to Perl's statement that "there's always more ways to do it". Scripting languages can be broadly divided into two categories, namely general and Web-specific scripting languages.

Scripting languages share many characteristics. Most scripting languages are free and some even provide open-source code. Proper scripting languages are extensible, i.e. they are written to be extended. Scripting languages are also portable across various platforms, provide good performance, are easy to install and may or may not provide object-oriented programming capabilities.

- **General scripting languages (CGI, Perl)**

Scripting languages in this category are appropriate for general purpose computing, i.e. they are used in computer systems but are not specific to Web applications although they may be used in Web applications. A typical application of these scripting languages is to glue together compiled components into a finished application. They may combine legacy components that are well tested and used to

form a single application. Some general scripting languages will now be overviewed with specific reference to their use in the Internet.

The common gateway interface (**CGI**) is not a scripting language but is a specification that allows WWW users to run programs on a server computer (CGI Programming OpenFAQ, http://www.boutell.com/openfaq/cgi/). CGI is a gateway that allows programs written using a scripting language to be run on a server computer (note that a personal computer may also be set up as a Web server computer).

Normally the CGI script or program lies in a default directory on the server. A Web page user would issue a command (e.g. press the submit button) and the input parameters to the script would be sent to it via the HTTP POST/GET service primitive. The script would use these inputs, execute the script and return the results in the form of a Web page generated by the script and sent to the user. A typical application for this is a search engine interrogating a database.

Perl is normally the language of choice for CGI type applications but other languages such as C/C++, Visual Basic or Tcl may also be used. It is a matter of preference although Perl has excellent string-handling capabilities.

Perl (Practical extraction and reporting language) is regarded as the biggest all-purpose scripting language and the most used scripting language on Unix machines. It was initially written by Larry Wall in 1987 (Perl Mongers, 2000) but is being developed by many developers. It inherits the best features from a variety of

languages among others C, awk, BASIC and a few others. Perl has very good text manipulation capabilities and also handles files and processes well making it well suited for a multitude tasks. It is able to handle databases, has Unicode support and supports procedural and object-oriented programming (Perl Mongers, 2000). Perl can also interface to C/C++ libraries and is said to be more portable than Java.

Perl Mongers reckon that Perl is the "most popular Web programming language due to its facility with text manipulation and rapid development cycle" (Perl Mongers, 2000). It can be used for secure transactions over the Web and is the language that is used by http://www.amazon.com/. Perl is also embedded in Web servers to speed up the server.

Apart from Perl the other languages of note are Pyhton and Tcl. These may be referred to as the "big three" but many others will argue this statement. The advantages of using one of these are that they are widely supported and used by many. Each of them have their niche area where they are more powerful, e.g. Python has a "strong object-oriented model" (Laird & Soriaz, 1997) and Tcl have simple syntax and is easy to learn. Other languages include Rexx, Scheme, Lua, E-Lisp, Guile and a host of others.

- **Web-specific scripting languages (VBScript, JavaScript, WebL, Java)**

Most of the languages in this category are used for adding interactivity to Web pages. They are normally embedded inside the Web page. **VBScript** is a "fast, portable, lightweight interpreter for use in World Wide Web browsers" (Microsoft VBScript homepage, http://msdn.microsoft.com/scripting/vbscript/). It is based on

Microsoft Visual Basic and VBScript scripts can be embedded in (MS Windows) applications. VBScript is a pure interpreter language and runs embedded code inside a HTML file to add interactivity to a Web page. Applets cannot be generated using VBScript.

VBScript is only fully supported by the Microsoft Internet Explorer browser. VBScript (and JavaScript) are object-based languages as compared to object-oriented languages (Krick, 1997). This means that they use objects but does not instantiate classes or use inheritance or any of the other complexities of object-oriented languages.

JavaScript is the said to be the scripting language of the Internet and was created by Brendan Eich of Netscape. "JavaScript is the world's most popular programming language, used on more platforms and in more languages than any other programming language in history (JavaScript Developer Central, http://developer.netscape.com/tech/javascript/index.html). It is a high-level interpreter language that is easy to learn and has a syntax similar to C. Its typical application is to provide interactivity to Web pages and it uses JavaScript runtime libraries in order to process the scripts on the fly. It is used both on the client side (i.e. in a browser) and on the server side.

JavaScript is an open standard and the JavaScript engine is open source making it very easy to incorporate a JavaScript with other applications. The core JavaScript (version 1.1) was standardised to what is known as ECMA Script but these two are essentially the same, except where 3rd parties have added functionality to the core

JavaScript. JavaScript is also one of the languages that can be used together with VRML.

WebL is the former name for Compaq's Web language. This script language is was created for specific Web tasks such as rapid prototyping of Web computations and to automate tasks on the WWW. It is built entirely on Java (except the MS Windows version) and therefore supports the protocols supported by Java. It is a "high level, imperative, interpreted, dynamically typed, multi-threaded, expression, language" (http://www.research.compaq.com/SRC/WebL/). WebL is used as a stand-alone application that fetches Web pages and processes these pages according to the scripts. The unique features of the language are the use of service combinators and markup algebra. These make computations on the Web more reliable and the latter eases extraction of information from Web pages. WebL is not a commonly known scripting language.

Java is not a scripting language but was added because of its importance in the Internet and as part of this dissertation. Java is an full object-oriented language that was developed by Sun in the early 1990's. It was originally developed by James Gosling out of frustration with the limitations of C++ and was called 'oak' (Java FAQ, http://www.ibiblio.org/javafaq/javafaq.html). Later it was renamed to Java.

Java grew out the need to develop smart consumer electronics and a language that would be able to support multiple different platforms. The Java platform enables Java programs to run anywhere – from small handheld device to network computers.

The Java platform also allows for ease of communication between all connected devices and communication over networks.

The language is not based on any particular language but attempt to learn from the good and bad characteristics of other languages. For example Java does not have multiple inheritance or operator overloading like C++. It adds garbage collection (i.e. automatic recollection of unused memory to avoid memory leaks), multithreading capabilities and simplifies complex network tasks. Java allows for secure execution of Java code over a network.

Java code is compiled to interpreter byte-code. This byte-code is interpreted on-the-fly by a Java interpreter when it is run. All Java code runs on a Java virtual machine (JVM). This is also the mechanism how Java code can run anywhere. The JVM is unique for each different platform or device and written using a native language. The amount of functionality that the JVM can offer depends on the device or machine.

Java can be used to write applications (programs) or applets. Java applications are full-blown programs that can be run on the various devices as stated above. Java applets are most commonly run inside a Web browser as is the case for the applet that was created for this project.

2.1.3.4. Operating system specific Internet technologies (ASP, ActiveX)

Both of the technologies that will be overviewed next are Microsoft-specific technologies that only works on Microsoft Windows or Microsoft applications. This is a big drawback to many who would rather prefer an open standard option.

Active server pages (**ASP**) was rolled out by Microsoft in 1996 when they wanted to shift their focus to from workstations to the Internet. Microsoft describes ASP as “a server-side scripting environment that you can use to create and run dynamic, interactive, high-performance Web server applications” (Warmkessel, 1999). ASP was intended to become an open standard but this has not yet materialised.

It can be seen that ASP focuses on the server side. An ASP page is similar to an HTML page with ASP code added and the file extension of .asp. The ASP code is not intended to be interpreted by the browser but is rather for the server.

For example when doing shopping online there will be a shopping basket for the current user with all the items that the user has added so far. Each time the user requests that some item must be added to the basket, the server uses a default page (with all the HTML code for the page complete) and interprets the ASP code on the page to find the current users information in a database. Thus the user sees a custom page constructed by the server using the ASP code in the page.

ASP combined with scripts and HTML can produce highly interactive and dynamic Web pages. The scripts can be used on the server side but is normally used on the client side to perform certain client-side actions or checks.

ASP also supports the use of COM objects, either built into ASP, as part of VBScript or as part of the server system.

ActiveX is a patented Windows technology. All controls in Windows are inherently ActiveX controls. An ActiveX control can be seen as a class instantiation that offers the user some methods and properties of the class object. ActiveX controls where originally OLE (object linking and embedding) controls but changed in early 1997 when Microsoft shifted its focus with the 'Active Platform' initiative – that is, to implement Active controls over the Internet. Microsoft wanted, among other things, to enable interactive content on the Web. One of the technologies to do this was ActiveX. ActiveX is said to be the glue to tie together these various technologies to enable active Web sites but VBScript is actually the glue that binds HTML pages and ActiveX together.

To add an ActiveX control to a Web page the OBJECT tag must be used and the class identification parameter must be used to unambiguously identify the correct control. Once it is imported, it may be manipulated by a VBScript script (Detert, 1999). Microsoft Internet Explorer is the only browser that currently supports ActiveX and VBScript, which is a big drawback for the larger Internet community.

2.2. Industrial robots

2.2.1. Introduction and brief history

In 1979 the Robot Institute of America defined a robot as "a reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialised devices through various programmed motions for the performance of a variety of tasks" (Robot Institute of America, 1979). The Webster dictionary defines robot as "an automatic device that performs functions normally ascribed to humans or a

machine in the form of a human". The word 'robot' comes from a Czech playwriter by the name of Karel Capek who wrote a play in 1921 by the name of Rossum's Universal Robots. The Czech word actually means forced labour.

There is a difference between robots and automated systems. Automated systems refers to machine that are set up to perform simple, monotonous tasks e.g. along an assembly line. They may be software controlled but are designed and built for a specific task and cannot be adapted for other tasks. Robots are designed to perform many different types of tasks. The task at hand depends on the software program that it is running thus making the robot much more versatile than the automated system. Robots were not designed to resemble a human being although today many robot manipulators are in the form of an arm with a number of joints and a grasp mechanism (resembling a hand) at the end of the manipulator.

The first industrial robots were developed by George Devol and Joe Engelberger in the late 1950's. These robots were called the Unimates. Engelberger formed a company called Unimation and is known as the father of robotics. In 1961 the first industrial robot was installed in a General Motors factory. Since those early years the technology had grown in maturity mainly due the growth in the micro-electronics or semiconductor industries. In the early 1980's the robot industry grew rapidly backed by funds from the automotive industry. The growth, however, came too fast and many American robot companies had been bought over by European or Japanese companies (Dowling, 1996). The world market had grown since then and there were more than 720000 operational robot across the world by the end of 1998

where 50% of all robots are bought for the automotive industry. Figure 2.3 shows the trend between 1992 and 1998 with a forecast to 2002.

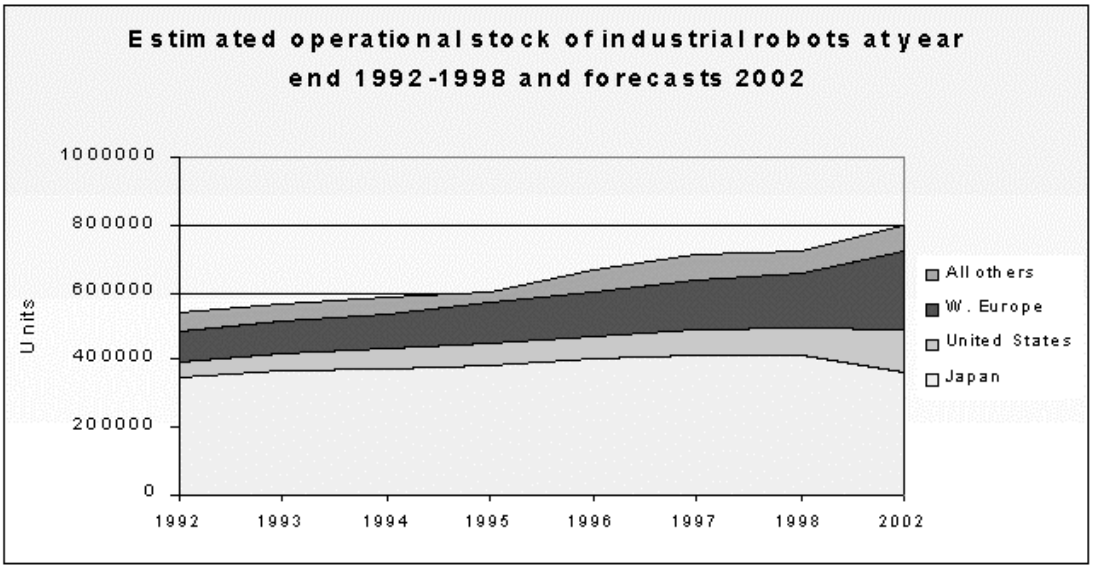


Figure 2.3 Operational robots in the world (International Federation of Robotics, http://www.ifr.org/)

Today robot manipulators are revolutionising the workplace (K-12 Outreach, 1997). Robotics can be seen as a solution looking for a problem. Although the introduction of robots in the workplace has introduced unforeseen psychological, emotional and personal problems to workers, it has benefited them and society in as a whole. For example in dangerous environments or with monotonous tasks a robot can do its job without incurring any of the problems that its human counterpart would have to deal with. Robots can work day and night without rest or food. Another advantage of robots are that the workers can do more skilled tasks.

2.2.2. A basic robotics system

A basic robotic system includes a manipulator (mechanical robot arm), the controller, production tooling, a power source(s) and a teach pendant.

The manipulator can have arm geometries of type rectangular, circular, spherical or jointed-spherical. The geometry type is determined by the type of drive elements, being either linear or rotary actuators. The number of axis of the manipulator determines the degree of freedom (DOF) of the robot. A spherical-jointed 6 DOF manipulator will use the first 3 axis (waist, shoulder and elbow) to position the tooling anywhere within the work envelope. The other 3 axis (pitch, yaw and roll) are used to get the tool in the right orientation.

The production tooling is fitted to the toolplate at the end of the arm. The tooling is called end-of-arm tooling, end effector or gripper. Depending on the robot and end effector, the power sources required may include electric, pneumatic and/or hydraulic systems.

The robot controller consists of one or more CPUs networked together, memory, inputs, outputs and one or more serial communications ports. The teach pendant is connected to the controller via some serial interface and is the main interface to control the robot actions. Some controllers has a teach terminal on the controller with an additional external teach pendant.

2.2.3. Robot kinematics

McKerrow defines kinematics as “the relationships between the positions, velocities, and accelerations of the links of a manipulator” (McKerrow, 1991, p.176). Fu et al. (1987, p.12) adds that the manipulator motion is “without regard to the forces/moments that cause the motion” (this area is known as robot arm dynamics). Kinematics involves setting up mathematical frameworks that describe these relationships. In this research project the focus is on finding relationships between the positions and orientations of the links / joints of the manipulator.

The relationship that is of particular concern in robotics is the transformation matrix, ${}^{R}T_{H}$, which relates the hand (or tool) coordinate system to the base coordinate system (coordinate systems will be studied later in this chapter). The base coordinate system is fixed (creating a reference system) while the hand (coordinate system) is able to move in free space.

Robot arm kinematics presents two fundamental problems, namely the direct (forward) kinematics problem and the inverse kinematics (arm solution) problem. In direct kinematics the joint angles of the robot is known or given while the required position and orientation of the end-effector must be obtained. In inverse kinematics the position and orientation of the end-effector is known or given and the joint angles must be calculated. The two problems are diagrammatically presented in Figure 2.4.

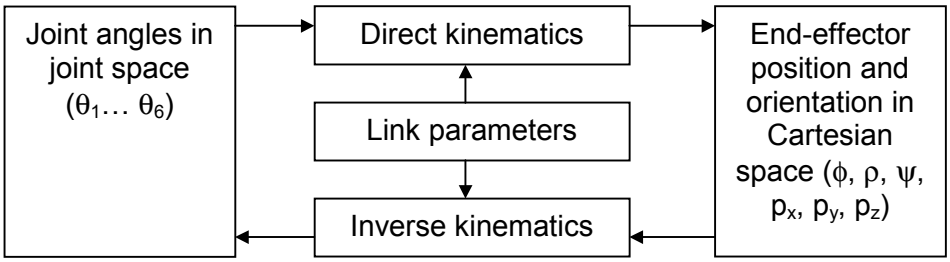


Figure 2.4 Direct-inverse kinematic relationship

There are various methods to solving the direct kinematics of a robot arm (Aspragathos & Dimitros, 1998). Certain basic concepts need to be understood in order to follow the homogeneous transformation matrix method that was used in this project. The algorithm to solve the direct kinematics and a detailed solution for the IRB1400, modelled in this project, is shown in Chapter 5.

Trigonometric (or geometric) methods to solving the direct kinematics can effectively be used for simple manipulators. For more complex manipulators, the trigonometric relationships becomes difficult to visualise, therefore homogeneous transforms are used (McKerrow, 1991, p.187). Megahed argues that the homogeneous transformation matrix is the base of all computations related to robot arm modeling (1993, p.31). The computational cost, in terms of processing time, in kinematic calculations is the main factor that determines which method is best (Megahed, 1993, p.31).

It should be noted that the results are always the same, irrespective of the method being used. The method that was used in this project is based on the homogeneous transformation matrix. In 1955, Denavit and Hartenberg described a method of systematically assigning a coordinate frame to each link of an articulated robot arm

(Fu et al., 1987, p.36). This so-called D-H representation results in a 4x4 (general) homogeneous transformation matrix that describes one link's coordinate system (situated at the joint) with respect to the previous link's coordinate system. By using matrix multiplication, a homogeneous transformation matrix can be obtained that describes the tool centre point position and orientation with respect to the base (or world) coordinate system.

2.2.4. Coordinate systems, joints and links

Several coordinate systems are used in robotic systems, namely the world-, base-, joint-, object-, user-defined- and tool (or hand) coordinate systems. The world coordinate system is a Cartesian system with an origin at an arbitrary point in Cartesian space. The base coordinate system has its origin at the base of the robot manipulator. The reason why a world- and a base coordinate system is required, is to facilitate a system with two or more robots or when the robot is attached to a carrier (ABB, 1998, p.2-15). In this case, the world coordinate system would provide the reference point to the system.

A joint coordinate system is fixed to each joint at the end of the link. The tool coordinate system specifies the tool centre point and orientation. The user-defined coordinate system is an arbitrary coordinate system that the user can program as a reference coordinate system. An object coordinate system defines a specific work-area that can be programmed in tool coordinates. This is done so that when another tool is used, the application can still be performed. Figure 2.5 shows the different coordinate systems with respect to the IRB1400 robot manipulator.

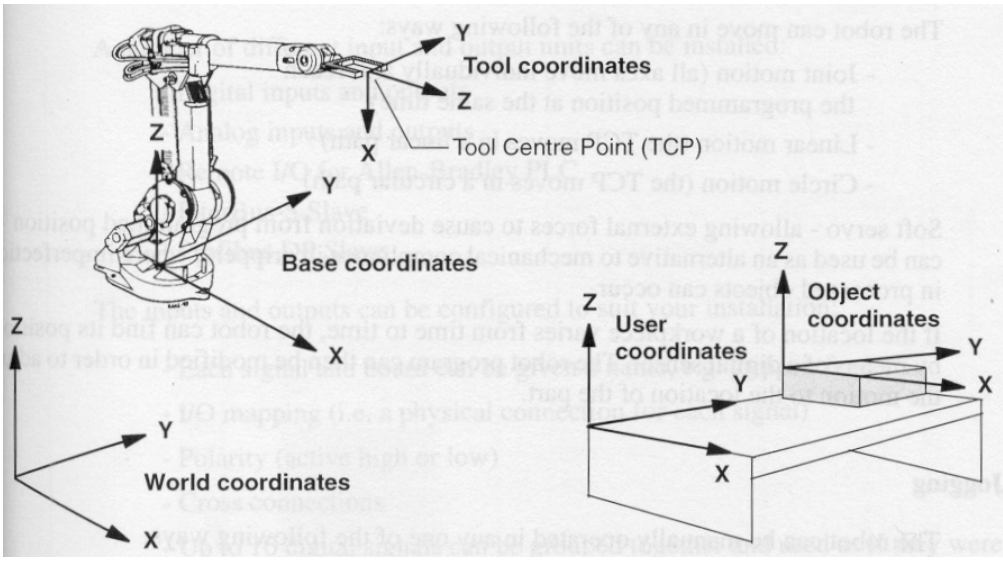


Figure 2.5 Coordinate systems in IRB1400 (ABB, 1998, p.2-15)

A link can be described as a solid mechanical object connecting two joints. It maintains a fixed relationship between the two joints at the ends. The first and last link of the robot manipulator only has one joint. The joint at the end of the link is called the distal joint and the one at the other end is called the proximal joint. Robot links are normally designed to be of a certain type of which various standard types exist. This is done so it is easier to manufacture and control the robot manipulator (McKerrow, 1991, p.180).

Revolute (or rotary) joints and prismatic (or sliding) joints are the common joints found in robotics. A joint in robotics only has one degree of freedom (unlike the human arm). This simplifies control and kinematics of the manipulator. Revolute joints provide one degree of rotation and this rotation is about the axis of the proximal joint. Prismatic joints provide one degree of translation and this translation is coincident with the centre line of the sliding link. Similar to links, there are theoretically an infinite number of joint configurations but for practical reasons,

common configurations are normally used. The IRB1400 only consists of revolute joints.

2.2.5. Categories of industrial robots

Industrial robots may be categorised in many different ways. The main ones would be categorisation according to mechanical structure and applications to which they are well-suited. Each type of robot has a unique mechanical structure producing a unique work envelope (or workspace). Five basic work envelopes have been considered namely rectangular, cylindrical, spherical, near-cylindrical and cone-shaped envelopes. The envelope determines the area in which the gripper will be able to move and do work. Each work envelope uses a certain coordinate system. The categories include:

- Cartesian robot

The rectangular (or Cartesian) coordinate system uses 3 linear axis with prismatic joints for positioning and 0-3 rotary axis for orientation. The orientation axis are not always necessary or required and depends largely on the application (This is true for all the robots mentioned in this section). The work envelope for this robot is rectangular-shaped. This type of robot is mostly used in a gantry type applications.

- Cylindrical robot

This robot uses 2 linear axis with prismatic joints and 1 rotary axis for positioning and 0-3 rotary axis for orientation. This type of robot has axes that form a cylindrical coordinate system (International Federation of Robotics, http://www.ifr.org/). The work envelope for this robot is cylindrical-shaped.

- Spherical robot

This robot forms part of the spherical (polar) coordinate system and uses 1 linear axis and 2 rotary axis for positioning and 0-3 rotary axis for orientation. The work envelope for this robot is spherical-shaped.

- Articulated (or spherical-jointed) robot

The jointed-spherical coordinate system uses 3 rotary axis for positioning and 0-3 rotary axis for orientation. This type of robot can normally be used in a very versatile manner. The work envelope for this robot is also spherical-shaped as the spherical robot.

- Parallel robot

This type of robot have arms that have concurrent prismatic or rotary joints. The work envelope for this robot is cone-shaped.

- Selective Compliant Assembly Robot Arm (SCARA) robot

The SCARA robot typically has 4 axis of which 3 are parallel to each other. These parallel axis are normally vertical. The parallel joints are revolute joints and the other joint is prismatic (either at the wrist or at the base). These robots tend to be much faster than articulated robots (Robot Encyclopaedia). The work envelope for this robot is also near-cylindrical shaped.

Figure 2.6 shows a table of the different types of robot geometries that were discussed above.

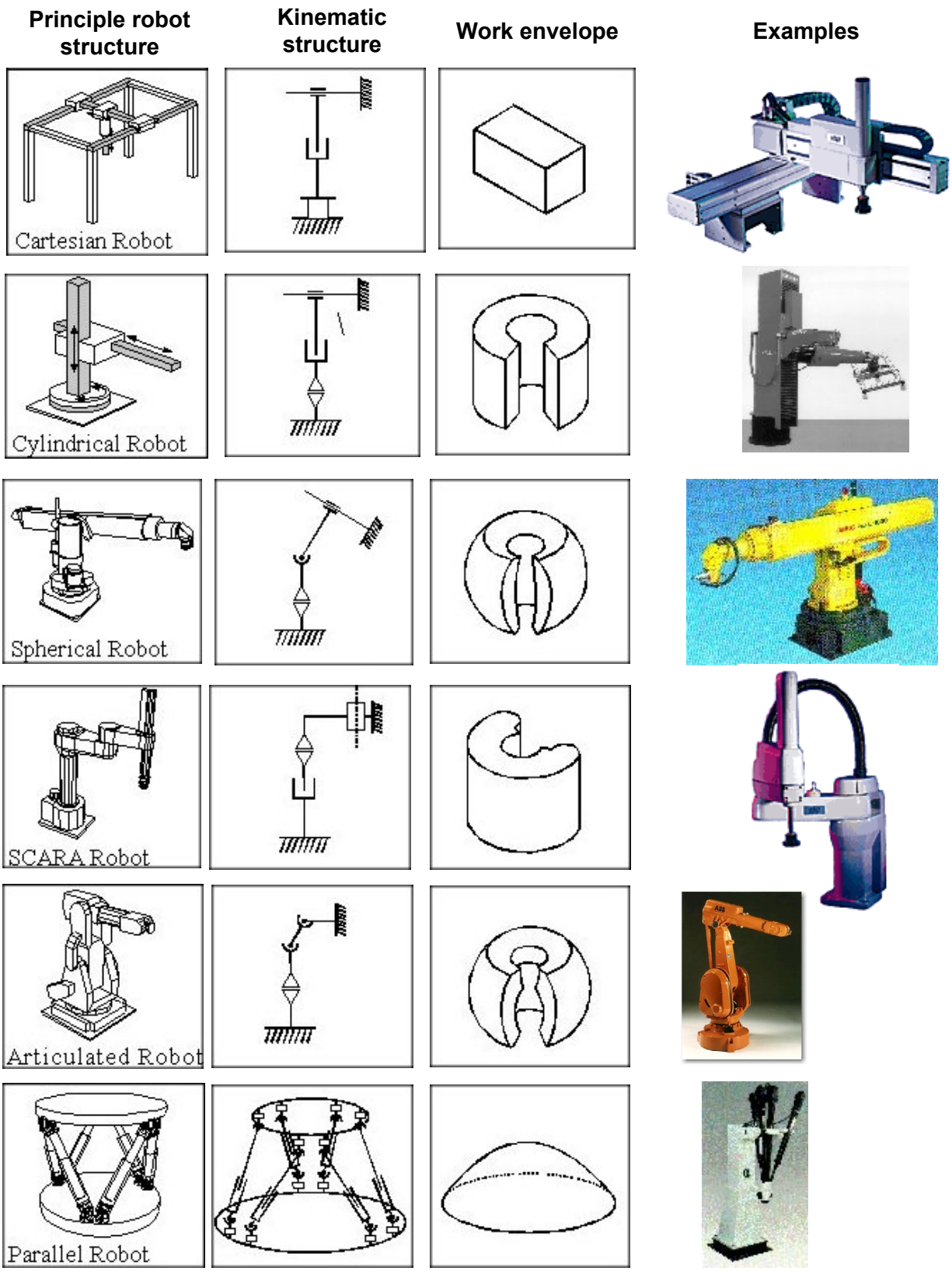


Figure 2.6 Various different (according to structure) types of robots

Three types of power sources are primarily used namely hydraulic, electric and pneumatic sources. Each power source has its advantages and drawbacks and also depends on the application.

Hydraulic power sources can pick up heavy loads but the oil leaks eventually, extra equipment is necessary and the oil is a fire hazard in arc welding applications. Pneumatic power sources are normally used in actuators that have only two possible positions namely retracted or fully extended, e.g. for opening and closing a gripper. These power sources are inexpensive and well developed and system leaks do not contaminate the work area. Pneumatic sources suffer from the fact that they are not normally proportionally controllable. Electric power sources uses electric motors (either stepper or servomotors).

Areas of application can be roughly subdivided into assembly, spot and arc welding, spraypainting and other non-assembly tasks. Non-assembly tasks include material handling, spraypainting and coating as well as machine loading and unloading. Some of these applications will now be briefly discussed with reference to robotic systems.

- Welding

Spot and arc (including metal inert gas and metal arc) welding is one of the most common uses of industrial robots in industry and industrial robots were first used for this type of application in the 1970's. They account for nearly 25% of all robot applications. Using robots for this type of application has various advantages including (Bolmsjo, 1989):

- No need for expensive qualified welders;
- No workers need to be in an unhealthy environment (fumes, etc);
- Higher productivity and efficiency;
- Increased flexibility; and
- High and consistent quality.

Robots with articulated joints are normally used for these type of applications. These type of robots also normally has an "open kinematic structure" (Bolmsjo, 1989) implying a larger workspace and distributed drives. The standard components for welding robots are the robot arm, robot controller, welding power supply, wire feeder and positioning equipment (http://www.jcdrobotics.com/apps/migwelding.htm). Welding robots need to have good repeatability, as they must weld the same set of spots over and over again on different workpieces. Knowledge-based systems and off-line programming are sometimes incorporated with this type of application.

- Assembly

Thirty-three percent (up to 1997) of all operational robots are used in the field of assembly. The industries that utilise assembly robots are mainly the automotive and electronics industries. Assembly tasks involve taking components or items (that may not be homogeneous) from different locations and assembling them according to certain criteria. An example of an assembly task is populating a printed circuit board (PCB) with electronic components before the PCB goes through the solder bath. Robots that do such assembly need not be of complex mechanical structure depending on the specific type of assembly that is done.

- Spraypainting

The area of spraypainting is most commonly found in the automotive industry. There are many advantages in replacing humans that to do spraypainting. Humans normally do a better job in such a delicate area as spraypainting. But the environment in which they must work is harsh because of toxic air and high noise levels. This type of environment is perfectly suitable for robots. It is, however, difficult to teach robots how to do the job and sometimes only 70-80% of the surfaces can be done by a traditional spraypainting machine and they tend to be more wasteful of paint (http://www.morph.demon.co.uk/Electronics/robots.htm). Regulations are making it more difficult to use humans for spraypainting tasks and it is foreseen that the automotive industry and robot manufacturers will work together to fill this gap in the market for better spraypainting robots.

- Other application areas

Food handling is a growing area in which robots can be used extensively and is currently seen as an emerging area of potential. Biorobotics is another area where robots are being used to do micropropagation of plants (International Federation of Robotics, http://www.ifr.org/). Packaging and palletising is a growing application for industrial robots with 2.8% cut of all robots and it is estimated that this area will grow as industrial robots become easier to handle.

Two types of control are singled out namely open loop and closed loop control. Closed loop control continuously monitors (with the use of sensors) the tool position and velocity with the required position and velocity on the desired path of the

manipulator. Open loop control does not monitor the actual required position but uses limit switches to monitor the position at each limit (called limit sensing).

Closed loop control is used in applications such as welding, coating and assembly where the path must be controlled closely. These system are more complex than open loop systems and generally costs more. Pick-and-place robots normally incorporate open loop systems.

Four types of path control can be identified namely stop-to-stop, continuous, point-to-point and controlled path control. Path control specifies the way in which the manipulator guides the tool along its trajectory path.

- **Stop-to-stop path control** operates in open loop and therefore the position and velocity is not known beforehand. The position of the axis is known when the actuator is driven to a limit. Only the sequential on/off commands of the actuators is saved for path control.

- **Point-to-point path control** is obtained by using a teach pendant to record certain points (position and orientation information) in the work envelope and running the robot so that it moves the tool centre point (TCP) in sequence through these points. It does not matter what path is followed while programming the points. Each axis moves to its next destination (to get to the programmed point) as fast as it can. This causes the TCP to take a non-straight line path, which is a disadvantage in applications such as arc welding.

- **Controlled path control** is similar to point-to-point path control but it has the added advantage of offering a straight-line motion of the TCP. It does so by increasing the rate of change for an axis requiring the most change and decreasing the rate of change for an axis requiring the least change when moving from one point to the next. This causes all the axis to reach their final destination at the same time.

- **Continuous path control** is similar to point-to-point path control but differs in that many points are saved instead of just programming individual points. This is done because it not only stores the points after the programmer reaches the destination position and orientation but also stores the path as the programmer moves the manipulator until the final destination is reached. This type of path control has applications where the manipulator must duplicate the skills of the operator, e.g. in spray painting applications.

Robot controllers can be classified as either high, medium or low technology controllers. It depends on the versatility of the controller. Previously the controller's computing power was used as a measure of the robot's intelligence because processing power was expensive. Nowadays processing power is not really a cost factor.

2.2.6. Current methods of remote monitoring and control of industrial robots

There are currently not as many methods of monitoring and control of robots as there are with other factory machines and equipment. The amount of monitoring and control that is required by a robot depends entirely on the application. In the

applications that were mentioned in the above section, the robot only needs to be programmed once. This can be done by online programming (i.e. where the operator uses a teach-pendant to program the robot while it is connected or by means of a connected computer) or by off-line programming where the operator uses some visual programming package to program and simulate the robot program and then download it on the robot controller.

Monitoring and control of a robot which has been programmed can be done by using some feedback mechanism to the robot controller, e.g. with visual feedback and a processor that does recognition and then performs adaptive control by manipulating the program or communicating with it. This type of operation would result in the robot performing tasks autonomously without any user intervention.

A more traditional method is the use of remote manual teleoperation where the operator physically controls the robot by means of the teach-pendant. This method has been used for decades because of its effectiveness in hazardous environments (Milgram et al., 1993). This method is viable for large variety of applications.

Remote manual teleoperation is advantageous since the human can do the job effectively but the disadvantages are that the operator may get tired and he is continually being occupied. The advantage of the autonomous robot method of control is that the robot can do the task much faster but it takes time for the system to learn to handle variations in the system or tasks.

The best method of control seems to be where the robot's best attributes such as speed, accuracy and repeatability are merged with the knowledge, creativity and skill of the human to the task at hand (Milgram et al., 1993). This has been researched to some degree and used in the medical field. In America patients who need hip surgery have been successfully been operated on by a doctor assisted by an industrial robot (Ropp, 1993). This should be commonplace in the not too distant future.

Control of applications and devices on the Internet is part of a major new leap of the new Internet (especially with the coming of Internet Protocol 6). Telerobotics on the Web has also emerged as researchers try to merge robotics with the Internet. Telerobotics can be defined as "robotics at a distance" (http://gypsy.rose.utoronto.ca/projects/telerobotics.html).

Normally telerobots on the Web may be programmed by many users on the Internet. A Web page with a picture of the current status and position of the robot, which includes the actual values of the axis of the robot, are normally the user interface to the robot. The "feedback" that these robots provide are the pictures (possibly from more than one angle) and status information. The user interfaces differ but normally allows the current user (teleoperator) to change robot characteristics by submitting new positional robot information. This information is used to move the robot to its new position to perform some arbitrary task. The various examples that will be shown differ mainly with regards to the type of environment that the robot is situated.

- Australia's telerobot on the Web

The University of Western Australia's Robotics Lab has an ABB industrial robot that is connected to the Internet. It is possible to control the robot by issuing commands to it by means of a CGI interface. The server processes these commands and moves the robot to the desired position. The workspace consists of a table with wooden blocks on it which the robot may pick up and place anywhere within the workspace. The server is intelligent enough to recover from collisions, and to keep the robot in the desired workspace. There is also a Java-based control environment from which users may control the robot. The robot is accessible online at http://telerobot.mech.uwa.edu.au/.

- The Telegarden

The telegarden is a telerobotic installation that allows Web users to view and interact with a remote garden filled with living plants (http://telegarden.aec.at/index.html). The garden has a radius the length of the robot's arm. Online users are allowed to plant, water, and monitor the progress of seedlings via the tender movements of an industrial robot arm. The robot has a camera mounted at the tool centre point of the robot and shows this view every time the Web page is started or refreshed. The robot is able to sprinkle water or to plant a seedling on the shown spot. The telegarden was developed at the University of Southern California and went online in June 1995.

- Intelligent manufacturing cell

Dr Jan Naude of the Rand Afrikaans University has developed a system in which an industrial robot is used in an assembly and material handling application. The robot

performs a series of actions that is controlled by an expert system with feedback circuitry. A three dimensional Java wireframe model of the robot and workcell have been created. The robot and workcell are connected to the network server and the model application runs on the server computer thus making it possible for Internet users to view the current status of the robot in its environment. It is also possible for users to do limited control of the robot without requiring any knowledge of the application, robot or programming because of the easy graphical user interface.

Many more of these online telerobots can be found on the Internet. They are just a small preview of what lies ahead in future for the area of robotics and the Internet.

2.3. Virtual Reality

Virtual Reality (VR) has been described in many different ways but maybe the most expressive of these is when VR is seen as the science of integrating man with information. VR does not necessarily aim to recreate reality as we know it but rather create an artificial reality. Three distinct characteristics of VR are that it is three-dimensional, interactive and computer-generated (Bryson, 1996, p.63). This implies that the user of a VR system is immersed in a computer-generated environment (also referred to as a virtual environment) where the user is allowed to interact with the environment or objects therein. VR grew “from the basic human desire to explore alternative realities” (Loeffler & Anderson, 1994, p.xv).

In contrast to systems where the user goes though predefined environment, a VR environment enables the user to follow any action or interactivity. VR systems are simulation environments that aim to involve as many senses of the user as possible.

This is done by means of sophisticated hardware and software. The hardware includes various inputs and outputs. These include, among others, head-mounted displays (HMD) with stereo-scopic vision, datagloves that may contain force-feedback or motion sensors for manipulating objects in the virtual environment and flying mouse for six degrees of freedom movement and control. The software is in the form of a computationally expensive real-time reality engine that uses the inputs to provide a proper output. This means that a user that wears a HMD and moves his head back and forth will cause the reality engine to generate an output such that the user thinks he is looking back and forth inside the virtual environment as well. It must be clearly stated that VR does not necessarily have to contain all the above hardware and software and that a personal computer may also provide a form of VR.

Loeffler & Anderson describes VR as a “technology in search of an application” (1994, p.xiii). This is indeed so because of the endless possibilities and capabilities that VR offers. The areas of application of VR is limitless and many disciplines are using VR for professional applications. In engineering, specifically the automotive industry, vehicle prototyping is done using VR systems to save costs and time for new vehicle designs. In education VR systems can be used to explain difficult concepts e.g. molecular structures. The military have used VR for many years for training pilots in flight simulators. In the medical field VR is used to test the properties of new medicine and for practising difficult surgery. The architectural community uses VR for designing new buildings. Lastly VR is used for recreation and it seems like this area will grow considerably in future.

2.4. Conclusion

This chapter discussed issues relevant to this project and served as a essential background to understand the importance and usefulness of this project to industry. The topics that were discussed are the Internet, industrial robotics. The chapter concluded with a short background on virtual reality. The next chapter discusses the Virtual Reality Modeling Language in detail and serves as a background to the VRML implementation for this project.

CHAPTER 3 IMPLEMENTING VRML

This chapter provides the background and basis for the VR model that was created for this project. The VR model serves as the starting point in providing the 3D remote access to the industrial robot used in this project.

This chapter gives a detailed description of what VRML is and its historical background. The required software to use VRML will be discussed as well as the steps in setting up the PC to view VRML. It explains the file format and language structure and includes practical examples to briefly show the usage of VRML. The different methods of extending VRML are discussed and compared. Various application areas of VRML are considered and practical examples are shown. The chapter concludes with a look at future trends and developments of VRML.

3.1. Introduction

VRML (also pronounced as ”vermel” by some programmers) is an acronym for the Virtual Reality Modeling Language. It is a scene description language that describes dynamic three-dimensional (3D) environments for platform-independent virtual reality applications over the Internet (Lemay, 1996, p.14). These environments are generally referred to as VRML worlds or scenes. VRML can be seen as the medium to display and interact with 3D worlds, as HTML is the medium to display hypertext files when browsing the Web. The language format is based on an extended subset of the OpenInventor file format from Silicon Graphics (Brutzman, 1998, p.57).

In VRML version 1.0 it was possible to create static worlds in which the user could navigate through the world without limits. The user could also interact with the world where the “interaction” consisted of clicking on links leading to other worlds (VRML files or HTML pages) similar to HTML links. Examples of static worlds that could be created were museums, architectural walkthroughs and building plans. This was not really a virtual reality (VR) language because of this lack real of interaction.

VRML 2.0 extended version 1.0 in terms of user interaction to provide dynamic worlds with a greater sense of immersion. It added behaviours and programming language support for JavaScript and Java (Nadeau, 1997, p.6). VRML 2.0 does not support multi-user virtual environments in the language specification although it is possible to build such environments using VRML 2.0 as the basis (Lemay, 1996, p.193).

3.2. History and background

In 1994 Mark Pesce and Anthony Parisi developed a prototype program, called Labyrinth, which showed the use of a platform-independent 3D graphics file format. An open working group examining the potential of a “virtual reality markup language” was established as a result of sessions held at the First International Conference on the World Wide Web (1994) and SIGGRAPH. The name “markup” was later changed to “modeling”. The original idea of VRML was to find a suitable 3D scene description file format that could be changed to incorporate hyperlinks as used by HTML (Brutzman, 1998, p.57). The file format had to describe 3D object geometry, attributes and where these objects were placed in 3D space. The files had to be

deliverable across the Internet and be platform-independent, i.e. they had to run in a Web browser on any platform.

Work on the first VRML version was started in late 1994 and the final specification for VRML 1.0 was released in May 1995. It was largely based on the OpenInventor file format from SGI and included support for shape building, texturing and lighting (Nadeau, 1997, p.1). In late 1995 VRML 1.0 Web browser plug-ins became available but none supported all the features of the specification. Some ambiguities in the specification led to VRML 1.0c (clarified version), which cleared out these problems but no new features were added. VRML 1.0c specification was released in January 1996. Attempts were made to add language extensions for VRML 1.1 but these were not implemented because it was felt that it would be better to press on to VRML 2.0.

As a stepping-stone towards VRML 2.0, the Moving Worlds proposal was submitted in January 1996 to the VRML Architecture Group (VAG) and it included extensions for interaction and animation. Seven months later (4 August 1996) the final VRML 2.0 specification was released. This version redesigned the language syntax and added many new features that include 3D sound, video, fog, programmable behaviours and language extendibility (Lemay, 1996, p.193). The VRML 2.0 specification was given up for standardisation and, following minor alterations/clarifications, it was accepted as an ISO/IEC international standard (ISO/IEC 14772-1) in March 1997 and renamed to VRML97 (VRML Consortium, 1997a).

There are significant differences between VRML 1.0 and VRML 2.0 as mentioned above. VRML 1.0 browser plug-ins cannot display VRML 2.0 scenes but most VRML 2.0 browser plug-ins can display VRML 1.0 scenes.

VRML is a fairly new technology in its infant stage and is still evolving to meet the needs of the larger Internet community. Although the language has progressed to be useful in various application areas, certain extensions to the language would benefit an even larger number of users. Such extensions that are currently under investigation are database integration conventions, world scalability, security, closer integration with HTML and DHTML, and possibly a compressed binary file format (Parisi, 1997, p.40-41). Streaming of VRML worlds is also being looked at because elaborate VRML worlds tend to become large (100KB+) causing unacceptable increases in downloading times.

3.3. VRML Setup

In order to view VRML files it is necessary to first setup the PC. The setup that was used for this project will now be outlined. A number of other setup configurations could also have been used. The following software was installed on the PC:

- Windows NT 4 operating system;
- Microsoft Internet Explorer 5 (MSIE5);
- Java Virtual Machine (JVM) that ships with MSIE5; and
- VRML plug-in that ships with MSIE4.

VRML files are viewed inside a Web browser, such as MS Internet Explorer (MSIE) or Netscape Navigator that must be installed on a PC or workstation. MSIE5 was used during the development of this project. The reason for this for using MSIE5 is because it is stable and has it own JVM and VRML plug-in.

The Java Virtual Machine (JVM) and its associated software must also be installed on the PC or workstation in order to run Java applets in the Web browser. The JVM that ships with MSIE 5 was used during the development of the project. The installation of the JVM is part of the MSIE5 installation and no other special software is required.

The Web browsers must have a VRML plug-in installed. There are many VRML plug-ins currently available that allows one to view VRML 2/97 files. Among these are CosmoPlayer from SGI, WorldView from InterVista, CyberPassage from Sony, Liquid Reality from DimensionX and Cortona from Parallel Graphics. Most of these still only support VRML 1. The plug-in that ships with MSIE 4.01 (WorldView) was used during the development of this project. The updated plug-in that ships with MSIE 5 could not be used as it did not handle VRML node event processing in the same way as the old version. To install the plug-in the vrml2c.exe file must be run. This is a self-extracting file that also installs the VRML plug-in for MSIE. After this procedure any VRML file can be viewed in the Web browser.

Although the VRML specification was used to create the above plug-ins, there are still different interpretations or implementation of the specification among them. A

VRML file created and viewed using a certain Web browser/plug-in combination may not look the same using another combination.

VRML files are text files (as will be seen in the next section) with a ".wrl" extension e.g. filename.wrl. Because VRML files tend to become large for complex worlds, they are sometimes compressed having a ".wrz" or ".wrl.gz" extension. A significant saving of Internet download time can be achieved when file compression is used. A .wrl VRML file can be zipped (using WinZip or Gzip utilities present on most systems) and then given a ".wrz" or ".wrl.gz" extension.

3.4. VRML Structure

Before VRML came into being there were other methods of creating 3D virtual environments using programs such as WorldToolKit from Sense8, AutoDesk CDK, VREAM and others. Each one of these packages used a different file format for storing the virtual environment and no standard file format existed. The aim of the creators of VRML was not to create a new file format, but to establish a standard file format that could be used as a "universal interchange format for integrated 3D graphics and multimedia" (VRML Consortium, 1997a), also referred to as an "open standard" (Roehl et al., 1997, p.5). It was also designed with the Internet in mind, as well as for use on local area networks (LAN) and local machines. It can therefore be said that VRML is centred on a file format (based on SGI's OpenInventor format).

A typical VRML file consists of two main sections, namely the header and one or more nodes. Figure 3.1 is a simple VRML file that shows these two sections.

```
#VRML v2.0 utf8

Cone
{
  bottomRadius 1.0
  height 2.34
  bottom FALSE
}
```

Figure 3.1 A simple VRML file

3.4.1. File header

The file header is the first line of the code. It starts with a hash ('#'), which normally denotes the start of a comment except if it's the 1st line of the file. It is followed by the string "VRML" indicating the type of file, "v2.0" indicating the VRML version (because version 1 and 2 has such different implementations) and "utf8" indicating what type of character set is being used .

In VRML 1 the standard ASCII text character set were used whereas in VRML97 the character set is UTF-8. UTF-8 is an acronym for Universal Multiple-Octet Character Set Transformation Format and is part of an internationalisation of computing systems. This transformation format was defined by the ISO because many writing systems still use the 8-bit US-ASCII code rather than 16-bit Unicode (Yergeau, 1996). ASCII characters use only one byte of the UTF-8 character. The advantage of using UTF-8 encoding lies in the fact that when typing non-ASCII characters, e.g. for a foreign language such as Danish, the VRML scene author can use Danish characters in the VRML file, except for the field and node names (Lemay, 1996, p.217; Roehl et al., 1997, p.11).

The two basic data structures defined in VRML are nodes and fields. Nodes are analogous to Java or C++ classes and a field is similar to a variable defined within a class (Roehl et al., 1997, p.9).

3.4.2. Nodes

A node is the most basic standalone structure in VRML. It contains one or more fields where each of these fields defines a property of the node. In Figure 3.1, a Cone node is used in which the properties of 3 fields are set. A node may have many fields defined in it but if a node is used without setting a field property, the default value for that field is assumed.

There are 54 different nodes in the VRML97 specification and these are split into 9 groups: grouping nodes, special groups, common nodes, sensors, geometry, geometric properties, appearance, interpolators and bindable nodes.

3.4.3. Fields

Fields cannot be used in a standalone manner like a node but can only be used inside a node. Fields are specified with one of four different access types. These access types can be split into two groupings, namely ordinary fields and fields that can send or receive events ("event" fields). Figure 3.2 shows these groupings and access type relationships.

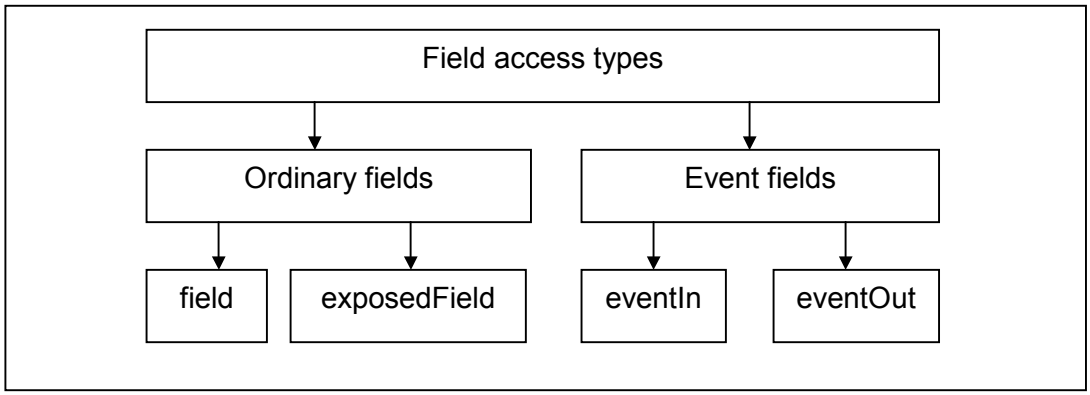


Figure 3.2 Field access type relationships

Ordinary fields can store a value that may be set or read. Event fields cannot store a value and are used to pass information from one node to another using events (this will be covered later). An exposedField is similar to a public class variable and a field is similar to a private class variable in terms of accessibility of that variable.

A variable in a programming language like Java not only indicates the access type but also the type of variable, e.g. integer or floating point variable. In VRML there are 11 different field types. Each field type (with two exceptions) may be a single variable (these types have the SF prefix) or it may be a multi-variable field (these types have the MF prefix). Multi-variable fields act like arrays containing any number of single variables. Figure 3.3 shows the MovieTexture node definition from the VRML97 specification.

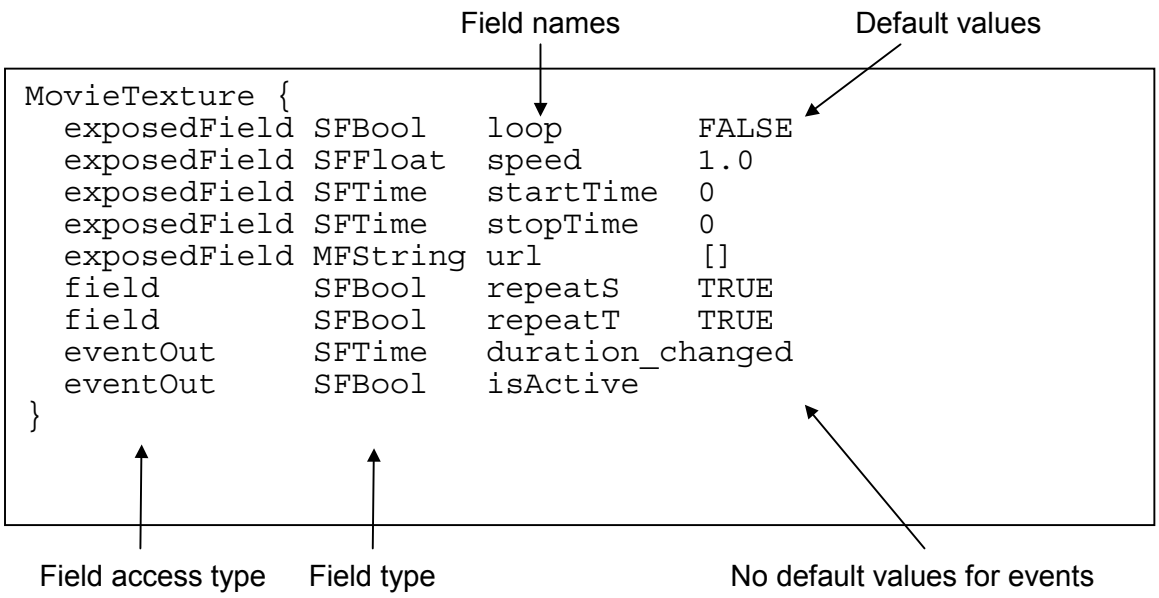


Figure 3.3 MovieTexture node definition

From Figure 3.3 it can be seen that the node contains a multi-variable field called url that is an array of Strings with a default value of [] (an empty array). The node also contains two single variable fields of type Boolean and default values of TRUE. The event field, duration_changed, is used to pass a single variable of type Time as an input event to another node.

3.4.4. Scene structure

Nodes in a VRML scene can be structured in one of two methods. Nodes can be placed in the scene one by one or related nodes can be grouped together using group nodes. The latter will make it easier to manipulate the “grouped object” as a single object. The nodes can be grouped together in one group or in a tree-like hierarchical structure. When a tree structure is used, a parent-child relationship is created where the upper-level node is the parent and the lower-level node(s) are the children. The properties of the parent node affect the properties of the children

(Roehl et al., 1997, p.12). The two methods of structuring can best be contrasted using an example of a human arm created in VRML.

Nodes containing the different parts of the arm (i.e. the fingers, the hand, the lower arm, the upper arm and shoulder) can be placed in the file in an ungrouped manner (i.e. non-hierarchical structure). When the shoulder is moved, the rest of the arm is expected to move as well to stay in the same relationship with the shoulder. However, because of the non-hierarchical structure, this is not the case. The shoulder will move but the rest of the arm will remain in the position it was previously.

On the other hand, when using a hierarchical structure, if an object is rotated, all the lower-level nodes attached to this node will move in relation to it while all the upper-level nodes remain intact. In other words when the hand is moved, the fingers attached to the hand node will move in relation to the hand. The logical structure of a VRML scene is called a scene graph (Roehl et al., 1997, p.9). The scene graph of the above example is shown in Figure 3.4.

3.4.5. Coordinate systems

Each VRML file defines its own 3D, global, right-handed coordinate system when it is loaded into the Web browser. This means that each node that creates some object in the scene, does so with respect to the global coordinate system. The Transform node is only node that can be used to translate or rotate its child nodes. So each time a Transform node is created, it creates its own local coordinate system that is relative to the parent's coordinate system and affects all the child nodes. The

SHOULDER node in Figure 3.4 has its own local coordinate system relative to the ARM's coordinate system (which is the global coordinate system if the ARM is defined within the file). The HAND's coordinate system is again relative to the LOWER ARM's coordinate system, and so forth.

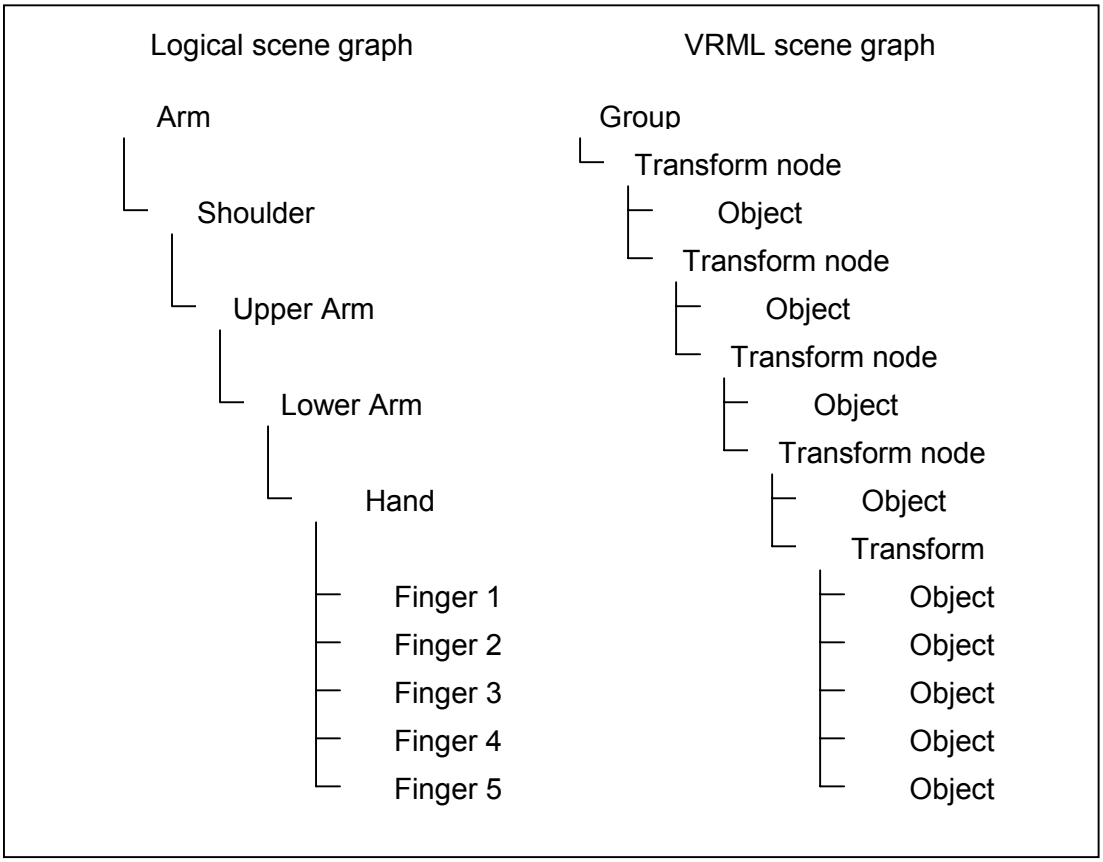


Figure 3.4 Scene graph of human arm hierarchy (adapted from Roehl et al., 1997, p.13)

3.4.6. Events and event routing

To effectively pass information from one node to another, these nodes need to be identified or named. VRML provides a unique way to name individual nodes. This is done using the DEF keyword. A node name given using the DEF keyword has file scope. Figure 3.5 shows the same node defined in Figure 3.1 but with the node that has been named MyCone.

```
#VRML v2.0 utf8

DEF MyCone Cone {
  bottomRadius 1.0
  height 2.34
  bottom FALSE
}
```

Figure 3.5 Using DEF to name a node

There is another advantage in naming a node. Copies of a named node can be made by simply using the keyword USE together with the name of the node that must be used. This enables a scene author to reuse code that has been generated earlier in a file. Figure 3.6 shows an example of this principle. In this example two cones having exactly the same properties are created next to each other.

```
#VRML v2.0 utf8

Cone {
  bottomRadius 1.0
  height 2.34
  bottom FALSE
}

Transform {
  translation 3 0 0
  children Shape{
    geometry USE MyCone
  }
}
```

Figure 3.6 Using USE to reuse code

Once a node is named, it can be used to generate events. Events are analogous to function calls in a programming language but differs in the fact that there is no stack (push & pop). It is used to encapsulate data sent that must be sent from one node to another (Roehl et al., 1997, p.13). Each event contains two pieces of information namely the data and a timestamp (the time the event was generated).

The method of actually connecting events generated by one node to a node waiting to receive input from this event is termed routing events. Ordinary field cannot be used with events. Only eventIn, eventOut and exposedField access type fields can be used to either generate or receive an event. The exposedField type can act as an eventIn or eventOut depending on how it is implemented. This means that the eventOut field of one node is linked or routed to the eventIn (or exposedField) of another node. Routes may be created using the ROUTE and TO keywords with the syntax:

```
ROUTE OutNode.eventOutName TO InNode.eventInName
```

For example:

```
ROUTE MyCone.Color_changed TO MySphere.set_Color
```

Only the same field type may be connected with ROUTEs. For example when the eventOut is of type SFInt32, the eventIn type must also be of this type and cannot be of the MFInt32 type (the one is a single variable and the other an array although both are of the same type Int32). ROUTEs must be placed after the nodes used in the ROUTE have been named, therefore they are normally placed at the end of a VRML file.

One eventOut can be routed to many different eventIns (this is also known as fanout). The opposite is also true that more than one eventOut may be routed to one eventIn (this is known as fan in). VRML uses a specific execution model to route these fields together but does not guarantee the order of execution when events occur at the same time.

Events are generated by the browser except for scripting nodes where events can be generated explicitly (Roehl et al., 1997, p.21). This also shows the two methods of creating animation/interaction in VRML, i.e. by making use of the built-in event-generators or by making use of scripting (scripting will be covered later).

If one looks at the built-in nodes that VRML provides, there a 7 sensor nodes that can be used, among others, to generate events based on user interaction (clicking on an object or moving around in the scene), time and whether an object is visible to the user or not. One of these, the TouchSensor, can be placed together with other nodes inside a grouping node that will make the other nodes' geometry touch-sensitive. This means that once an object in the scene that is connected with the sensor is clicked, it will generate an event. Figure 3.7 shows an example of using event routing where a light is switched on while the mouse button is being pressed over the sphere.

Interpolators are also very important built-in VRML nodes for creating predefined animations. There are 6 different kinds of interpolator nodes. Interpolators are used to change object properties (colour, position, orientation, etc.) based on a time input (derived from a TimeSensor node). For example a PositionInterpolator can be used to move object(s) in a predefined path. Figure 3.8 shows an example of a cylinder moving on a square path and completes one lap every 10 seconds. The TimeSensor is used to generate events for the interpolator, which in turn generates events for the cylinder that will translate its position according to the interpolator.

```
#VRML v2.0 utf8

Transform {
  children [
    DEF BALL Shape{
      geometry Sphere {radius 2.5}
    }
    DEF TOUCH TouchSensor {}
  ]
}

DEF LIGHT DirectionalLight {
  direction 1 0 -1
  on FALSE
}

ROUTE TOUCH.isActive TO LIGHT.on
```

Figure 3.7 Simple event routing example

```
#VRML v2.0 utf8

DEF DRUM Transform {
  Translation 0 0 0
  children
   Shape{
     geometry Cylinder {radius 1.5 height 2.5}
   }
}

DEF TIME TimeSensor
{ cycleTime 10 }

DEF PI PositionInterpolator {
  key [0 .25 .5 .75 1]
  keyValue [0 0 0, 1 0 -1, 0 0 -2, -1 0 -1, 0 0 0]
}

ROUTE TIME.fraction_changed TO PI.set_fraction
ROUTE PI.value_changed TO DRUM.set_translation
```

Figure 3.8 Simple animation using a position interpolator

3.5. Extending VRML

The built-in event generator mechanism enables one to create basic animations and interaction capabilities. The scripting node can be used to create extremely complex animations, behaviours, objects and scenes. This section will introduce different methods to extend the capabilities of VRML, a few of which rely heavily on scripting nodes.

3.5.1. Prototypes

VRML enables a scene author with a natural method of extending the language by creating new nodes. These new nodes are called prototypes and once they are defined, act in exactly the same way as any other built-in node. A prototype is analogous to a Java or C++ class. The prototype definition (similar to the one in Figure 3.3) acts as a template of which copies may be instanced. In other words, there are no instances of the node until it is used in the VRML file.

These prototypes (often just called protos) can contain fields and generate or receive events just like other nodes. The difference is that these must be user-defined in the prototype definition. Protos are defined using the PROTO keyword. The rest of the definition is made up of the name of the proto (i.e. the new node name), a list of the fields and events that the proto contains (in [] brackets) and the implementation of the proto (in {} brackets). The first part of the definition, excluding the implementation, is similar to a C++ class declaration (class interface) while the implementation is similar to the C++ class definition.

New node name

List of fields and events

```
#VRML v2.0 utf8

PROTO PineTree [
exposedField SFColor trunkColor .5 .25 0
exposedField SFColor leaveColor 0 .8 0
]
{
  Transform {
    Children Shape {
      geometry Cylinder {height 10 radius 2}
      appearance Material {diffuseColor IS trunkColor}
  }
  Transform {
    translation 0 2 0
    Children Shape {
      geometry Cone {height 6 bottomRadius 4}
      appearance Material {diffuseColor IS leaveColor}
  }
  Transform {
    translation 0 5 0
    Children Shape {
      geometry Cone {height 6 bottomRadius 4}
      appearance Material {diffuseColor IS leaveColor}
  }
}

PineTree {
  trunkColor .9 .9 0
  leaveColor .2 .8 .2
}
```

Node implementation

Instancing a new Tree node

Implicit routing

Figure 3.9 Defining and using a prototype

Figure 3.9 defines a new PineTree node that can be used to create a pine tree that is 13 meters high of which the trunk and leave colours can be set (the default colours being brown and green respectively). An instance of the node is created and different color values are specified. The IS keyword is used to implicitly route the proto fields to multiple internal nodes or fields. So whatever field values are specified when instancing a node, these will get routed to the appropriate internal implementation (this is also true for events).

It is important to note that the implementation of the proto is totally hidden from the rest of the VRML file. Therefore names DEFined in the proto have only scope between the curly brackets of the proto implementation (Roehl et al., 1997, p.21).

Protos may contain scripting nodes as well as any other event routing. This is also internal to the proto and this is one of the most appealing aspects of using protos in a VRML file. Complete behaviours can be encapsulated inside a proto and used any number of times in the scene (Marrin et al., 1997, p.2).

VRML allows a scene author access to protos defined in other files. This is called external prototypes and is done using the EXTERNPROTO keyword instead of the PROTO keyword in the following manner:

```
EXTERNPROTO PineTree [
exposedField SFColor trunkColor
exposedField SFColor leaveColor
]
"www.InetAddress.com/directory/filename.wrl#PineTree"
```

It can be seen that the external proto interface is exactly the same as the original proto interface with the difference that one or more URLs (or URNs) are placed after the externproto interface. This tells the VRML browser where to find the proto implementation. When a list of URLs are specified, the browser will search in preceding order for the first URL containing "interpretable data" (VRML Consortium, 1997a). The "#PineTree" string at the end of the URL specifies the original proto name. This enables the external proto to have a different name than the original proto. If no proto name is specified, the first proto in the file "filename.wrl" is used.

External protos provide a mechanism whereby libraries of commonly used objects can be created and then be accessed from other dedicated VRML files. This has the advantage that a single piece of code is reused and has the added benefit of reducing download times because the common library only needs to be downloaded once.

Marrin et al. gives a good example of a behaviour that was created using a prototype (1997, p.2). He developed a snake prototype where the snake contains 15 segments that start to wiggle when the user gets closer than a predefined distance (internally using ProximitySensors). When more than one snake is placed in the scene, these can be set to wiggle out of phase with each other based on a single field setting.

Prototyping provides many advantages that may be summarised as follows:

- They can be used in exactly the same way as built-in nodes.
- There is no need to create an instance of a node before you can make copies of that node (such as when DEF / USE is used).
- Libraries of commonly used nodes can be created using external protos.

3.5.2. Scripting

Programming languages can be used to add behaviours and interaction capabilities to a VRML scene that can be far more complex than built-in animation techniques allow. The mechanism provided by VRML is the Script node. VRML 1.0 did not support scripting. VRML 2.0/97 supported Java and a core subset of JavaScript called VRMLScript at the time of writing the VRML 2.0 specification. This core

subset was later declared a standard and the named was changed to ECMAScript (ECMA-262). Therefore when VRML97 was standardised it supported language bindings for Java and ECMAScript. The Script node's function and capabilities will now be discussed showing the use of each of these language extensions as well as where they are applicable.

3.5.2.1. The Script node

The script node is similar to other nodes in the sense that it also contains fields, can be given a name, can generate and receive events and its event routing is done in the same way. The difference is that it may define its own set of fields (excluding exposedField access types) and provides programmable control of the scene. The script node contains 3 basic fields as shown in the node definition in Figure 3.10.

```
Script {
  exposedField MFString url            []
  field        SFBool   directOutput   FALSE
  field        SFBool   mustEvaluate   FALSE
  # And any number of:
  eventIn      eventType eventName
  field        fieldType fieldName initialValue
  eventOut     eventType eventName
}
```

Figure 3.10 Script node definition

The mustEvaluate field, when set to TRUE, tells the browser that eventOuts generated by the script must be evaluated immediately after they occur. The directOutput field, when set to TRUE, enables the script to directly set eventIns of other nodes so that they don't need to be routed using the ROUTE statements. The url field contains the actual inlined programming code (or a reference to a file

containing the code), which is also called the script. It can be seen that this is a multi-variable field and it may contain a comma-separated list of alternatives.

Defining fields in the node is similar to defining fields in prototypes and has the generic form as shown in the last three lines of Figure 3.10. Once these are defined, they may be used inside the script similar to prototypes. A typical example of a script node is shown in Figure 3.11. Four new fields are defined namely a Boolean field named LightOn with an initial value of FALSE, a node named myLight2 with an initial value of Light2 (a node that is DEFined in the VRML file), an eventIn named itHappened and an eventOut named MotorSwitch. The url contains three alternative script options in different languages in order of preference. If the first alternative cannot be loaded, the browser tries to load the second and subsequent alternatives until an interpretable script is loaded. These URLs may also specify the same file that can be loaded from different locations.

```
DEF LightSwitch Script {
  directOutput  TRUE
  field         SFBool LightOn      FALSE
  field         SFNode myLight2 USE Light2
  eventIn       SFTime itHappened
  eventOut      SFBool MotorSwitch
  url ["lightsw.class",  #1 Java version
       "lightsw.js",     #2 JavaScript file version
       "javascript: ..."] #3 inline ECMAScript version
}
```

Figure 3.11 Basic script node example

All language bindings are forced to provide an initialise function and a shutdown function. These two functions are analogous to a constructor and destructor in C++. Code that is placed in them will automatically be executed once the scene had been

completely loaded before any events have been generated (`initialise()`) or just before the script is removed from the scene (`shutdown()`) (Roehl et al., 1997, pp.49-50).

A static browser class interface is provided in all language bindings that provide the script author the ability to communicate with the browser (Roehl et al., 1997, p.53). The interface provides functions to get the current URL, frame rate, version, and other useful information. More importantly, it provides the ability to dynamically load or create new objects and ROUTEs on the fly. For example, when an object is clicked in the scene, the script behind the click may create a new object and place it in the scene dynamically. Four functions perform this task, namely createVRMLFromURL, createVRMLFromString, loadURL and replaceWorld. The function names are self-explanatory and an example of this will be given in the ECMAScript section.

Each language binding differs in the way it handles events (specifically eventIns). The similarity among all events, however, is that they all contain two pieces of information, i.e. the data and a timestamp.

3.5.2.2. ECMAScript scripting

ECMAScript and JavaScript are often used interchangeably because the latter stems from the former. In a script, JavaScript can be used by either using it inline or by creating a reference to a JavaScript file (having a .js extension). There is fundamentally no difference in the execution. Figure 3.11 shows that the inlined

version starts with a header in the form of “javascript:” and this is followed by the actual code.

Each eventIn gets associated with a function with the same name and the function takes on one or two parameters (the data and/or the timestamp). Fields are used in the same way that variables are normally accessed, i.e. by assignment. EventOuts are generated by assigning a value to an eventOut. Figure 3.12 gives an example of using inlined JavaScript code.

```
#VRML v2.0 utf8

Transform {
  children [
    DEF BALL Shape{
      geometry Sphere {radius 2.5}
    }
    DEF TOUCH TouchSensor {}
  ]
}

DEF PlaceBox Transform { translation 2 2 0 }

DEF CreateBox Script {
  eventIn SFBool itHappened
  field SFInt32 BoxCnt 0
  field MFNode theBox
  eventOut MFNode child
  url “javascript:
    function itHappened(val, time) {
      if(val == 0) { #left button
        theBox = Browser.createVRMLFromString
                  (“Shape {geometry Box {2 2 2}}”);
        child = theBox;
        BoxCnt++;
      }
    }”
}

ROUTE TOUCH.isActive TO CreateBox.itHappened
ROUTE CreateBox.child TO PlaceBox.add_Children
```

Figure 3.12 Example of using inlined JavaScript

The above example creates a new Box object when the left mouse button is clicked over the Sphere object. The new Box is placed inside the PlaceBox transform as seen from the ROUTE statements. If a JavaScript file is used instead of inlining, the file will contain exactly the same code as that contained between the inverted commas in the url field (but without the “javascript: ” header).

3.5.2.3. Java Scripting

Java is the second programming language that can be used in Script nodes. This is sometimes called Java in Script nodes Authoring Interface (JSAI) (Roskothen, 1997, p.48). Java is a full programming language and offers everything JavaScript offers and in addition it also provides networking, multithreading and user interface capabilities.

The only other major difference between Java and JavaScript, with respect to VRML scripting, is the way eventIns are processed (Roehl et al., 1997, p.54). To this extent Java provides two functions, namely processEvent and processEvents. The former handles only one event at a time while the latter may handle more than one event.

As shown in Figure 3.11, when Java scripting is used a reference to a Java class file is made. Java code cannot be inlined because it needs to be compiled (to bytecode) first. The Java API does not provide class libraries that support VRML. The VRML class libraries are loaded with the VRML plug-in and contains three basic scripting packages namely vrml, vrml.field and vrml.node (Roehl et al., 1997, p.55). These packages contain general VRML classes and node and field definition classes.

These classes are used in Java applets/applications to enable the script author to communicate with the scene.

The Java class file contains a class that subclasses the Script class. The main purpose of the class is to do the event handling of the script, which is similar to event handling for the Java AWT (abstract windowing toolkit). It may also contain or call other classes to perform other tasks, e.g. networking. An example of such a class that implements a light switch (together with the VRML file) is shown in Figure 3.13 and Figure 3.14.

```
#VRML v2.0 utf8

Transform {
  children [
    DEF BALL Shape{
      geometry Sphere {radius 2.5}
    }
    DEF TOUCH TouchSensor {}
  ]
}

DEF LIGHT DirectionalLight {
  direction 1 0 -1
  on FALSE
}

DEF SWITCH Script {
  eventIn SFBool itHappened
  eventOut SFBool output
  url "theSwitch.class"  #Reference class file
}

ROUTE TOUCH.isActive TO SWITCH.itHappened
ROUTE SWITCH.output TO LIGHT.on
```

Figure 3.13 An example of using Java scripting (VRML file)

```
import vrml.*;      //Import VRML packages
import vrml.node.*;
import vrml.field.*;

public class theSwitch extends Script {
  private SFBool lightOn;
  private boolean isLightOn = false;

  public void initialise() {  //Initialization code
    lightOn = (SFBool)getEventOut("output");
  }

  //Handle the event: toggle output
  public void processEvent(Event e) {
    if(e.getName().equals("itHappened")
      if(e.getValue()==false) //left button pressed
        if(isLightOn==true) {
          isLightOn=false;
          lightOn.setValue(false);
        }
        else {
          isLightOn=true;
          lightOn.setValue(true);
        }  }}
```

Figure 3.14 An example of using Java scripting (Java file)

It can be seen from the example that all the access to the fields in the script is done via get and set functions provided by the Java packages. The field types are actually classes defined in the VRML packages. EventOuts must be captured before it can be assigned some value that will generate an event.

3.5.2.4. Comparison between JavaScript and Java

There are a lot of differences between Java and JavaScript and the choice of which one to use for an application depends on the specific application. These two languages will now be compared with respect to their use in Script nodes.

JavaScript is an interpreted language and does not need to be compiled. Therefore no compiler is necessary and code can be inlined in the VRML file. It does not run as fast as Java does, however, because it gets interpreted when executing. JavaScript is object-based and it does not have strong type checking that may lead to unforeseen errors that are difficult to debug. It uses dynamic binding for reference checking (Buchanan, 1997, p.418). It handles events very comfortably. It is not possible to create network connections or threads with JavaScript. JavaScript can be used for generating quick code that need not be complex. JavaScript does not require any environment variables (Roskothen, 1997, p.48).

Java files needs to be compiled to bytecode and executes on a JVM. Java bytecode runs faster than JavaScript code. Java has strong type checking and is fully object-oriented (Netscape, 1997). It uses static binding for reference checking. There are more overheads when Java is used but it is possible to create large elaborate classes and to include network and threading capabilities. Java requires a CLASSPATH environment variable. One similarity is that both Java and JavaScript are platform independent.

3.5.3. The External Authoring Interface (EAI)

3.5.3.1. Introduction

During the time that the VRML 2.0 specification was being written the need arose for an interface that could permit communication between VRML and an external environment. After the final VRML 2.0 specification was completed on the 4th of

August 1996, Chris Marrin from SGI submitted a proposal for a Java based external interface for VRML (EAI-WG, 1998). In December 1997 a working group was formed to formalise the proposal and include a language neutral implementation. In January 1999 the EAI specification was given to the VRB (VRML Review Board) for ISO formalisation (VRML Consortium, 1999). The EAI specification, once accepted, will form the second part of the VRML97 specification. At this stage it is still only a proposed Informative Annex to the VRML97 specification (Finlayson, 1999).

The draft specification defines the EAI as "the interface that applications external to the VRML browser may use to access and manipulate the objects defined in ISO/IEC 14772-1" (VRML Consortium, 1999). This means that another environment external to the VRML browser may gain access to nodes in a VRML scene through a predefined interface while still using the existing event model. The specification aims to generalise this 'environment' so that objects in the VRML browser may be accessed equally well from a Java applet as from a custom application written in another language that is using this interface.

The specification states four methods of accessing this scene information (VRML Consortium, 1999):

- Accessing the functionality of the Browser Script Interface.
- Sending events to eventIns of any defined nodes inside the scene.
- Reading the last value sent from eventOuts of any defined nodes inside the scene.
- Getting notified when events change values of any defined node fields inside the scene.

The Browser Script Interface defined in the EAI specification adds functionality to the interface that is used by JSAI. It allows direct access of named nodes and their fields inside the VRML file and the ability to get real-time notification of registered events.

A Java language binding for the EAI is included in the specification. Most of the major browsers support the EAI although it is not strictly necessary in order for them to be VRML97 compliant as it is not yet officially part of the standard.

In order to get a better understanding of the usefulness of EAI as well as where it fits into the bigger picture the EAI and JSAI will now be compared.

3.5.3.2. EAI vs. JSAI

A VRML scene using JSAI contains a reference to the Java class file (see Figure 3.13) while a scene controlled by the EAI has no reference to the class controlling it. Instead the class, which is part of an applet, is linked to the VRML file and creates references to known named nodes in the VRML scene.

Java class files imported 3 VRML-specific packages namely vrml, vrml.field and vrml.node while a class that wants to use the EAI needs to import EAI-specific packages namely vrml.external, vrml.external.field and vrml.external.exception.

A Java class called from a Script node cannot have access to other named nodes in the scene except if these have specifically been defined inside the Script node. The

EAI enables one to access any named node, its fields (excluding fields of type field) and events in the scene (except nodes DEFined in a prototype).

Both eventIns and eventOuts are handled differently. The JSAI provides 2 functions that can be called when the Script node receives an eventIn. An eventOut is generated by getting a reference to the eventOut field and assigning the value to it. The EAI enables one to directly read an eventOut from any node (including a Script node) or to use a call-back mechanism where a class may register to listen to a specific eventOut (from any node). When an eventOut is generated by the node, a function in this class is called that may execute event-handling code. Data can also be written directly to any eventIn.

The Java class called using JSAI need not use the Browser interface to access fields from the Script that it is called from. When using EAI all access is done through the Browser interface. This means that a reference must first be created to nodes that need to be accessed. Once a reference has been obtained, fields inside the node may be accessed.

It can be seen that although the Java language binding is used for both the EAI and JSAI, the two methods differ in a number of respects. The demands set by the application will ultimately determine which of the two methods will be used.

3.5.3.3. EAI implementation

Although the purpose of the EAI is to interface any external environment to a VRML scene, only a Java language binding have been defined up to date. This means that

a VRML scene may be accessed by a Java applet when they are both embedded on the same HTML Web page. An example of a HTML page (code) containing a VRML file reference as well as a Java applet to control the VRML scene is shown in Figure 3.15.

From Figure 3.15 it can be seen that the semantic “mayscript” is used after the Java applet reference. This is the glue that connects the Java applet to the VRML scene. Finlayson, when referring to this semantic, states that it is used “so the applet can get the VRML plug-in browser object instance from the browser” (1999).

```
<html> <body>

<embed src="VRMLfile.wrl">
<applet code="EAIclass.class" mayscript>

</body> </html>
```

Figure 3.15 HTML code linking VRML and Java

Once the HTML page and the VRML files are created, the Java applet class can be written. The main entry point of accessing the scene from the applet is by acquiring a reference to the instance of the VRML browser using the getBrowser function. After this, a reference to any named node can be acquired using the getNode function. From the node reference, a reference to an eventIn or eventOut can be obtained using the getEventIn and getEventOut functions. This will allow scene data to be read or changed as needed.

As mentioned earlier, the EAI allows one to read eventOuts directly or to use a call-back mechanism that calls a function once a registered eventOut occurs. An

example of reading an eventOut directly and changing scene data is shown in Figure 3.16.

```
import vrml.external.*;          //Import EAI packages
import vrml.external.exception.*;
import vrml.external.field.*;

public class myApplet extends Applet {
  Browser browser = null;

  public void start() {  //Initialization code
    browser = Browser.getBrowser(this);
    Node myNode = browser.getNode("NamedNodeFromScene");
    EventOutSFVec3f myPos =
      (EventOutSFVec3f)myNode.getEventOut("translation");

    float pos[] = myPos.getValue();
    //do something with position data
  }
}
```

Figure 3.16 Reading an eventOut

Figure 3.16 clearly shows the steps involved in reading an eventOut from the scene. First a browser object is obtained, and then getNode is used to get a reference to a Transform node in the scene named NamedNodeFromScene. This reference is then used to get the translation field from the node. The getEvent methods return an Event object that must be cast to the correct Event object. The variable, myPos, will thus contain an array with 3 floating point values. The actual position vector data can be retrieved from the variable using the getValue function. Normally when data is read from or written to fields in this way, get and set functions are used.

The above example shows how data can be read from a field. In the same way data can be written to a field. If a field is an exposedField, data can be written to or read

from the same field. But in this case an eventIn and an eventOut variable should be defined although it is the same field. This is because the EAI packages only provide the two functions getEventIn and getEventOut. Figure 3.17 shows a variation of the previous example where data is written to a node's field. It retrieves a translation field from a Transform node for writing and then changes the translation vector data in the field by using the setValue function.

```
import vrml.external.*;          //Import EAI packages
import vrml.external.exception.*;
import vrml.external.field.*;

public class myApplet extends Applet {
  Browser browser = null;

  public void start() {  //Initialization code
    browser = Browser.getBrowser(this);
    Node myNode = browser.getNode("NamedNodeFromScene");
    EventInSFVec3f myPos =
      (EventInSFVec3f)myNode.getEventIn("translation");
    myPos.setValue(new SFVec3f(1,2,3));
  }
}
```

Figure 3.17 Writing to an eventIn

The second method of receiving an eventIn is by using a callback mechanism. This is useful when the applet needs to wait for an event to happen before it can execute a section of code. There are a few steps involved in setting up this mechanism. The EAI packages provide an EventOutObserver interface that is used to implement this mechanism.

First a class that contains the event-handling code must be created. It implements the above interface and contains a callback function implementation. The callback function takes as parameters the eventOut that it was registered to, a timestamp and a user-defined class that may contain information required to handle the event. An

instance of this class must then be created in the applet code and a reference to the eventOut that must be monitored must be obtained. The advise method is then used to register the eventOut to the callback class. This first parameter of this function is the observer class and the second is the user-defined data class that the callback function receives. Every time the specific field generates an event, the callback function will execute. Multiple eventOuts may use the same callback function or different instances of the observer class may be created for each event. Figure 3.18 shows an example of this method of receiving an eventOut.

```
import vrml.external.*;          //Import EAI packages
import vrml.external.exception.*;
import vrml.external.field.*;

public class myApplet extends Applet {
  Browser browser = null;
  public void start() {  //Initialization code
    browser = Browser.getBrowser(this);
    Node myNode = browser.getNode("NamedNodeFromScene");
    EventOutSFBool Active =
      (EventOutSFBool)myNode.getEventOut("isActive");
    Listen ls = new Listen();
    Active.advise(ls,null);
  }
}

class Listen implements EventOutObserver {
  public void callback(EventOut e, double timestamp,
                       Object data) {
    //event-handling code goes here
  }
}
```

Figure 3.18 Using a callback mechanism to retrieve an eventOut

EAI can also be used to create VRML nodes and routes on-the-fly. This is done using the addRoute, deleteRoute, createVRMLFromString and createVRMLFromURL functions, provided by the Browser interface, in a similar way

as JSAI would do it. Once these nodes have been added to the scene, it is difficult to access them again except if a reference is kept in a hashtable or vector. This is because the nodes created dynamically have their own scope.

The EAI packages provide exception classes that must be used to catch exceptions thrown by some of the functions that have been used in the above examples (the exception handling code have been left out for clarity). For example, a getEventOut function may throw an InvalidEventOutException exception that must be caught and handled when a problem occurs in writing to a field. An example of exception handling is shown in Figure 3.19.

```
import vrml.external.*;          //Import EAI packages
import vrml.external.exception.*;
import vrml.external.field.*;

public class myApplet extends Applet {
  Browser browser = null;
  public void start() {  //Initialization code
    browser = Browser.getBrowser(this);
    try {
    Node myNode = browser.getNode("NamedNodeFromScene");}
    catch(InvalidNodeException ine) {/*handle exception*/}
    try {
    EventOutSFBool Active =
      (EventOutSFBool)myNode.getEventOut("isActive");}
    catch(InvalidEventOutException i) {
     /*handle exception*/}
  }
}
```

Figure 3.19 Exception handling example

The EAI has a number of very useful applications and differs from scripting and prototypes in a number of ways. The different methods covered in this section will be compared next and the comparison will include typical applications where each method will be most appropriate.

3.5.4. Conclusion

Previously the choice of which language and method to use to extend VRML was limited to what browser was going to be used to run the VRML file in. Nowadays all the major browsers support prototypes, JavaScript and Java scripting, and the EAI. Thus the choice of how to implement a VRML scene depends on the size and complexity of the required application.

Figure 3.20 Theoretical implementation of a VRML browser (Marrin, 1996)

Prototypes are the natural method of extending VRML to form new nodes. These nodes may or may not incorporate Script nodes that use Java or JavaScript scripting. Scripting is a method to include programming language support that can be used to create behaviors in a VRML scene. From this it can be seen that these two methods cannot really be compared at the same level as they extend VRML differently. Figure 3.20 shows a theoretical implementation of a VRML browser.

Figure 3.20 clearly shows the three methods discussed in this section. The EAI is completely outside of the VRML browser and controls the scene from an external

environment. On the other hand, the Script and Proto nodes are internal to the VRML browser.

Campbell states that the real difference between scripting used in Script nodes and the EAI is the fact that the former is VRML-centric while the latter is Java-centric (1997, p.44). Marrin et al. also notes that because Java is such a powerful language and the possibility is there to construct and control a complete VRML scene using the EAI, one might be tempted to use VRML simply as a 3D rendering engine (1997, p.1). This is not recommended for various reasons stated by Marrin et al. (1997, p.1-2) and because VRML has powerful built-in features. The idea is to control the VRML scene at a higher object-based level.

A better method of extending VRML is by using a combination of the methods discussed. This will give a VRML scene more flexibility (Campbell, 1997, p.44). A small VRML scene may contain only a few ROUTEs and require simple JavaScript scripting while a larger VRML scene may contain more ROUTEs and Java scripting. If reusable objects, which may include behaviours, are required, prototypes must be used. For any VRML scene that must be controlled externally, the EAI must be used but use must be made of scripting and prototypes because such a setup will utilise the best features of Java and the EAI.

3.6. Applications of VRML

3.6.1. Introduction

VRML as a 3D scene description language offers many advantages over 2D scenes and pictures. VRML enables the user to freely move in the 3D environment and view objects from any angle. A 2D picture of a factory can only be viewed in one way. A 3D virtual factory can be viewed from any angle. The user can walk around different objects, touch buttons and see how parts are being manufactured. As far as applications are concerned, the sky is the limit. Through the use of various built-in nodes in VRML the level of realism can also be increased dramatically.

VRML can be used to represent real and abstract (or virtual) systems in a 3D world (Beier, 1999). This is contrary to the notion that only abstract systems can be presented by VRML. In the following section examples will be shown of VRML worlds that have been created for various application areas.

3.6.2. VRML in entertainment

3D action games have received a lot of attention in the past few years with titles such as DOOM and Quake. These games contain a very large amount of graphics and this is the area where VRML can play an important role. This is especially true for multi user domain (MUD) type games that are played on the Internet. One example of a game written in VRML is Kosmodrom designed by Brian Washburn (http://world.std.com/~bsw/servodrom/index.html).

3.6.3. VRML in manufacturing

Various companies and individuals have realised the tremendous potential that VRML offers the manufacturing industry. VRML developments have been made in areas such as virtual prototyping systems, product evaluation systems, simulation of assembly sequences, and machine monitoring and control.

- Virtual lathe

Figure 3.21 shows a VRML example of a virtual lathe that was developed at the University of Michigan's Virtual Reality Laboratory (VRL). The user can cut a work piece at will by dragging a tool with the mouse. Where the tool tip and the work piece intersect, a cut is made in the work piece. The workpiece can be reset to start from scratch or the VRML code for the cut workpiece may be exported to a file. This data can then be used for prototyping purposes. The example also incorporates 3D spatial sound.

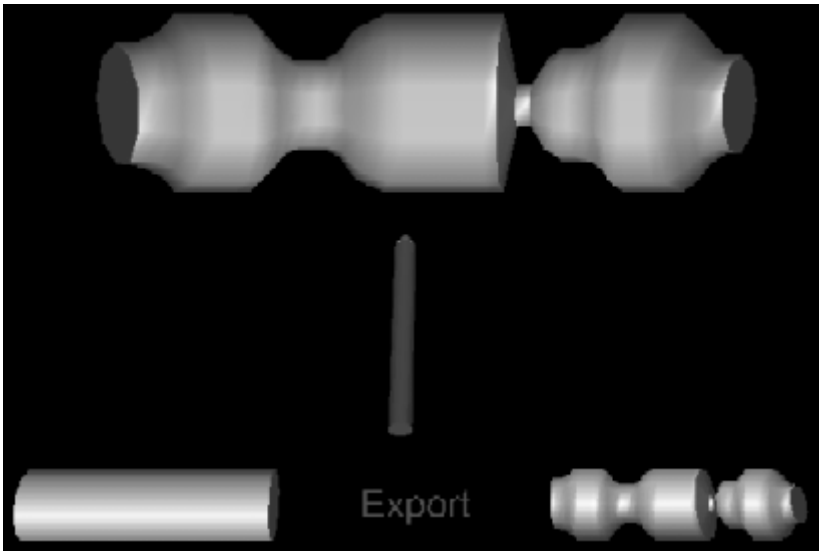


Figure 3.21 Virtual Lathe example

- Visual Interface to Manufacturing (VIM)

Ressler et al. explains a prototype system that can be used to access manufacturing data using different visual media including VRML (1997). VIM is a Web-based system containing data for manufacturing a Black & Decker mitre saw on an assembly line with a number of stations. A VRML model of the saw displays all its components and each component acts as an interface to a database. The database contains information on each part and also gives information as to where on the assembly line the part should be found, i.e. at what station. The system is intended to assist low-skilled labourers with detailed information of parts, how these parts are to be fitted together and at what station specific parts are to be found.

- Manufacturing system simulation and integration

Ressler et al. developed a system that integrates different VRML components into a manufacturing cell (1999). The system incorporates two industrial robots, a conveyor belt system and humanoid models. Figure 3.22 shows an example of the VRML scene. The user clicks a button to start the process. Robot 1 places some

widget inside a box that is on the table and robot 2 places the box on the conveyor belt. At the end of the conveyor belt the box falls onto a plate. A humanoid may walk next to the system when the user clicks on a button.

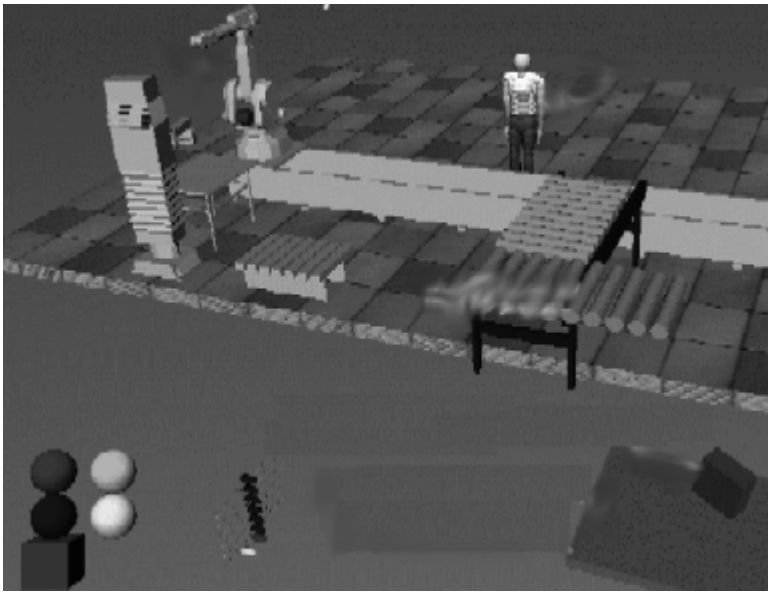


Figure 3.22 Manufacturing system simulation

Dauner et al. discusses a VRML-based product presentation system where various interior design companies display their products with a high degree of aesthetic quality (1998, p.57). Coors & Jung describes a system urban planning application that makes use of VRML to access data in a spatial database via the World Wide Web (1998, p.121).

3.6.4. VRML in architecture

One of the useful applications of VRML is providing online architectural structures and designs. Architects and real estate agents can put models of houses, apartments and buildings on the Internet together with other information regarding the structure. This can be done either for selling or renting it or for purposes of

online collaboration between architects or designers in different parts of the world. One example of a company that makes use of VRML for this purpose is Mischek Bau (http://www.mischek.at/).

VRML can also be used to show tourists what buildings or areas of interest look like without them physically having to travel there or see pictures of it. A good example of this is the Chapel of Gaurini in Italy (http://www.csi.it/cupola).

3.6.5. VRML in scientific/engineering visualisation

Bryson states that "scientific visualisation is the use of computer graphics to create visual images that aid in the understanding of complex numerical representations of scientific concepts or results" (1996, p.64). Djurcilov & Pang adds that as the Internet evolves and grows there is a greater demand to get access to scientific data in a more accepted format such as VRML (1998, p.7). This is also true for visualisation of complex mechanical and structural systems. Typical data sets that can be represented include computational fluid dynamics (CFD), molecular structure, stellar cartography, mathematical systems and graphs, magnetic fields and acoustics, to name a few. Figure 3.23 shows an example of using VRML for CFD representation (http://www-vrl.umich.edu/sel_prj/flow/). Figure 3.24 shows an example of a molecular structure representation in VRML (http://cssj.chem.sci.hiroshima-u.ac.jp/molda/).

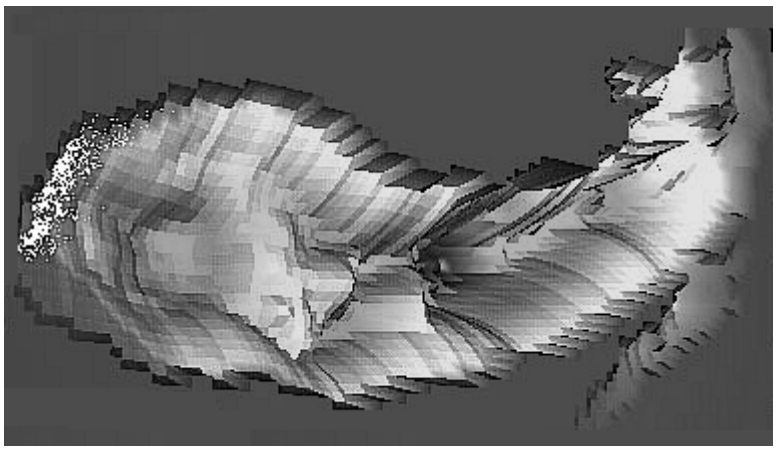


Figure 3.23 Using VRML for CFD representation

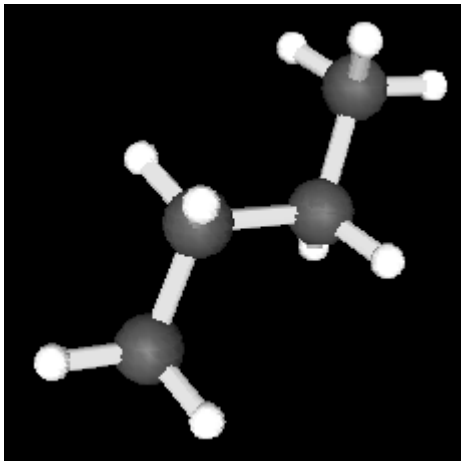


Figure 3.24 Using VRML for molecular structure representation

A typical mechanical system that can be displayed using VRML is gears and their operation. An example of differential gear operation using animated VRML is shown in Figure 3.25 (http://wizlab.com/vrml5.html).

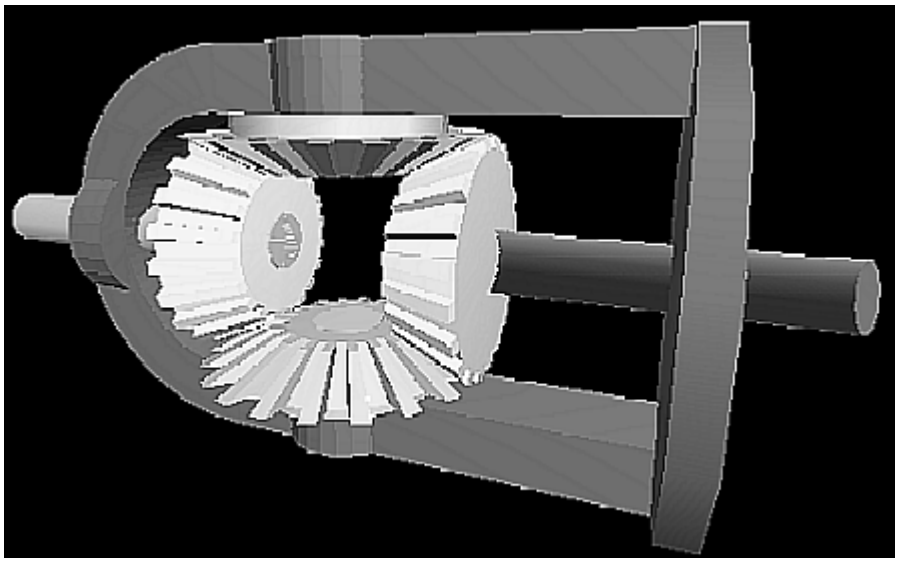


Figure 3.25 Using VRML to represent a differential gear system

3.6.6. VRML in education

As can be seen from the previous application areas especially scientific visualisation, that VRML can be an extremely useful tool for education and training purposes. VRML enables low-cost virtual reality type applications to be created for use anywhere on the Internet. VRML as a data visualisation and simulation tool has the potential to almost replace the traditional laboratory approaches in educational institutions (Loftin et al., 1993, p.67).

3.6.7. Other applications of VRML

There are literally hundreds of applications where VRML can make a contribution. If one thinks of the way in which computers have evolved from command line DOS applications to 2D Windows-based applications, it is only natural that interfaces of the future will be 3D because everything around us has three dimensions.

Other interesting applications of VRML include art and design systems, training systems (such as the Medical Readiness Trainer at http://www-vrl.umich.edu/mrt/), accident simulation (http://www-vrl.umich.edu/project/accident/) and online product documentation and support.

3.7. Future

Because VRML is still a relatively new Web language, there is still a lot of scope for improvement. The Web3D Consortium, which represents all aspects of 3D technologies on the Internet, facilitates various task and working groups on 3D related topics. These include groups on the EAI, VRML streaming, Universal Media, Virtual Reality Transfer Protocol and object-oriented extensions, to name a few. Although these areas of interest are only at the working group stage, the Web3D aims to set standards in these areas that will form the basis of 3D on the Web in the future. Some of these areas will now be briefly discussed.

3.7.1. Universal Media (UM)

The aim of UM (formerly known as Universal Media Element Library) is to create a small, cross-platform library of locally resident media elements and a universal method of incorporating these elements into 3D worlds. This library will perform two important tasks namely to improve the realism of online 3D worlds and to decrease network downloading times or required bandwidth (Walsh, 1999). The elements contained in this library include textures, sounds and 3D objects.

UM will enable a 3D scene author to include references to these elements in a 3D world. The element library will be loaded onto the end user machine once. When the 3D world is loaded, only the actual world needs to be loaded through the network. The media elements are loaded from the locally resident library.

The referencing mechanism used by UM is Uniform Resource Names (URN) (Mitra et al., 1999). "URNs are location-independent pointers to a file or to different representations of the same content" (VRML Consortium, 1997). This means that the elements in the library may be referenced independently from where they physically reside. This combined with the fact that url fields in VRML are multi-valued (i.e. they can take a list of alternative urls) enables scene authors to create media-rich 3D worlds of high quality.

3.7.2. Java3D

Java 3D is a cross-platform API (Application Programming Interface) developed jointly by Intel Corporation, Silicon Graphics, Apple Computer and Sun Microsystems. It was developed out of the need to fully integrate Web-based multimedia because normally a number of plug-ins needs to be loaded that enables the Web browser to display the multimedia. Various other Java Media APIs are available e.g. speech API, 2D API and animation API.

The API can be used to create high-performance stand-alone 3D graphics applications or Web-based 3D applets through its rich set of features in a true object-oriented environment for rapid application development (Sun Microsystems, 1997, p.3, 6). The Web3D Consortium have established a Java3D-VRML working group to

aims to find ways to bring VRML and Java3D closer together and create a better interoperability between the two.

3.7.3. VRML networking protocols

VRML provides the capability to construct large-scale virtual environments (LSVE) for the Internet. The network support that HTTP provides to VRML, however, is insufficient for such LSVEs (Brutzman, 1997, p.179). The Virtual Reality Transfer Protocol (VRTP) is an application layer protocol to extend HTTP's client-server capabilities with peer-to-peer communications and network monitoring. The VRTP is designed to support VRML as HTTP is designed to support HTML.

The authors of VRTP do not aim to create a new protocol but this protocol will consist of an optimised set of existing but currently separate components. This includes HTTP, Java agents, network monitoring etc.

The VRTP working group aims to provide client, server, multicast streaming and network-monitoring capabilities in support of internetworked 3D graphics and LSVEs.

The realisation of shared virtual worlds on the Internet is still an unresolved issue. The Distributed Worlds Transfer and communication Protocol (DWTP) is another protocol that provides a scalable architecture for distributed shared virtual environments on the Internet (Broll, 1998, p.50). It is an application layer protocol similar to VRTP. It is built on top of TCP/IP and UDP/IP and therefore provides connectionless and connection-oriented services depending on what type of data needs to be sent.

3.7.4. Database integration

Coors and Jung developed a prototype system that uses VRML to access data in a 3D data warehouse (1998, p.121). The system is used in an urban planning application where users can interact with 3D models of the city of Frankfurt, Germany. The system uses VRML and Java (EAI) for the front-end and CORBA as a means of accessing data in a spatial database.

There is no dedicated database support for VRML yet but the EAI and Java scripting can be used to establish database access using the above method. Coors and Jung proposes a new name dictionary management node and a SQL node that can be included in the core VRML language that will eliminate the need to write complex custom code to access databases (1998, p.121). The Web3D Consortium's Enterprise Technology Working group has as one of its goals the setting of a specification for the integration of CORBA into VRML.

3.8. Conclusion

VRML was designed with the Internet in mind. VRML file sizes normally tend to be small for moderately complex worlds. Because of this reason, VRML can be effectively used as an online data visualisation tool. If one compares the file size (and therefore the required bandwidth) of a world describing a machine that animates the machine's behaviour to a medium such as streaming MPEG to monitor the same machine, the difference in required bandwidth is immense. One

disadvantage of VRML the fact that 2D input (mouse) and output (screen) devices are used to control and view 3D worlds.

All in all VRML can be an excellent visualisation tool for the Internet and the possibilities with regards to applications seem limitless. Only time will tell whether this medium will succeed in convincing the Internet community of its usefulness or whether it will be replaced by something else. For now it seems a viable tool in the hands of 3D creators.

CHAPTER 4 REMOTE CONTROL OF INDUSTRIAL ROBOT – SYSTEM ARCHITECTURE

This chapter will provide a brief overview of the system architecture in terms of the required hardware as well as the software components to enable the remote control of an industrial robot. It provides the basis of how the system was implemented in this research project.

4.1. Introduction

Figure 4.1 shows the system architecture of this project. It can be seen that this project is largely focussed on the software components for three hardware subsystems.

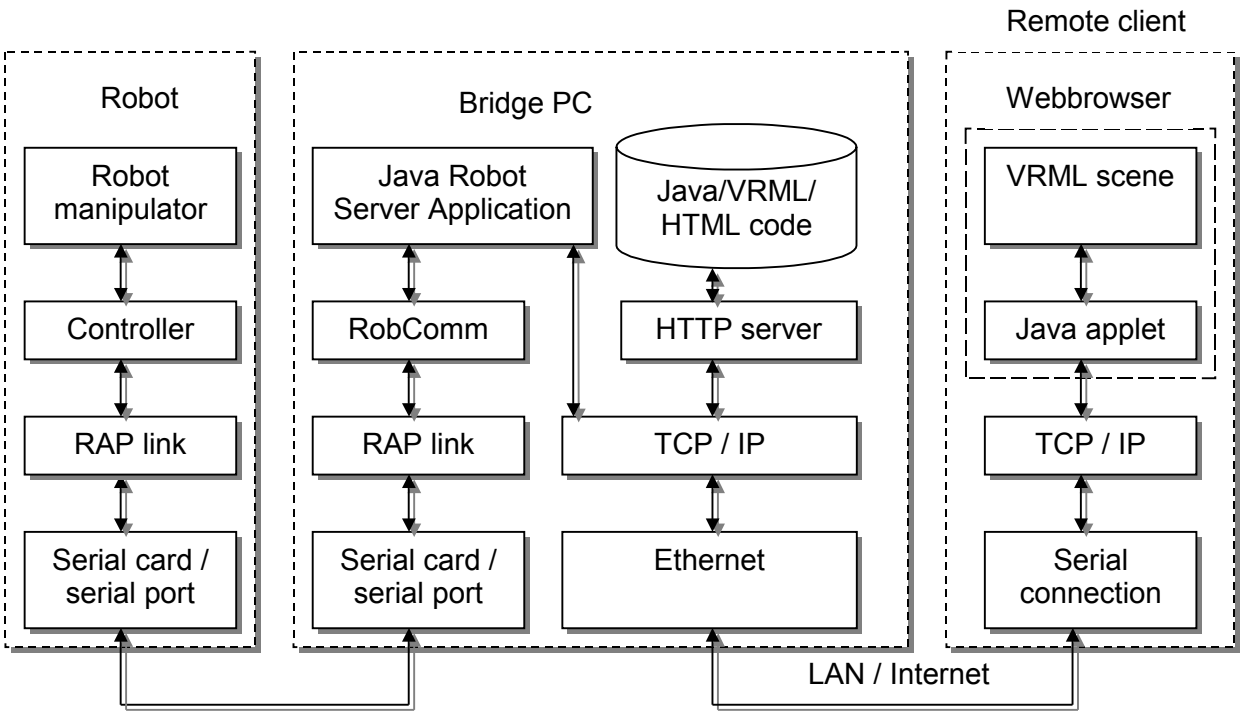


Figure 4.1 System architecture

The research is aimed at connecting a physical manufacturing system (an industrial robot) to a virtual manufacturing system (a robot model). Once these two are connected an Internet user (client) is able to monitor and control the physical manufacturing process using the virtual system as an interface to the real one. The virtual system was realised in the form of a VRML robot model. A Java applet on the client machine acts as a communications and control interface between the VRML model and the robot server application. The VRML model and Java applet are both embedded on a Web page running in a Web browser container on the client machine. Therefore the Java applet and the robot server application communicates in a peer-to-peer manner using a client-server model.

The bridge PC is set up as a Web (HTTP) server. This means that the machine has a host name and IP address and has a server application running that listens on port 80 for HTTP requests. Once a request for a specific page is made, the server sends the page to the machine requesting it. The Web page containing the VRML model and Java applet was published to the Web server on this machine making it accessible to other LAN users. When a user requests this Web page, the VRML model is displayed and the applet starts execution. The applet enables the user to connect to the robot server application through sockets.

Once connected, the user monitors the physical robot operation through the VRML interface. The user may attempt to log on as a control user if he wishes to control the robot as well as monitor it. Otherwise, if the user remains a monitor, he will receive robot data updates from the robot server application as changes are made by the control user (if any) or by the operator of the robot server application.

The robot server application runs on the bridge PC and this application simulates the physical robot data and parameters because at the time of this writing the robot is operating stand-alone. This means that it is not connected to the bridge PC. The PE Technikon has purchased an Ethernet card and the required software for the robot but it has not yet been received. This research will aim to explain the complete system with the assumption that these components are in place. Chapter 5 will cover the section to explain how the robot must be physically connected and how the software must be incorporated into the developed system.

4.2. Hardware subsystems

Three distinct hardware subsystems can be identified namely the robot, the bridge (PC) and the remote client machine. These will now be discussed.

4.2.1. Industrial Robot

The robot being used is an Asea Brown Boveri (ABB) IRB1400 industrial robot. This robot was used as it was already installed at the research facility where the research was undertaken (MTRC – PE Technikon). The robot consists of two components namely the robot manipulator and the controller.

The robot manipulator has six axis, with spherical-jointed geometry, and therefore allows six degrees of freedom (6DOF), i.e. three axis for positioning and three axis for orientation. The robot was designed for manufacturing applications and specifically flexible robot-based automation. It is well suited to communicate with

and control external peripherals. The robot has a handling capacity of 5kg (gripper and load) and a work envelope of approximately 1.444m radius with a repeatability of 0.05mm.

The S4C controller is a standard controller that is used to control 22 different ABB robots, which include six axis versions as well as Cartesian type gantry systems (Scouten, 1999, p.41). The controller contains 3 onboard computers to achieve control of the robot, its axis and to perform input/output (I/O). An operating system called BaseWare runs on the computer and is similar to DOS for PCs. It provides complete control of the robot to control program execution, communication and motion control (ABB, 1998a, p.2-3). Application programs are able to run on top of this operating system. When support is required for specific application areas such as packaging, spot and arc welding, painting and coating, and gluing, optional software packages can be installed.

The programming language that is used to program the robot is called RAPID and is similar to a high level language such as C. A program may consist of a number of modules, which include user-defined modules and system modules. A module is made up of a set of data and functions. A program may contain a number of modules and only one module will implement a main function, which is the entry point of the program (similar to a C program). System modules contain system specific routines and data (ABB, 1998b, p.4-8).

A teach pendant is connected to the controller and is used as the control and programming user interface. Pull-down menus, dialogs and windows are used to

display information to the user and touch keys and a joystick are provided as input devices. Figure 4.2 displays the IRB1400 robot and controller.

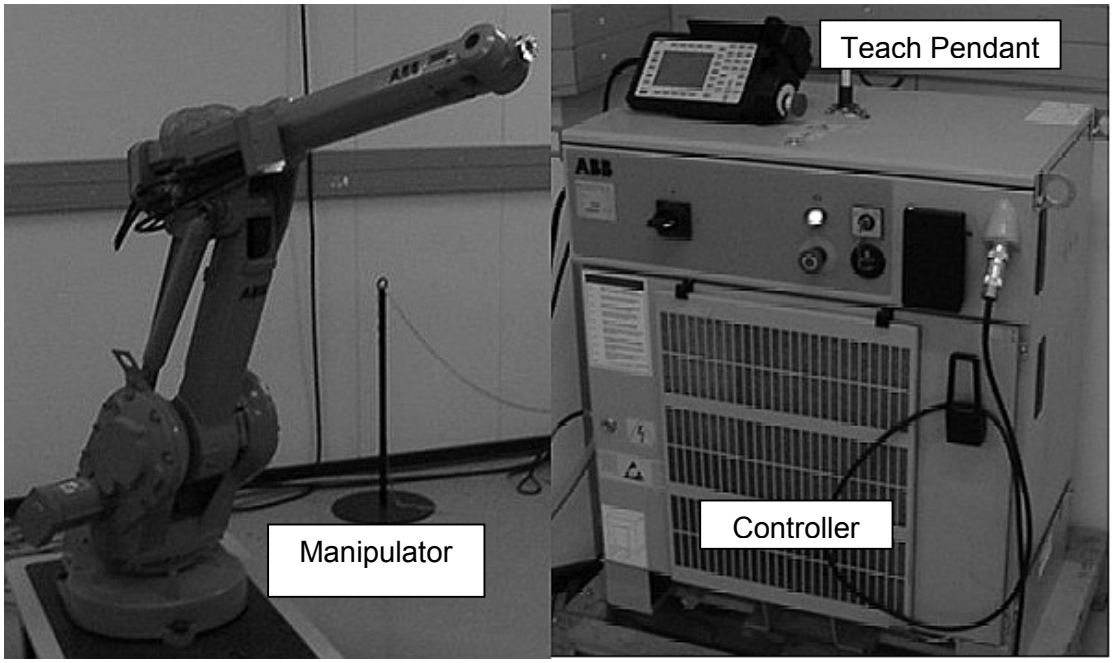


Figure 4.2 IRB1400 robot and controller

The controller provides an RS-232 port and an RS-422 port for point-to-point serial communications with a PC or printer. When higher speed communication is required between the robot and a PC or between a network of controllers and a PC, a special Ethernet card may be fitted to the controller. Each controller is then linked to a hub that also connects to the PC. There are various dedicated software packages available to add to the base operating system. For network communication there are two software options, i.e. advanced functions for BaseWare operating system and Robot Application Protocol (RAP). The former provides the ability to only send data from the robot to any device (PC or printer) on the other end of the connection. RAP provides the ability to send data from the robot to a PC and vice versa.

4.2.2. Bridge PC

The bridge PC consists of an office PC (300MHz / 128MB) running Windows NT 4. It is connected to a LAN via Ethernet and the LAN is connected to the Internet through a firewall. The bridge PC is set up as a local Web (HTTP) server so that Web pages published to it is accessible to everyone on the LAN. For this reason it needs to have an IP address and hostname. The PC that was used had the host name "heinv" (URL http://heinv.petech.ac.za/) and was assigned a dynamic IP address (using a DHCP server). The host name is resolved to an IP address once a request is sent to the server and thus it is better to use the host name rather than the IP address (because the IP can change).

As can be seen from Figure 4.1, the bridge PC is used to run the main robot control application and to act as a server with which remote clients can interact to monitor and control the robot.

4.2.3. Remote client

The remote client is any PC or workstation that has access to the Internet whether it be through a LAN connection or other Internet connection. The only requirements for the remote machine are in terms of software, i.e. it must have certain software components loaded onto the machine.

4.2.4. Conclusion

The robot in its current state has no gripper mechanism attached to it although an intelligent gripper is currently being designed by researchers at the PE Technikon.

Because of this reason the robot is not used in any specific application, e.g. materials handling.

4.3. Software components

There are 4 major software components that this research is based on. These are made up of the VRML model, the client control and communications applet, the robot server application and the RobComm ActiveX component. It can be seen from Figure 4.1 that peer-to-peer communication is realised between the different software levels in the remote client and the bridge PC as well as between the robot controller and the bridge PC. The software components will now be briefly discussed.

4.3.1. VRML model

The VRML model of the industrial robot is a replica of the real IRB1400 robot in terms of dimensions and geometry. The model was originally constructed by ABB in Sweden in VRML 1.0. This model was very large (approximately 250KB) because it was exclusively made up of polygons and provided no user interaction. The model was edited (with permission from ABB) and rewritten in VRML97, which enabled user interaction as well as the ability to control the model using the EAI. The file size of the VRML97 model was also considerably cut to 95KB and after GZIPing only 20KB.

The robot model on its own does not provide any control apart from viewing the model at any angle. The robot model must be linked to the EAI to accomplish control of the model. Figure 4.3 shows the VRML robot model.

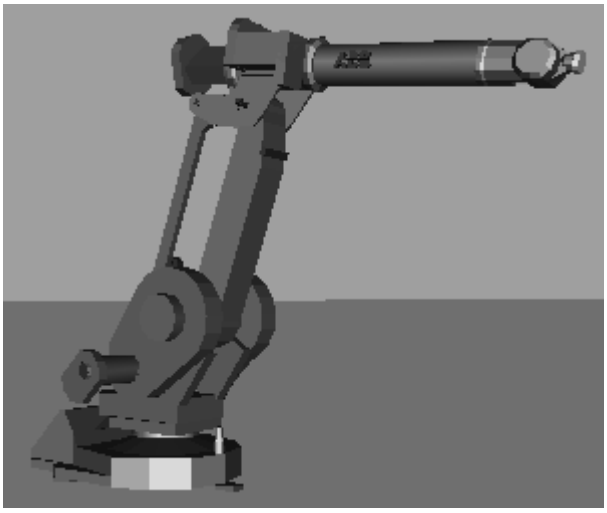


Figure 4.3 IRB1400 VRML robot model

4.3.2. Client control and communications applet

The client Java applet runs in a Web browser on a remote machine. The main tasks of the applet is:

- To communicate with and control the VRML robot model using the EAI;
- To establish communication with the robot server application and provide a communications protocol in order to monitor or control the robot;
- To display the robot status to the remote user; and
- To provide user authorisation so that only one authorised user is allowed to control the robot at any time.

Figure 4.4 shows the client Java applet as it would appear on the Web page.

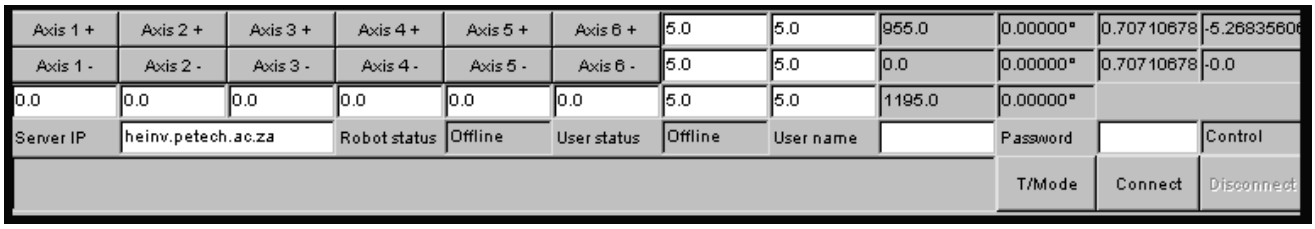


Figure 4.4 Client Java applet

4.3.3. Robot Server Application

The robot server application is written in Java and runs on the bridge PC. It performs the following tasks:

- Act as a server to which clients can connect to monitor or control the robot;
- Simulate robot parameters when the robot is not connected to the application;
- Act as master controller of the robot, i.e. it can override any currently connected control user;
- Receives updated information from a control user and sends this information to all connected users;
- Connect to robot by means of the RobComm ActiveX component.

Figure 4.5 shows the robot server application.

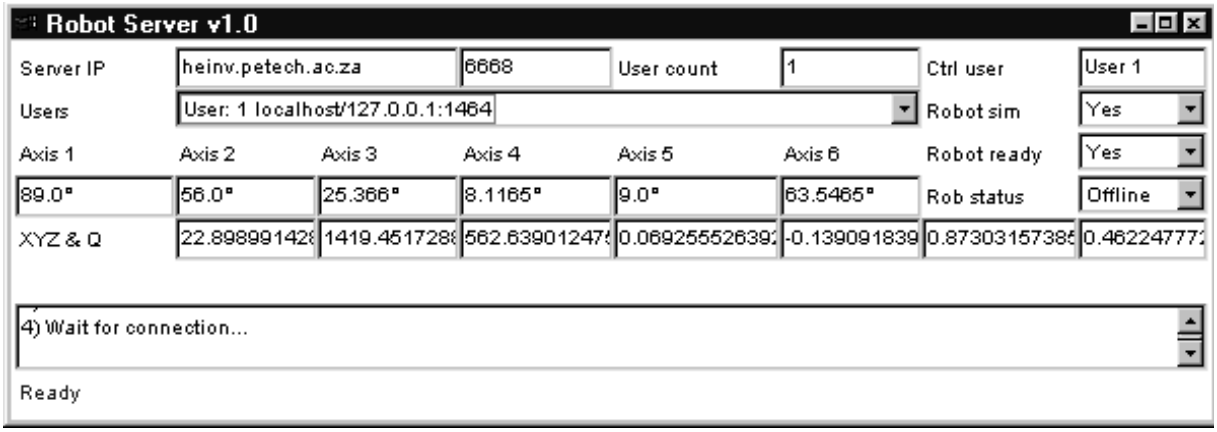


Figure 4.5 Robot server application

4.3.4. RobComm ActiveX component

ABB provides a complete suite of applications and utilities for use with their robots. These can be divided into applications running on the robot controller (on top of BaseWare operating system) and applications running on the PC connected to the robot.

The applications running on the robot controller include utilities for specific application areas e.g. gluing (GlueWare) or spot welding (SpotWare). Then there are utility applications (also running on the robot controller) for communicating with a PC serially (be it via Ethernet or a serial port) e.g. Ethernet Services, RAP Communication and the FactoryWare Interface. The FactoryWare Interface includes the RAP communication software and it enables the robot system to communicate with a PC when RobComm is loaded on the controlling PC.

The software packages that run on the PC include DeskWare and FactoryWare. DeskWare include applications for training people to use the robot, creating robot programs and online documentation. FactoryWare include applications for programming operator interfaces and monitoring robotic cells. RobComm is a collection of ActiveX controls that is part of FactoryWare.

The RobComm ActiveX controls can be used by applications written in MS Visual C++, MS Visual Basic, Wonderware InTouch (v7.0) and MS Visual J++ (Java). It frees the programmer from the underlying communication protocol to concentrate on creating Windows-based applications to remotely control and monitor the robot. RobComm was designed to enable even multi-threaded applications to communicate

with a robot or with a network of robots. The following is a list of functions that can be performed using RobComm and the FactoryWare Interface (ABB, 1998a, p.20):

- Start and stop program execution;
- Transfer programs to/from the robot;
- Transfer system parameters to/from the robot;
- Transfer files to/from the robot;
- Read the robot status;
- Read and write data;
- Read and write output signals;
- Read input signals;
- Read error messages;
- Change robot mode; and
- Read logs.

4.4. Conclusion

This chapter overviewed the various components of the project. The following chapter discusses these components in detail as well as how each of these components were implemented for remote control the industrial robot.

CHAPTER 5 REMOTE CONTROL OF INDUSTRIAL ROBOT – PROJECT IMPLEMENTATION AND INTEGRATION

The previous chapter gave a brief overview of the project components. This chapter will explain each component in detail including coding examples.

5.1. VRML robot model

5.1.1. VRML and robot coordinate systems

It is important to note that only one three-dimensional world coordinate system exist but variations may exist in the fact that they are transformations of this system. There does not seem to be a standard regarding this. The IRB1400 has a world coordinate system as shown in Figure 5.1(a) while the VRML97 specification defines a world coordinate system as shown in Figure 5.1(b).

Figure 5.1 World coordinate systems for (a) IRB1400 and (b) VRML

The VRML robot model of the IRB1400 is the only visual interface that a user has of the robot, apart from the robot data that is displayed by the applet that include joint angles and tool centre point (TCP) values. The robot model aims to show the exact relationship between different axis. The geometry and dimensions of the model is precisely that of the real robot to the nearest millimeter.

Designing a VRML model to be used in system like this requires that full advantage be taken of the inherent advantages that VRML offers. The design criteria that was followed in the design of the robot model is given in the following sub-sections.

5.1.2. File size

Every time that a VRML file is loaded through the Web, the file in its entirety must be transferred. Fairly complex worlds can have file sizes that range from approximately 100KB up to a few hundred kilobytes. This takes up time, e.g. a 100KB file takes 28.4 seconds to download at 28.8 kilobits per second. Because VRML files are written in UTF-8 encoding and normally contains a number of whitespace characters, the file can be made much smaller by zipping them. The VRML robot model is 95KB and after zipping only 20KB. This means that the file can be downloaded in less than six seconds.

5.1.3. VRML97 compliance

The robot model is completely VRML97 compliant and can (theoretically) be viewed in any browser that is also VRML97 compliant. As stated earlier, the original robot model was created in VRML 1.0 and made up entirely out of polygons. This was changed by substituting a large amount of polygons with VRML primitives. VRML primitives are standard geometries built into the language. These include, among others, spheres, boxes, cylinders and cones. This is also the main reason why the file size was diminished.

5.1.4. Prototyping

Each individual object of the robot model was created using a prototype. This has many advantages as summarised with the following:

- There is no need to create an instance of an object (node) while creating an object (node) interface;
- File size is kept smaller by code reuse; and
- Prototyping provides an object-oriented structure to the scene that is easier to conceive than a system where all code is dumped into one segment.

Prototyping was also used to ensure that behaviours required for the model was contained within the prototype. One example of where behaviour was used is for a set of levers that drive Axis 3. The following code segment shows the prototype of one of these levers. The prototype receives a rotation eventIn. When the user changes the angle of Axis 3 (through the EAI), this value is routed to all the necessary nodes and prototypes. This specific prototype uses a script node to change the angle of this lever so that the robot functions as it should.

```
PROTO bo18 [ eventIn SFRotation rot ]{
  DEF BackLever Transform {
    translation 13 -38 475
    centre -33 0 600
    children [
      bo39{ rot IS rot } #power lever
      Shape {
        appearance Orange{}
        geometry IndexedFaceSet {
          coord Coordinate {
            point [
              0 0 0
              -4.4 0 16.5
              …]}
          coordIndex [
            0 1 2 3 4 5 6 7 8 9 10 11 -1
            52 51 101 -1
            … ]
```

```
  ]}}]}

  DEF BackLeverScript Script {
  eventIn SFRotation rotation IS rot
  field SFRotation tempRot 0 0 0 0
  eventOut SFRotation newrotation
  url "javascript:
    function rotation(val){
    tempRot = val; tempRot[3] = -tempRot[3];
    newrotation = tempRot;
    }"}
ROUTE BackLeverScript.newrotation TO BackLever.set_rotation
}
```

5.1.5. Kinematics

The robot model has to be kinematically correct. This means that all the joint position relationships has to be exactly the same as the real robot. VRML provides capability to make this possible. Each axis only has one axis of rotation and this is similar to the functionality that the rotation and centre fields of a Transform node provides.

5.1.6. EAI controllability

The robot model had to be controllable through the EAI. In order to do this proper named nodes had to be created for the individual axis of the robot. Each axis object was created using prototypes and these were then instanced inside Transform nodes that formed the links between axis. The robot itself could have been placed inside a prototype but it would have been more difficult to control it through the EAI because prototypes have their own naming scope. Therefore the EAI would not be able to access individual axis and would have to use an event routing mechanism to achieve control.

A hierarchical Transform node structure was set up to create the axis of the robot model. This is similar to the analogy of a human arm that was used in a previous chapter. When a higher level component in the hierarchy is moved or changes position, e.g. the upper arm, then the rest of the lower level components (e.g. the lower arm and hand) all move or changes position to stay in the same relationship to the higher level component. In the case of the robot model a change in the position or orientation of Axis 2 causes a change of position and orientation of Axis 3, 4, 5 and 6 but leaves Axis 1 unchanged. Following is the code of the main Transform node of the robot model.

```
DEF ROBOT Transform {
 rotation 1 0 0 -1.57
 children [
  DEF LINK0 Transform {
   rotation 0 0 0 0
   children[
    DEF BaseEvent Base{}
    DEF LINK1 Transform {
     rotation 0 0 1 0    #Turn around the z-axis
     children [
      DEF A1Event Axis1{}
      DEF LINK2 Transform {
       rotation 0 1 0 0  #Turn around the y-axis
       centre 150 0 475
       children [
        DEF A2Event Axis2{}
        DEF LINK3 Transform {
         rotation 0 1 0 0#Turn around the y-axis
         centre 150 0 1075
         children [
          DEF A3Event Axis3{}
          DEF LINK4 Transform {
           rotation 1 0 0 0   #Turn around the x-axis
           centre 150 0 1195
           children [
            Axis4{}
            DEF LINK5 Transform {
             rotation 0 1 0 0 #Turn around the y-axis
             centre 870 0 1195
              children [
               Axis5{}
               DEF LINK6 Transform {
                rotation 1 0 0 0 #Turn around the x-axis
                centre 955 0 1195
                children ]
```

```
            Axis6{}
                ]}]}]}]}]}]}]}]
}
```

The centre field in the Transform node is used to specify the centre point of rotation of the node. The axis of rotation is determined by the translation and rotation fields of the node. It will be noticed that all the Transform nodes have translations of 0 0 0 (by default). This is because originally each object was created with reference to the same point and not with reference to each other as it should be. So the translations for each object is contained within each prototype. Although this was not corrected, it was not seen as a major problem and therefore left as it was.

It can be seen that each link in the main Transform node contains a centre and rotation field. The robot model was created with the initial position of the robot model being the home position of the real robot with all the joint angles set to zero degrees. The rotation field for LINK1 (robot base – Axis 1) is for example:

```
rotation 0 0 1 0    #Turn around the z-axis
```

The first 3 values indicate a 3D vector (x, y, z), which is the axis of rotation, and the last value indicates the actual angle of rotation. Each axis has its own axis of rotation because it is contained within its own Transform.

The model provided a minor challenge in the fact that Axis 3 of the IRB1400 does not work exactly as in the hierarchical structure as explained above. The angle of Axis 3 is taken with respect to the ground and not with respect to Axis 2 but is still linked to the Axis 2. The problem was handled at a higher level in the Java applet by using the EAI.

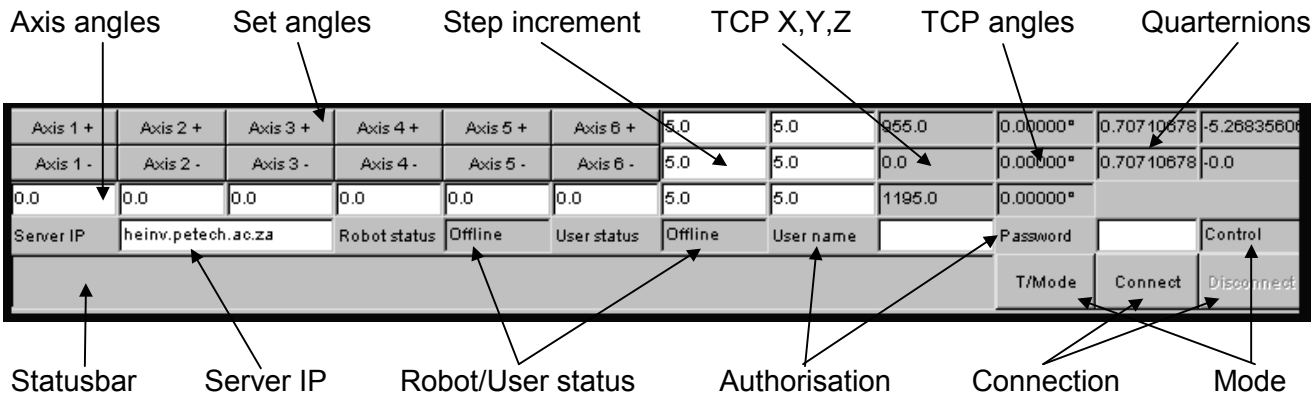
5.2. Client control and communications applet

The purpose of the control and communications applet is to provide a way to control the VRML robot model through the EAI and to communicate with and pass information to and from the robot server application that controls the physical robot.

The applet consists of a number of Java classes where the functionality for a specific area is implemented with one or more classes. These areas include the graphical user interface, robot control, kinematics, communication, authorisation and user mode, and will be discussed next with reference to the classes used.

5.2.1. Graphical User Interface (GUI)

The GUI provides the interface that will be used to control the robot model and view the status of the network and physical robot. The GUI is the front end of the applet. Figure 5.2 shows the GUI with details of the elements it contains. These elements



will be explained briefly and then their implementation will be discussed.

Figure 5.2 Applet GUI

- GUI elements

The angle of each axis can be set in one of two ways. The axis angle can be incremented or decremented by a variable number of degrees by pressing the increment/decrement buttons, e.g. the button labelled “Axis 1 +” to increment Axis 1. The second method to set an axis angle is by entering the new axis angle in the axis angle’s TextField.

The step increment is the value that is decremented or incremented when the above method is used to change an axis angle. There is a step increment angle for each of the axis so that each axis can be increment or decremented by a custom value. This, however, was not implemented in the final applet implementation and only one step increment is used for all axis.

The tool centre point (TCP) position values (x, y and z) are shown in millimeters. The tool coordinates are given in two forms namely in quarternions (q1-q4) and in Cartesian space angles (roll, pitch and yaw in degrees). Only the quarternions have been implemented in the final applet. These calculations have been done based on the kinematic equations that will be covered in a later section.

A statusbar at the bottom of the GUI gives details of networking related information. For example when the user is trying to log on to the server as a control user and this operation is unsuccessful, details regarding the operation and messages returned from the server will be displayed in the statusbar. It may also contain information that would be useful for the user.

The Server IP is the hostname or IP address of the machine where the robot server application is running. As will be seen later, this is the same machine where the applet is loaded from.

The robot status provides a way of showing the user whether the physical robot is currently connected or not. The user status shows whether the user is connected to the robot server application or not. When the user is working offline, the values displayed for the robot status, position and angle values are for the robot model are not data derived from the robot server application.

The username and password fields are used when a user wants to become a control user (these details will still be covered). The password field shows asterisks ('*') when letters are typed for security reasons.

The connect and disconnect buttons enable the user to connect or disconnect from the server. When the user is already connected, only the disconnect button is enabled to avoid unnecessary reconnection and vice versa.

The mode control status and mode control button allow the user to change mode, i.e. from monitor to control and vice versa (modes will still be covered). The mode shows in what mode the user currently is and the mode button (T/Mode – toggle mode) is pressed to request a mode change from the server.

- Implementation

The components used in this GUI are all part of Java's Abstract Windowing Toolkit (AWT) package of classes (i.e. java.awt). These classes include basic user interface elements such as Buttons, TextFields, TextAreas and Labels. A GridBagLayout was used to position the elements on the applet where this layout is similar to the layout of a table in MSWord.

Java GUI elements are event-driven meaning that when an action occurs (e.g. a button is pressed) an event is generated. The underlying windowing system ensures that a Java program receives the event. Java uses an delegation event model, which means that the event-handling is delegated to an object in the program (Deitel & Deitel, 1998, p.524). The programmer has the task of creating the event handler object. This is done in the following steps:

- Create a class that implements the listener interface for events of a specific type(s);
- Create n instance of the class in the main program;
- Register each GUI element with an event listener; and
- Implement the event-handling code (inside the class) for each type of event that the class can handle.

The following code segment shows an example of an event handler class. The class catches events of type ActionEvent and contains the function actionPerformed that executes automatically once such an event occurs.

```
class ControlHandler implements ActionListener {
  protected mainApp mapp;

  public ControlHandler(mainApp m) {mapp = m;}
```

```
  public void actionPerformed(ActionEvent ae) {
    if(mapp.tftheta[i]==ae.getSource()) {
     float inputVal = Float.valueOf(
              ae.getActionCommand()).floatValue();
     mapp.rob.ChangeAxisAngle(i+1, inputVal);
     }
    … //handle other events
  }
}
```

The following code segment shows an example of the class instancing and registering a GUI element with an event listener object.

```
//Main initialization code
…
//Instance handler class and GUI element
CtrlHandler = new ControlHandler(this);
btConnect = new Button("Connect");
//Register button with handler
btConnect.addActionListener(CtrlHandler);
…
```

The implementation of the GUI was done inside the main applet class (mainApp.class) because the applet does not have a separate view class that handles the GUI elements. In Java this is done in the application (or applet in this case) class. Two event handler classes were used. ConnectionHandler.class does the event-handling that relate to the network connection and ControlHandler.class does the event-handling for GUI elements that relate to the control and monitoring of the VRML model.

Java is based on objects and an object must have a reference to another object in order to use the other object or its functions. Each event-handler class receives a reference to the main applet class so that code within the event-handler will have access to the other objects defined in the main applet.

The ExTextArea.class is inherited from the TextArea AWT class. A TextArea class is a multi-line text box. The class was inherited with the sole purpose of overriding the append function for use as the statusbar text box. The append function is normally used to append text to the end of the string displayed in the text box. The append function was changed so that users of the function need not be concerned with inserting newline characters ('\n') or line numbers. This is done automatically with each entry to the text box. The following code segment shows the overridden class ExTextArea.

```
public class ExTextArea extends TextArea {
  int i=1;
  public ExTextArea(int p1, int p2){ super(p1,p2); }
  public synchronized void append(String p1) {
    super.append(i + ") " + p1 + "\n");
    i++;
  }
}
```

5.2.2. Robot control

This software component concerns itself with controlling the VRML model of the robot. The data from the real robot is used in modeling the VRML model because these two must correlate with respect to position and orientation.

The IRB1400 may be controlled in primarily one of two ways namely by specifying the joint angles of each axis or by specifying the tool centre point (TCP) data (x,y,z, roll, pitch and yaw values). As will be seen in the following section on kinematics, when using the former method joint angles can be directly used to position the real robot and robot model. The latter method, however, requires that the TCP data be manipulated using inverse kinematics. The inverse kinematics may provide different

solutions for the joint angles that must be used to control the real robot and robot model. Therefore it was decided to only use the former method in controlling the real robot and robot model. This was partly also due to the complexity in solving for the inverse kinematics.

Control of the VRML model is by means of the EAI that was discussed in detail in Chapter 3. The EAI was used in this project for the following reasons:

- The Java applet is able to use networking and threading capabilities;
- Java objects can be created to form clean code and interfaces;
- The programmer has access to all named nodes inside the VRML file (in contrast to JSAI); and 2D data, e.g. values as text, can be displayed better in the applet than in VRML.

Two classes were created to handle robot data and VRML scene data respectively namely Robot.class and Axis.class. The robot class contains an array of the six axis objects and a kinematics object that handles the kinematic calculations. The axis objects in the robot class handle axis and VRML scene data details.

- Robot class

The robot class provides a layer of abstraction between the main application and the axis data through set and get functions, e.g. setting or retrieving one or all axis angles or step increments. It therefore hides the axis implementation details from the main application. The class also provides functions that use the kinematics object to determine the final result to the direct kinematics of the robot from the data

in the axis objects. The following code shows a large part of the robot class with an explanation of important functions and variables.

```
public class Robot {
 //Axis object array
 protected Axis[] theAxis = new Axis[NumberOfAxis];
 Kinematics kine; //Kinematics object
 // max/min angles required by robot calculations
 protected final double MAX_ANGLE = 153.8;
 protected final double MIN_ANGLE = 20;
 public Robot() { //Constructor
  … // Initialization
 }
 //Overridden function to calculate forward kinematics
 // with angles from the Axis array
 public void CalcFK() {
  kine.CalcFK(theAxis[0].getAngle(),…
              theAxis[5].getAngle());
 }
 //Overridden function to return XYZ of tool centre point
 public double[] GetFK_XYZ() {
  //first do calculation, then return value
  CalcFK();  return kine.GetFK_XYZ();
 }
 //Overridden function to return quarternion of tool
 // centre point
 public double[] GetFK_Quarternion() {
  //first do calculation, then return value
  CalcFK(); return kine.GetFK_Quarternion();
 }
 //Get node references from the VRML scene and store so
 //scene objects can be manipulated using them
 public void setAxisNodes(Browser browser) {
  for(int i=0;i<getNumberOfAxis();i++) {
   theAxis[i].setNode(
              browser.getNode(theAxis[i].getLinkName()));
   try{
    theAxis[i].setEventIn((EventInSFRotation)theAxis[i]
                 .getNode().getEventIn("set_rotation"));}
   catch (InvalidEventInException iee)
   {iee.printStackTrace(); }
  }
 }
 //Function to increment or decrement a specific axis.
 // The angle is incremented by preset step value that is
 // set for each axis.
 public void ChangeAxisAngle(int axisno, Boolean
                                          increment){
  float angle=0;
  if(increment==true)
  angle = theAxis[axisno-1].getAngle() +
                            theAxis[axisno-1].getStep();
  else angle = theAxis[axisno-1].getAngle() -
                            theAxis[axisno-1].getStep();
```

```
  ChangeAxisAngle(axisno, angle);
 }
 //Overloaded function to increment or decrement a
 // specific axis. Angle is incremented by the preset
 // step value set for each axis.
 public void ChangeAxisAngle(int axisno, float angle){
  if((axisno<2) || (axisno>3)) {
   theAxis[axisno-1].setAngle(angle);
   theAxis[axisno-1].setRot(theAxis[axisno-
                                    1].getRadAngle());
   try {theAxis[axisno-1].getEventIn().
                 setValue(theAxis[axisno-1].getRot());}
   catch (IllegalArgumentException iae) {…}
  }
  //Axis 2,3 are handled separately because of control
  // problems
  else if(axisno==2){…}
  else if(axisno==3){…}
 }
 //Set step increment for all axis
 public void setAxisStep(float step) {
  if(step > 0 && step <=30) //angle range checking
   for(int i=0;i<getNumberOfAxis();i++)
    theAxis[i].setStep(step);
 }
 //Get step increment value - all increments are equal
 public float getAxisStep(){return theAxis[0].getStep();}
 //This function is used when all the axis must be
 // changed at once. ChangeAxisAngle() cant be used
 // because angles 2/3 will cause problem when setting
 // the one and then the other.
 public boolean setAxisFromAngles(float ang[]) {
  for(int i=0;i<theAxis.length;i++){
   theAxis[i].setAngle(ang[i]);
   //FOR AXIS 2(SHOULDER) THE VRML ANGLE FOR AXIS 3 IS
   // SET TO THE -(AXIS_2_ANGLE) TO COMPENSATE FOR THE
   // WAY IN WHICH THE VRML LINKS OPERATE.
   if(i==1) theAxis[i+1].setVRMLAngle(-ang[i]); //axis 2
   theAxis[i].setRot(theAxis[i].getRadAngle());
   try{ theAxis[i].getEventIn().
                      setValue(theAxis[i].getRot());}
   catch (IllegalArgumentException iae)
   { return false; //INDICATE ERROR}
  }
  return true; //NO PROBLEM
 }
 //Returns all 6 axis angles
 public float[] getAxisAngles() {
  float temp[] = new float[6];
  for(int i=0;i<getNumberOfAxis();i++)
   temp[i] = theAxis[i].getAngle();
  return temp;
 }
}
```

- Axis class

The axis class stores data relating to individual axis, which include:

- axis angle;
- VRML node that relates to the specific axis;
- Axis name and VRML node name;
- Step increment; and
- High and low limits according to robot specifications.

Apart from providing set and get functions for the above data, the class provides range checking, especially for the axis angles. The following code shows most of the axis class.

```
public class Axis {
 … //declaration of variables and objects
 EventInSFRotation theEvent;
 Node theNode;
 float[] theRot;
 float angle; // in degrees
 //The vrmlangle below is mainly used because of the way
 // in which the vrml links differ from the actual angle
 // produced by the robot. Obviously the VRML scene
 // should produce a correct model of the robot and a
 // correct kinematic model
 float vrmlangle; //in degrees
 float step;
 float hilimit, lolimit;
 String axisName, linkName;

 public Axis() {
  … //instantiation of variables and objects
  … //set default values
 }

 //Set the VRML event to which the axis is linked
 public void setEventIn(EventInSFRotation e)
 {theEvent = e;}
 public EventInSFRotation getEventIn(){return theEvent;}

 //Set and get the VRML node to which the axis is linked
 public void setNode(Node n) {theNode = n;}
 public Node getNode() {return theNode;}

 //Set and get VRML axis rotation array
 public void setRot(float x,float y,float z,float r) {
```

```
  theRot[0] = x; theRot[1] = y; theRot[2] = z;
  theRot[3] = r;
 }
 public void setRot(float r) { theRot[3] = r; }
 public float[] getRot() {return theRot;}

 //Set and get new angle in degrees
 public boolean setAngle(float a) {
  if(a>=lolimit && a<=hilimit) { //range checking
   angle = a;return true;
  }
  else return false;
 }
 public float getAngle() {return angle;}

 //Set and get upper and lower limits for the axis angle
 public void setLimits(float a, float b)
 {hilimit = a; lolimit = b; }
 public float getHiLimit() {return hilimit;}
 public float getLoLimit() {return lolimit;}

 //Include the VRML angle because this function is used
 // mainly to draw the VRML scene.
 public float getRadAngle()
 {return (float)((vrmlangle+angle)*Math.PI/180);}

 //Inc- or decrementing overloaded functions
 public void incAngle() {
  angle+=step;
  if(angle>hilimit) angle = hilimit;
 }
 public void incVRMLAngle(float a) {vrmlangle+=a;}
 public void decAngle(){
  angle-=step;
  if(angle<lolimit) angle = lolimit;
 }
 public void decVRMLAngle(float a) {vrmlangle-=a;}

 //Set and get the axis names
 public void setAxisName(String s) {axisName = s;}
 public String getAxisName() {return axisName;}

 //Set and get the VRML link names for an axis
 public void setLinkName(String s) {linkName = s;}
 public String getLinkName() {return linkName;}
}
```

- Implementation

The main application receives events from GUI elements or from the server application (through the network) to change one or all of the axis angles. The ChangeAxisAngle function in the robot class is then called from the application. This

function changes the axis object data, reads back the data (after range checking) and writes this data to the VRML scene to change the model.

The VRML robot model is controlled in such a way that its geometry and motion (position and orientation) functions in the same way as the real robot functions. To achieve this the axis angles had to be kept in a specified range and the model had to be kinematically correct. The latter proved to be more of a challenge than the former. This is because of the motion of axis 3 of the IRB1400. All the other five axis operate in a hierarchical manner (as explained in Section 5.1), i.e. when axis 1 rotates all the other axis are transformed with respect to its transformation because they are attached to it. Axis 3, however, did not operate in this manner when the axis 2 angle was changed. It was as if the reference for axis 3 was taken as the ground level when the axis 2 angle was changed. Figure 5.3(a) shows home position of the robot with all the axis at 0°. Figure 5.3(b) shows how a hierarchically structured robot would respond when the axis 2 angle is incremented while the axis 3 angle remains unchanged and Figure 5.3(c) shows how the IRB1400 responds when the axis 2 angle is incremented with axis 3 unchanged. The figures clearly show that in a hierarchical structure axis 2 and 3 keep a fixed angle between them when the axis 2 angle changes, which is not the case for the IRB1400.

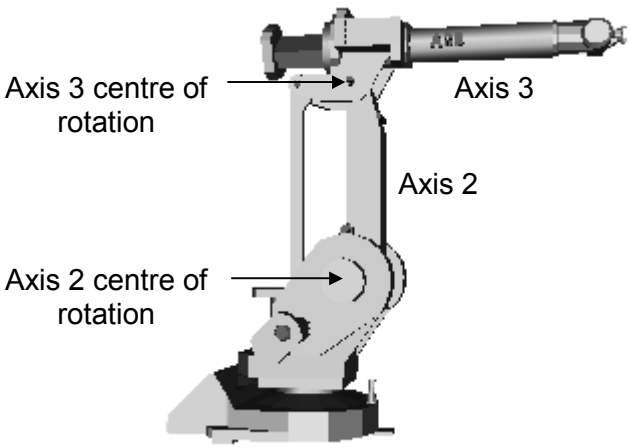


Figure 5.3(a) Home position of IRB1400

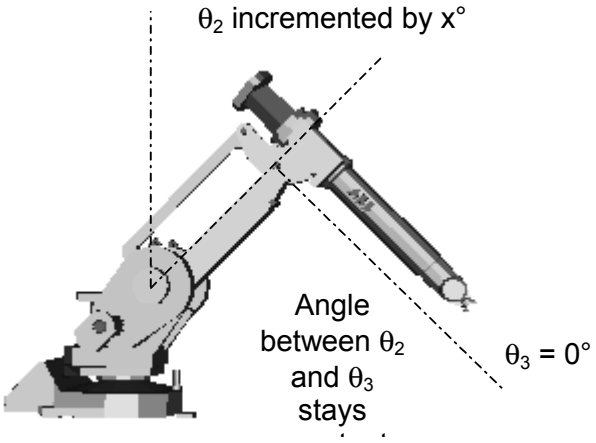


Figure 5.3(b) Axis move according to hierarchical structure

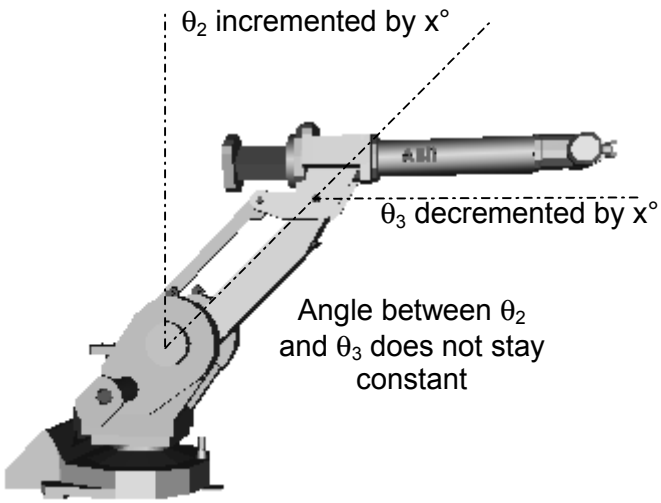


Figure 5.3(c) Axis 2/3 movement of the IRB1400

Because of the above discrepancy between the way that the real IRB1400 operates and the way the VRML model's hierarchy operates, adjustments had to be made in the control software (robot and axis classes). Fundamentally this was done by subtracting the axis 2 angle from the axis 3 angle to obtain the angle that must be used for axis 2 in the VRML scene. In Figure 5.3(a) both the angle of axis 2 and 3

are 0°. In Figure 5.3(b) axis 2 is changed to 45° while axis 3 remain 0°. The position of axis 3, however, should be as in Figure 5.3(c) in a upward (0°) position. To correct this the angle used to program axis 3 in the VRML model was changed to:

VRML angle for axis 3 = axis 3 – axis 2

Thus in the previous example:

VRML angle for axis 3 = 0° – 45°

= -45°

This problem was solved in this manner explained above. It must be noted that this correction could also have been made inside the VRML model using scripting and routing.

A further stumbling block was that of determining the physical limits of the robot where only axis 2 and 3 proved to be a problem. The limits according to the IRB1400 product manual (ABB, 1998a, p.33) for axis 2 are –70° to +70° and for axis 3 are –65° to +70°. It was found while controlling the real robot that axis 3 had different physical limits for different axis 2 angles. Its limits were not only confined to that of the specification but could reach –134° to +134° depending on axis 2. Axis 2 and 3, as a sub-section of the robot, can be compared to that of a two-link manipulator. It was realised that the physical limits of these two axis could be determined by looking at the minimum and maximum angle formed by the two axes. This was obtained by moving axis 2 to a maximum and axis 3 to a minimum and vice versa. This yielded a minimum difference angle of 20° and a maximum difference angle of 153.8°. These values were used to check that the robot remained within its physical limits as shown in the following code segment from the robot class.

```
float x = angle;  //Set new angle for axis 2
if(x>theAxis[1].getAngle()) {
 x1 = x - theAxis[1].getAngle();
 if((theAxis[1].getAngle())+x1 <=
                                theAxis[1].getHiLimit()){
  //Formula of final angle=90+step+ang2-ang3
  if((90-theAxis[2].getAngle()+theAxis[1].getAngle()+x1)
                                          <= MAX_ANGLE){
   //Increment axis 2 angle, decrement axis 3 angle
   // by same amount
   theAxis[1].incAngle(x1);
   theAxis[2].decVRMLAngle(x1);
}}}
else{
 x1 = theAxis[1].getAngle() - x;
 if((theAxis[1].getAngle()) - x1 >=
                                theAxis[1].getLoLimit()){
 //formula of final angle=90-step+ang2-ang3
 if((90-theAxis[2].getAngle()+theAxis[1].getAngle()-x1)
                                        >= MIN_ANGLE) {
  //Increment axis 2 angle, decrement axis 3 angle
  // by same amount
  theAxis[1].decAngle(x1);
  theAxis[2].incVRMLAngle(x1);
}}

theAxis[1].setRot(theAxis[1].getRadAngle());
theAxis[2].setRot(theAxis[2].getRadAngle());
try{
 //Write new angles to VRML scene
 theAxis[1].getEventIn().setValue(theAxis[1].getRot());
 theAxis[2].getEventIn().setValue(theAxis[2].getRot());}
catch (IllegalArgumentException iae) {}
```

5.2.3. Kinematics solution

In order to find and display the tool centre point values of the robot it was necessary to determine the direct kinematics from the given joint angles. A study was made on kinematics theory in Chapter 2. Using this theory the direct kinematic solution for IRB1400 was determined. This solution will now be discussed together with its implementation in the control software.

5.2.3.1. Solving the direct kinematics for the IRB1400

McKerrow (1991, p.177) defines an algorithm for obtaining the direct kinematic solution for the robot manipulator. The following algorithm will be used to show how the D-H representation solution was obtained:

- Move manipulator to zero position
- Assign joint coordinate systems to each link
- Obtain link parameters
- Define A matrices
- Multiply A matrices to obtain ${}^{R}T_{H}$
- Determine position of tool centre point
- Determine orientation of tool centre point

Each link of the manipulator is numbered from 0 to n, where n is the number of degrees of freedom (DOF). Each link is assigned a coordinate system (x_n, y_n, z_n) at the distal joint axis of link n, which thus corresponds to joint n+1. The coordinate system for link n is fixed in link n, and therefore moves with link n. The coordinate system is set up such that when the actuator for joint n moves, link n moves with respect to link n-1. The base is assigned the 0th coordinate system (x_0, y_0, z_0). In the case of the IRB1400, the tool coordinate system is (x_6, y_6, z_6).

Coordinate frames are established based on three rules in the correct order:

- Axis z_{n-1} lies in the axis of motion of the nth joint.
- Axis x_n is normal to z_{n-1}.
- Axis y_n is determined using the right hand rule.

The base coordinate system is chosen so that z_0 is in the axis of motion. The other two axes can be chosen at will, as long as it conforms to the right hand rule. An algorithm for the assignment of a set of coordinate systems will be discussed shortly.

The D-H representation requires four parameters, known as the link coordinate parameters (θ_n, d_n, a_n, α_n), which completely describe any revolute or prismatic joint. It characterises the transformation between adjacent links. A description of each parameter, making use of the right hand rule, is given next (Fu et al., 1987, p.37):

θ_n is the joint angle about z_{n-1} from x_{n-1} to x_n.

d_n is the distance from the origin of the $(n-1)^{th}$ coordinate system to the intersection of the z_{n-1} with x_n along z_{n-1}.

a_n is the offset distance from the intersection of z_{n-1} with x_n to the origin of the nth coordinate system along x_n (or the shortest distance between z_{n-1} and z_n).

α_n is the offset angle from z_{n-1} to z_n about x_n.

θ_n is the joint variable for rotary joints, and since d_n, a_n and α_n remains constant they are called the joint parameters. d_n is the joint variable for prismatic joints, and since θ_n, a_n and α_n remains constant they are called the joint parameters.

A procedure is necessary for establishing a set of consistent coordinate systems for the robot manipulator. Fu et al. details the procedure that has been used in this project. It will now be outlined briefly. First, the base coordinate system needs to be established. The z_0 axis is always along the axis of motion of joint 1 with the other axis normal to it as dictated by the right hand rule. In this project, the base

coordinate system was chosen to coincide with that of the IRB1400's base coordinate system.

For the remaining coordinate systems, the joint axes (z_n) and the origins of the nth coordinate systems must be established. The origin of a coordinate system is the intersection of z_{n-1} with z_n, or at the intersection of the common normal between them, along z_n. The x_n axis is found along the common normal between z_{n-1} and z_n, and may be in either the positive or the negative direction. The y_n axis is then determined by using the right hand rule. Finally, the link coordinate parameters can be obtained from the set of coordinate systems. The link coordinate system for the IRB1400 is shown in Figure 5.4 and the link coordinate parameters are shown in Table 4.2.

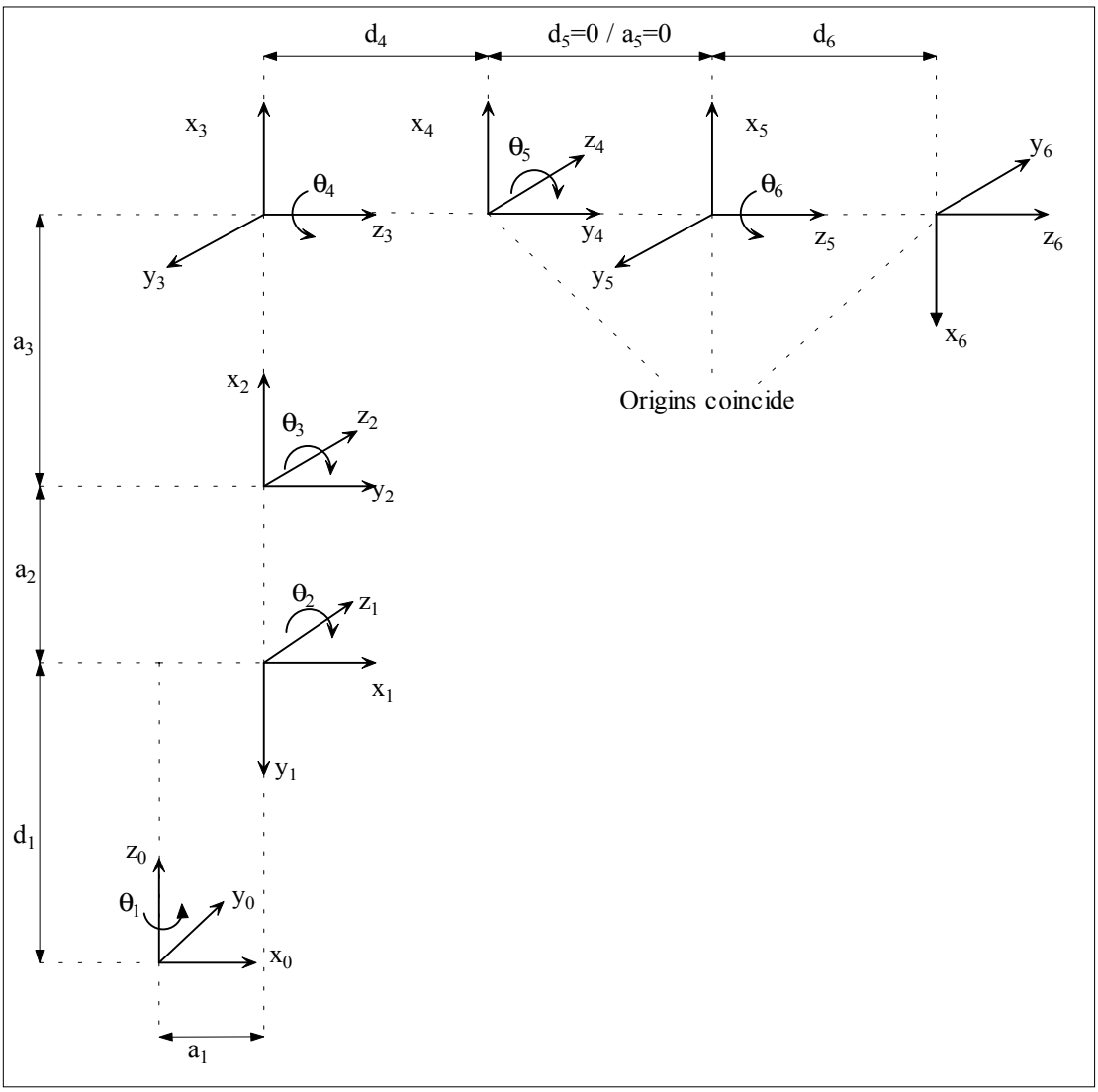


Figure 5.4 Link coordinate system for IRB1400

Joint i	θ_i	α_i	a_i (mm)	d_i (mm)	Joint Range
1	0°	-90°	150	475	-170° to +170°
2	-90°	0°	600	0	-70° to +70°
3	0°	-90°	120	0	-65° to +70°
4	0°	+90°	0	720	-150° to +150°
5	0°	-90°	0	0	-115° to +115°
6	180°	0°	0	85	-300° to +300°

Table 5.1 Link parameters for IRB1400

It can be seen from Figure 5.4 that the tool coordinate system is similar to the tool coordinate system shown in Figure 2.5. This was purposely done so that the values

obtained from the IRB1400 will correlate with those in these calculations and thus have the same frame of reference to the base coordinate system.

As stated previously, the link parameters characterise the transformation between adjacent links. The four link parameters can each be expressed by a basic homogeneous rotation-translation matrix. These are then multiplied to produce the general A matrix, ${}^{n-1}A_n$, which is also known as the D-H transformation matrix for adjacent coordinate systems. The A matrix describes the combined effect of these four link parameters in one transformation matrix. The general A matrix is shown in following equation.

$$
\begin{aligned}
{}^{n-1}A_{\mathrm{n}} &= Rot(z,\theta).Trans(0,0,d).Trans(a,0,0).Rot(x,\alpha) \\
&= Rot(z,\theta).Trans(a,0,d).Rot(x,\alpha) \\
&= \begin{bmatrix} \cos(\theta_{\mathrm{n}}) & -\sin(\theta_{\mathrm{n}}) & 0 & 0 \\ \sin(\theta_{\mathrm{n}}) & \cos(\theta_{\mathrm{n}}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha_n) & -\sin(\alpha_n) & 0 \\ 0 & \sin(\alpha_n) & \cos(\alpha_n) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos(\theta_n) & -\cos(\alpha_n).\sin(\theta_n) & \sin(\alpha_n).\sin(\theta_n) & a_n.\cos(\theta_n) \\ \sin(\theta_n) & \cos(\alpha_n).\cos(\theta_n) & -\sin(\alpha_n).\cos(\theta_n) & a_n.\sin(\theta_n) \\ 0 & \sin(\alpha_n) & \cos(\alpha_n) & d_{\mathrm{n}} \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

The A matrix relates one link coordinate system to another. For example, ${}^{0}A_1$ relates coordinate system 1 to coordinate system 0. ${}^{0}A_6$ relates coordinate frame 6 to coordinate frame 0. This is known as the transformation matrix of the manipulator because it actually relates the tool centre point (TCP) of the manipulator (coordinate system 6) with the base of the robot (coordinate system 0). By substituting the four

link parameters for each link into the general A matrix, ${}^{n-1}A_n$ specific A matrices are derived as shown below.

$$ {}^{0}A_1 = \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & a_1.\cos\theta_1 \\ \sin\theta_1 & 0 & \cos\theta_1 & a_1.\sin\theta_1 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{1}A_2 = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & a_2.\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & a_2.\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{2}A_3 = \begin{bmatrix} \cos\theta_3 & 0 & -\sin\theta_3 & a_3.\cos\theta_3 \\ \sin\theta_3 & 0 & \cos\theta_3 & a_3.\sin\theta_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{3}A_4 = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 & 0 \\ \sin\theta_4 & 0 & -\cos\theta_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

$$ {}^{4}A_5 = \begin{bmatrix} \cos\theta_5 & 0 & -\sin\theta_5 & 0 \\ \sin\theta_5 & 0 & \cos\theta_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{5}A_6 = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ \sin\theta_6 & \cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The transformation matrix, ${}^{R}T_H$, which is also called the arm matrix describing the tool centre point position and orientation with respect to the base coordinate system, needs to be calculated by the chain product of the A matrices. It is given by:

$$ \begin{aligned} {}^{R}T_H &= T_1T_2 = {}^{0}A_6 = {}^{0}A_1.{}^{1}A_2.{}^{2}A_3.{}^{3}A_4.{}^{4}A_5.{}^{5}A_6 \\ &= \begin{bmatrix} {}^{0}x_6 & {}^{0}y_6 & {}^{0}z_6 & {}^{0}p_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} {}^{0}R_6 & {}^{0}p_6 \end{bmatrix} \\ &= \begin{bmatrix} n & s & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} $$

where

[${}^{0}x_6$, ${}^{0}y_6$, ${}^{0}z_6$] is the orientation matrix of the end effector with respect to the base coordinate system and is the upper left 3 x 3 matrix of ${}^{R}T_H$.

${}^0R_6$ is a short form for describing the orientation matrix of the end effector with respect to the base coordinate system and is the upper left 3 x 3 matrix of ${}^RT_H$.

${}^0p_6$ or p is the position vector that points from the origin of the base coordinate system to the origin of the end effector.

n is the normal vector of the end effector.

s is the sliding (or orientation) vector of the end effector and points in the direction of the fingers of the end effector.

a is the approach vector of the end effector.

It can be seen in the above equation that ${}^0p_6$ = T1.T2. This is done to make the hand multiplication simpler by first multiplying $T_1 = {}^0A_1.\ {}^1A_2.\ {}^2A_3$ and $T_2 = {}^3A_4.\ {}^4A_5.\ {}^5A_6$ and then finding the arm matrix ${}^RT_H$. The calculations for T_1 and T_2 follows (note that $\cos\theta_n = c_n$ and $\sin\theta_n = s_n$).

$$T_1 = {}^0A_1.{}^1A_2.{}^2A_3$$

$$= \begin{bmatrix} c_1 & 0 & -s_1 & a_1.c_1 \\ s_1 & 0 & c_1 & a_1.s_1 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_2 & -s_2 & 0 & a_2.c_2 \\ s_2 & c_2 & 0 & a_2.s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_3 & 0 & -s_3 & a_3.c_3 \\ s_3 & 0 & c_3 & a_3.s_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_1 & 0 & -s_1 & a_1.c_1 \\ s_1 & 0 & c_1 & a_1.s_1 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_2.c_3 - s_2.s_3 & 0 & -c_2.s_3 - s_2.c_3 & a_3.c_2.c_3 - a_3.s_2.s_3 + a_2.c_2 \\ s_2.c_3 + c_2.s_3 & 0 & -s_2.s_3 + c_2.c_3 & a_3.s_2.c_3 - a_3.c_2.s_3 + a_2.s_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_1(c_2.c_3 - s_2.s_3) & s_1 & -c_1(c_2.s_3 + s_2.c_3) & c_1(a_3.c_2.c_3 - a_3.s_2.s_3 + a_2.c_2 + a_1) \\ s_1(c_2.c_3 - s_2.s_3) & -c_1 & -s_1(c_2.s_3 + s_2.c_3) & s_1(a_3.c_2.c_3 - a_3.s_2.s_3 + a_2.c_2 + a_1) \\ -(s_2.c_3 + c_2.s_3) & 0 & s_2.s_3 - c_2.c_3 & -(a_3.s_2.c_3 + a_3.c_2.s_3 + a_2.s_2) + d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2 = {}^3A_4 . {}^4A_5 . {}^5A_6$$

$$= \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5 & 0 & -s_5 & 0 \\ s_5 & 0 & c_5 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_4 & 0 & s_4 & 0 \\ s_4 & 0 & -c_4 & 0 \\ 0 & 1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_5.c_6 & -c_5.s_6 & -s_5 & -d_6.s_5 \\ s_5.c_6 & -s_5.s_6 & c_5 & -d_6.c_5 \\ -s_6 & -c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c_4.c_5.c_6 - s_4.s_6 & -(c_4.c_5.s_6 + s_4.c_6) & -c_4.s_5 & -d_6.c_4.s_5 \\ s_5.c_5.c_6 + c_4.s_6 & -c_5.s_4.s_6 + c_4.c_6 & -s_4.s_5 & -d_6.s_4.s_5 \\ s_5.c_6 & -s_5.s_6 & c_5 & d_6.c_5 + d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The final arm matrix is given below with the equations for each of the matrix elements. These equations will be used to calculate the joint solution for a given set of joint angles. The θ_n angle link parameter must be added together with the given joint angle to give the correct solution (these non-zero θ parameters are actually offset angles).

$$
{}^{R}T_{H} = T_1 T_2
$$

$$
= \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

where

$$
n_x = c_1(c_2.c_3 - s_2.s_3)(c_4.c_5.c_6 - s_4.s_6) + s_1(s_4.c_5.c_6 + c_4.s_6) - c_1.s_5.c_6(c_2.s_3 + s_2.c_3)
$$

$$
n_y = s_1(c_2.c_3 - s_2.s_3)(c_4.c_5.c_6 - s_4.s_6) - c_1(s_4.c_5.c_6 + c_4.s_6) - s_1.s_5.c_6(c_2.s_3 + s_2.c_3)
$$

$$
n_z = -(s_2.c_3 + c_2.s_3)(c_4.c_5.c_6 - s_4.s_6) + s_5.c_6(s_2.s_3 - c_2.c_3)
$$

$$
s_x = -c_1(c_2.c_3 - s_2.s_3)(c_4.c_5.s_6 + s_4.c_6) - s_1(s_4.c_5.s_6 - c_4.c_6) + c_1.s_5.s_6(c_2.s_3 + s_2.c_3)
$$

$$
s_y = -s_1(c_2.c_3 - s_2.s_3)(c_4.c_5.s_6 + s_4.c_6) + c_1(s_4.c_5.s_6 - c_4.c_6) + s_1.s_5.s_6(c_2.s_3 + s_2.c_3)
$$

$$
s_z = (s_2.c_3 + c_2.s_3)(c_4.c_5.s_6 + s_4.c_6) - s_5.s_6(s_2.s_3 - c_2.c_3)
$$

$$
a_x = -c_1.c_4.s_5(c_2.c_3 - s_2.s_3) - s_1.s_4.s_5 - c_1.c_5(c_2.s_3 + s_2.c_3)
$$

$$
a_y = -s_1.c_4.s_5(c_2.c_3 - s_2.s_3) + c_1.s_4.s_5 - s_1.c_5(c_2.s_3 + s_2.c_3)
$$

$$
a_z = c_4.s_5(s_2.c_3 + c_2.s_3) + c_5(s_2.s_3 - c_2.c_3)
$$

$$
\begin{aligned}
p_x = {} & -c_1.c_4.s_5.d_6(c_2.c_3 - s_2.s_3) - s_1.s_4.s_5.d_6 \\
& - c_1(c_2.s_3 + s_2.c_3)(d_6.c_5 + d_4) + c_1(a_3.c_2.c_3 - a_3.s_2.s_3 + a_2.c_1.c_2 + a_1)
\end{aligned}
$$

$$
\begin{aligned}
p_y = {} & -s_1.c_4.s_5.d_6(c_2.c_3 - s_2.s_3) + c_1.s_4.s_5.d_6 \\
& - s_1(c_2.s_3 + s_2.c_3)(d_6.c_5 + d_4) + a_3.s_1.c_2.c_3 - a_3.s_1.s_2.s_3 + a_2.s_1.c_2 + a_1.s_1
\end{aligned}
$$

$$
p_z = c_4.s_5.d_6(s_2.c_3 + c_2.s_3) + (s_2.s_3 - c_2.c_3)(d_6.c_5 + d_4) - (a_3.s_2.c_3 + a_3.c_2.s_3 + a_2.s_2) + d_1
$$

The calculations were first tested by making use of a Visual C++ application and the IRB1400 manipulator. The test data is given in Table 5.2(a) and Table 5.2(b). The manipulator was moved to a certain position and these values were then used as input values to the test application. The results were then compared to that given by the robot. Initially these values were completely incorrect (T1 – T3). It was noticed that if only the orientation of the manipulator was changed and the position is held at zero, then the values correlated (for T4 – T6). Then for T7 – T9 each of the position axis was tested individually. T7 and T9 gave the correct results.

After evaluating more test points with joint 2 at zero, it was observed that the action of the manipulator is such that when joint 2 angle increases by ϕ degrees, it is as if the joint 3 angle decreases by the same angle ϕ (although the actual value did not change). This is done to keep the elbow link at the same angle with respect to the base plane unless joint 3 angle changes. Thus, a slight change was made to the calculations that subtract the given joint 2 angle from the given joint 3 angle before the calculations are done. This proved to solve this problem.

	T1	T2	T3	T4	T5	T6	T7	T8	T9
θ_1	-71.2	92.2	114.6	0	0	0	25.0	0	0
θ_2	61.9	15.1	-13.4	0	0	0	0	-20.0	0
θ_3	42.2	-36.2	56.6	0	0	0	0	0	25.0
θ_4	-47.2	30.6	-150.0	75.0	33.4	33.8	33.8	33.8	33.8
θ_5	-29.8	-42.2	-81.2	52.0	-31.8	-95.7	-95.7	-95.7	-95.7
θ_6	-17.2	-68.6	-68.6	-126.4	-264.7	32.7	32.7	32.7	32.7
X	468.8	-3.2	-226.9	922.4	942.3	861.7	800.8	656.3	875.5
Y	-1284.8	839.0	395.5	65.1	-24.2	-46.7	321.5	-47.6	-47.8
Z	335.5	1653.3	472.6	1177.7	1232.6	1265.4	1265.4	1229.2	947.0
Q_1	.36793	.90333	.04770	.63001	.46077	.92099	.84404	.92111	.92688
Q_2	-.02716	-.06339	-.11556	-.58078	.51303	.26543	.28671	.26536	.31435
Q_3	.81570	.20948	-.88656	.51474	.12825	-.12753	-.06719	.12781	.07429
Q_4	.25507	.36889	.44541	.02894	.71278	.25507	.44820	.25458	.19120

Table 5.2(a) Forward kinematic test data (T1-T9)

	T10	T11	T12	T13	T14	T15	T16	T17
θ_1	34.0	34.0	-45.8	-127.2	113.0	113.0	0	0
θ_2	0	0	0	0	0	-41.8	-30.6	45.7
θ_3	25.0	-9.9	26.6	69.8	23.7	23.7	0	0
θ_4	33.8	33.8	-44.1	41.6	34.7	34.7	0	0
θ_5	-95.7	-95.7	17.0	-114.7	-42.1	-42.1	0	0
θ_6	32.7	32.7	-79.0	-244.8	-167.3	-167.3	0	0
X	752.1	704.7	624.4	-375.1	-335.0	-178.5	649.4	1384.4
Y	450.6	418.6	-665.9	-410.1	872.8	504.1	0	.1
Z	947.0	1385.0	808.2	493.9	912.4	759.5	1111.4	1014.3
q_1	.83029	.78584	.04493	.83128	.71700	.71723	.70712	.70717
q_2	.27883	.29233	.51804	.18234	-.58206	-.58201	-.00017	-.00017
q_3	.16309	-.12704	-.73428	-.39197	-.24210	-.24148	.70708	.70705
q_4	.45417	.52998	.43639	-.34941	-.29748	-.29759	-.00012	-.00011

Table 5.2(b) Forward kinematic test data (T10-T17)

5.2.3.2. Kinematics implementation

The kinematics solution was implemented using a class called Kinematics.class. It was written so that if the inverse kinematics solution is done it can be included in it. The class provides a function to calculate the direct kinematics and other functions to return the results in the form of the 3D vector (x,y,z in millimeters) and the quarternion values (q1 – q4).

The quarternion values are four values for representing the rotational matrix of the tool coordinate system angles with respect to the base coordinate system, and is more concise than the matrix (ABB, 1998b). The main reason why these were used instead of angles (as in roll, pitch and yaw) was because when the IRB1400 is programmed, it takes the 3D vector and the quarternion values as parameters to specify the tool centre point values with a unique robot configuration.

The final arm matrix (${}^{R}T_{H}$) describing the relationship between the base and the tool is given below with the rotational matrix shown to the right of it:

$$
\begin{aligned}
{}^{R}T_{H} &= T_1 T_2 \\
&= \begin{bmatrix} n_x & s_x & a_x & p_x \\ n_y & s_y & a_y & p_y \\ n_z & s_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\qquad
Rotational\ mattrix = \begin{bmatrix} n_x & s_x & a_x \\ n_y & s_y & a_y \\ n_z & s_z & a_z \end{bmatrix}
$$

The rotational matrix is made up of 3 3D vectors. Vector *n*, *s* and *a* correspond to the rotated x, y and z axis respectively of the tool coordinate system with respect to the base coordinate system. The value of n_x will then be the x component of the x

vector. The quarternion values can be calculated using this matrix values as shown in the following equations (ABB, 1998b):

$$q1 = \frac{\sqrt{n_x + s_y + a_z + 1}}{2}$$

$$q2 = \frac{\sqrt{n_x - s_y - a_z + 1}}{2} \qquad sign\, q2 = sign\,(s_z - a_y)$$

$$q3 = \frac{\sqrt{s_y - n_x - a_z + 1}}{2} \qquad sign\, q3 = sign\,(a_x - n_z)$$

$$q4 = \frac{\sqrt{a_z - n_x - s_y + 1}}{2} \qquad sign\, q4 = sign\,(n_y - s_x)$$

These values together with the 3D position vector of the tool centre point are displayed on the GUI on the applet for information purposes. The code for the kinematics class is shown below.

```
public class Kinematics {
 //Variable initialization
 …
 //Link parameters
 protected double offset[] = {0,-90,0,0,0,180};
 protected double Dparam[] = {475,0,0,720,0,85};
 protected double Aparam[] = {150,600,120,0,0,0};

 //Overridden function for home position
 public void CalcFK() {CalcFK(0,0,0,0,0,0);}

 //Calculates the forward kinematics for the IRB1400
 public void CalcFK(double a0,double a1,double a2,
                    double a3,double a4, double a5){
  //Calculate often used angles for use in equations
  // and convert to radians
  sn[0] = Math.sin((a0+offset[0])*Math.PI/180);
  cs[0] = Math.cos((a0+offset[0])*Math.PI/180);
  sn[1] = Math.sin((a1+offset[1])*Math.PI/180);
  cs[1] = Math.cos((a1+offset[1])*Math.PI/180);
  //Correct angle for correct robot operation
  sn[2] = Math.sin((a2-a1+offset[2])*Math.PI/180);
  cs[2] = Math.cos((a2-a1+offset[2])*Math.PI/180);
  …
  cs[5] = Math.cos((a5+offset[5])*Math.PI/180);

  //Calculate arm matrix values
```

```
  n[0] = cs[0]*(cs[1]*cs[2]-sn[1]*sn[2])*
    (cs[3]*cs[4]*cs[5]-sn[3]*sn[5]) + sn[0]*
    (sn[3]*cs[4]*cs[5]+cs[3]*sn[5]) - cs[0]*cs[5]*
    (cs[1]*sn[2]+sn[1]*cs[2])*sn[4];
  n[1] = sn[0]*(cs[1]*cs[2]-sn[1]*sn[2])*
    (cs[3]*cs[4]*cs[5]-sn[3]*sn[5]) - cs[0]*
    (sn[3]*cs[4]*cs[5]+cs[3]*sn[5]) - sn[0]*sn[4]*cs[5]*
    (cs[1]*sn[2]+sn[1]*cs[2]);
  n[2] = -(sn[1]*cs[2]+cs[1]*sn[2])*(cs[3]*cs[4]*
    cs[5]-sn[3]*sn[5]) + sn[4]*cs[5]*
    (sn[1]*sn[2]-cs[1]*cs[2]);
  s[0] = -cs[0]*(cs[1]*cs[2]-sn[1]*sn[2])*
    (cs[3]*cs[4]*sn[5]+sn[3]*cs[5]) - sn[0]*(cs[4]*
    sn[3]*sn[5]-cs[3]*cs[5]) + cs[0]*sn[4]*sn[5]*
    (cs[1]*sn[2]+sn[1]*cs[2]);
  …
  p[2] = (sn[1]*cs[2]+cs[1]*sn[2])*Dparam[5]*cs[3]*
    sn[4] + (sn[1]*sn[2]-cs[1]*cs[2])*(Dparam[5]*
    cs[4]+Dparam[3]) - (Aparam[2]*sn[1]*cs[2]+Aparam[2]*
    cs[1]*sn[2]+Aparam[1]*sn[1]) + Dparam[0];

  //Calculate quarternions and their signs
  q[0] = Math.sqrt(n[0]+s[1]+a[2]+1)/2;
  q[1] = Math.abs((Math.sqrt(n[0]-s[1]-a[2]+1))/2);
  q[2] = Math.abs((Math.sqrt(s[1]-n[0]-a[2]+1))/2);
  q[3] = Math.abs((Math.sqrt(a[2]-n[0]-s[1]+1))/2);

  if((s[2]-a[1])<0) q[1]*=(-1);
  if((a[0]-n[2])<0) q[2]*=(-1);
  if((n[1]-s[0])<0) q[3]*=(-1);
 }

 //Get the XYZ values of the direct kinematics
 public double[] GetFK_XYZ(){
  double[] XYZ = new double[3];
  for(int i=0;i<3;i++)
   XYZ[i] = p[i];
  return XYZ;
 }

 //Get the quarternion values of the direct kinematics
 public double[] GetFK_Quarternion(){
  double[] Quart = new double[4];
  for(int i=0;i<4;i++)
   Quart[i] = q[i];
  return Quart;
 }
}
```

In the calculation section it can be seen that the axis 3 angle correction is done to compensate for the fact that the IRB1400 does not function exactly according to the hierarchical structure as explained in the robot control section.

5.2.4. Communication

The communication section is one of the primary tasks of the applet because it provides a mechanism to link to the robot server application to send and retrieve robot data from it. This section consists of a number of classes to realise this goal. Figure 5.5 gives the communications architecture of the Java applet.

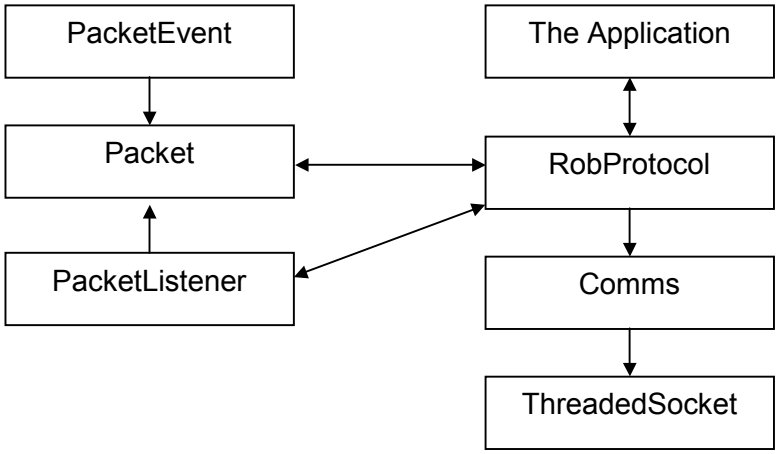


Figure 5.5 Communications architecture of applet

The communications are implemented using a layered approach were each base class has its own specific functionality. This was done so that some classes could be used for other applications as well, including the robot server application. Three classes have been created to handle the communication and another three are used to handle messages to and from the applet.

Once data is received from the server application, it must feed this data to the upper application layers in an appropriate manner. It does this by using MsgPacket objects as well as PacketListener objects. The MsgPacket object encapsulates the received

data. The PacketEvent is an interface that generate events for objects that are registered for the specific event.

Java provides a mechanism called interfaces that “implements the observer pattern” according to Venners (1998) or, in other words, to generate custom events. The observer design pattern is a common design pattern that are often encountered in object-oriented software.

The **RobProtocol** class implements the PacketListener interface and it is then registered as a listener (or observer) to events generated by the Packet class. Once data is received by the base class (Comms.class), a new MsgPacket is generated containing this data and an event is fired telling the RobProtocol object that a new packet has arrived that it must process. The individual classes together with code examples will now be discussed.

The **ThreadedSocket** class is the communications abstract base class and the same base class that is used by the Java applet and the robot server application. It provides:

- Threading capabilities so that the class can run in the background and therefore not block other sections of the code;
- Abstract (virtual) functions for sending and receiving data packets over the network;
- Destructor code to ensure proper termination of network connections; and
- A storage facility for storing data related to communication, e.g. port number, URLs, socket references, etc.

The following code is a portion of the code from ThreadedSocket.class:

```
public abstract class ThreadSocket extends Thread {
  protected DataInputStream input;
  protected DataOutputStream output;
  protected Socket mainsocket;
  protected int port;
  protected String url;

  public ThreadSocket(String url, int port){
    this.port = port;
    this.url = url;
  }

  public abstract void listen();
  public abstract void send(String str);

  public void run(){
    listen();
    try {close();}
    catch(IOException ioe) {}
  }

  protected void close() throws IOException {
    if(mainsocket!=null){
      mainsocket.close();
      mainsocket = null;
      stop();//stop thread
    }
  }}
```

The **Comms** class is the main communications class of the applet. It performs the following tasks:

- Provides base class functionality to the application layer;
- Continually listens for new messages from the server application by implementing the listen function in the ThreadedSocket abstract base class;
- Provides the ability to send messages to the server application; and
- Provides a means to cleanly exit once communication is terminated by either the client or the server.

The following is a section of the Comms class:

```
public class Comms extends ThreadSocket {
  //variable initialization
  …
  public Comms(String url, int port, mainApp App) {
    super(url, port);
    theApp = App;
    //Add packet object to enable callback functions
    // to packetlisteners
    packet = new Packet();
    try{socket = new Socket(url, port);
      setSocket(socket);
      //If successfull, get the streams
      input = new DataInputStream(
                        getSocket().getInputStream());
      output = new DataOutputStream(
                        getSocket().getOutputStream());
      theApp.taOutWin.append("Socket created!");
      //start base class thread
      super.start();}
      catch(IOException ioe){
        try{close();}
        catch(IOException ie){}
        theApp.taOutWin.append(
          "Socket creation failed. Socket has been
          closed.");}
      }

     /* Main communications function that will contain
      * the communications loop to listen for new data.*/
     public void listen(){
     while(true){
       try{str = input.readUTF();}
       catch(IOException ie){
         theApp.taOutWin.append(
           "IO exception in writing to server");
         break;//break out of while loop
       }
      MsgPacket p = new MsgPacket(str);
      packet.PacketArrived(p);  //'call' event handler
    }
    theApp.taOutWin.append("Exited from listen()");
    theApp.disconnectFlag = true;
  }

  /* Main function to send data to server
     Data is sent as a string of characters */
  public void send(String str){
    try{output.writeUTF(str);}
    catch(IOException ie){}
  }
```

The **RobProtocol** class is the main robot protocol class that is used to send and receive high level message packet from the robot server application. It specifically contains the function that gets called to indicate that a new data packet has been

received. The function then handles the function in a appropriate manner. A class called ConnectInfo is used to store information related to the socket connection. A section of the RobProtocol code follows:

```
/* Include these functions because its
 * an abstract function in the base class.*/
public void listen() {super.listen();   }
public void send(String str){ super.send(str); }
/* Event handling routine This function is called
   whenever new data is received at the socket. */
public void NewPacketArrived(
                  PacketEvent e, MsgPacket packet){
  int type = packet.getMsgType();
  //DO PROCESSING OF INPUT PACKET
  switch(type) {
    //Error in format of new message
    case MsgPacket.PACKET_FORMAT_ERROR:
    theApp.taOutWin.append("Packet format error!");
    break;
  case ConnectInfo.C_NEWPOSITION:
  case ConnectInfo.I_ROBPOSITION:
    theApp.taOutWin.append("New robot position");
    float temp[] = new float[6];
    if(packet.getNewPosition(temp)){
      //Set robot model with new data
      theApp.rob.setAxisFromAngles(temp);}
    else
     theApp.taOutWin.append("New position: Bad format ");
     break;
  case ConnectInfo.I_ROBSTATUS:
    boolean robstatus[] = new boolean[3];
    theApp.taOutWin.append("New robot status");
    if(packet.getNewRobotStatus(robstatus)){
      theApp.cInfo.setRobConnected(robstatus[0]);
      theApp.cInfo.setRobSimulate(robstatus[1]);
      theApp.cInfo.setRobReady(robstatus[2]);}
    else
      theApp.taOutWin.append("Robot status: Bad format");
    break;
  case ConnectInfo.I_USERMODE:
    theApp.taOutWin.append("New user mode");
    break;
  case ConnectInfo.C_SETCONTROLUSER:
   theApp.cInfo.setCurrentMode(ConnectInfo.MODE_CONTROL);
    theApp.taOutWin.append("Set control user");
    break;
  case ConnectInfo.C_SETMONITORUSER:
   theApp.cInfo.setCurrentMode(ConnectInfo.MODE_MONITOR);
    theApp.taOutWin.append("Set monitor user");
    break;
  case ConnectInfo.F_CONTROLUSEREXISTS:
    theApp.taOutWin.append("Control user already exists -
                           remain in monitor mode");
```

```
    break;
  case ConnectInfo.F_NOTVALIDJOINTANGLES:
    theApp.taOutWin.append("Fault: Not valid joint
                            angles!");  break;
  case ConnectInfo.F_NOTVALIDUSERNAME:
    theApp.taOutWin.append("Not valid password or
           username - retype and submit mode change");
    break;
  case ConnectInfo.F_ROBOTERROR:
    theApp.taOutWin.append("Fault: Robot error!");
    break;
  default:
    theApp.taOutWin.append("Unrecocognized message type
                            received");
    break;
  }
  theApp.UpdateGUI();
}

//Send a message packet object to server application
public boolean SendNewPacket(MsgPacket p){
  send(p.getStringFromPacket());
  return true;
}
```

The declaration of the RobProtocol class, that is shown next, clearly shows that it acts as an observer for PacketEvents:

```
public class RobProtocol extends Comms
                      implements PacketListener {
  //Constructor
  public RobProtocol(String url, int port, mainApp App){
    super(url, port, App);
    //ensure that event handling mechanism is in place
    super.packet.addPacketListener(this);
  }}
```

The PacketListener interface links the object generating the event to the object(s) that is registered with the event by specifying the function that must be called one an event is generated. The complete code can be seen below.

```
public interface PacketListener extends
                              java.util.EventListener {
  //This 'funcion prototype' includes the event type
  //class and the data packet class
  void NewPacketArrived(PacketEvent e, MsgPacket p);
```

```
}
```

The **PacketEvent** class declares a new event type that is a derived class of the EventObject base class. The class does not do anything but is necessary for the event-handling mechanism to perform correctly. The code is shown below.

```
public class PacketEvent extends java.util.EventObject{
  public PacketEvent(Packet source)
    {super(source);}
}
```

The **Packet** class is the class that is used to carry the data from the event generator to the event listeners. It does this by creating a linked list (an array of objects) of event listeners. Event listeners are added by calling the addPacketListener function. When an event of this type is generated, the class will automatically call the event handler function defined for each event listener. A section of the Packet class is shown next.

```
public class Packet {
  …
  //Declare a list for all observer classes
  private Vector packetListeners = new Vector();
  //Function called when event must be fired
  public void PacketArrived(MsgPacket p) {
    firePacketArrived(p);
  }
  //Add/remove observers to/from list
  public synchronized void addPacketListener(
                                PacketListener l){
    if(packetListeners.contains(l))
      return;
    packetListeners.addElement(l);

  }
  public synchronized void removePacketListener(
                              PacketListener l){
    packetListeners.removeElement(l);
  }
  //Notifies all observers that event has occured
  private void firePacketArrived(MsgPacket p){
    Vector tl;
    synchronized(this) {
      tl = (Vector)packetListeners.clone();}
```

```
    int size = tl.size();
    if(size==0) return;
    PacketEvent event = new PacketEvent(this);
    for(int i=0;i<size;++i){
      PacketListener listener =
                 (PacketListener)tl.elementAt(i);
      listener.NewPacketArrived(event, p);
    }
 }}
```

From the above code it can be seen that the data that is passed from event generator to event observer(s) is in the form of a high level MsgPacket class. This class encapsulates the received data and data that must be sent back to the server. The message packets mainly contain the type of message, the message data and functions to process the data easily. It also does error handling to check for valid message data. A section of the class is shown below.

```
public class MsgPacket{
  //USED FOR INCOMING MESSAGES
  public MsgPacket(String s){
    msg = new String();
    processNewMsg(s);}

//USED TO PROCESS ALL INCOMING MESSAGES AND
// SORT THE MSGTYPE FROM THE MESSAGE
  private void processNewMsg(String s){
  int i;
  String str;
  //Check if msg is in the correct format
  if((i=s.indexOf(' ')) != -1){
    str = s.substring(0,i);//Type
    msgType = Integer.valueOf(str).intValue();
    msg = s.substring(i+1);}
  else{
    msgType = PACKET_FORMAT_ERROR;
    msg = "";}
  }

  public String getMsg(){ return msg; }
  public int getMsgType(){ return msgType; }
  public void setMsg(String s) { msg = s; }
  public void setMsgType(int i) { msgType = i; }

  /*Returns the set of new angles; @param1 An array in which
the angles must be placed; @return Flag to indicate if packet
format is bad; */
  public boolean getNewPosition(float angles[]){
    //FORMAT OF MSG: "val1 val2 val3 val4 val5 val6 "
    s = msg;
    for(int i=0;i<6;i++)
```

```
      …
  return true; //all is well
}

  public boolean getNewRobotStatus(boolean status[]){
    //FORMAT OF MSG: "val1 val2 val3 "
    s = msg;
    for(int i=0;i<temp.length;i++)
      …
  }

  public String getStringFromPacket()
  { return msgType + " " + msg + " ";}
}
```

5.2.5. User modes

User modes have been introduced into the application since it is possible for many users to connect to the robot server application simultaneously. It is, however, impractical for all connected users to be able to control the robot because of contradicting commands issued at random by the different users. Two user modes have been defined namely a monitor user mode and a control user mode. Provision is made for one control user and many monitor users up to a maximum of 20 users.

When the Java client applet starts up there are no connection to the server application and the user operates in an offline control user mode. The user is able to control and manipulate the VRML robot scene but has no access to the physical robot data.

Once the user has connected to the robot server application successfully he is made an online monitor user by default. In this mode the user has no control over the physical robot (or robot data) as well as the virtual robot in the VRML scene (although he may still perform VRML related rendering control, e.g. viewing the VRML robot from other angles or zooming in or out). This mode provides the ability

to monitor the physical robot while it is being controlled by the control user (if one exists) or the robot server application (that acts as a master controller). Immediately after the user is connected, the server sends the latest robot data to the user and his VRML scene and robot data is updated.

If an online monitor user wishes to become the controller, he may do so is he has the appropriate username and password. The username and password provides basic authentication and this security function may be extended to include encryption on data sent to and from users. Once the user applies to be a control user, he will be notified by the server application if his application was successful. If it is, he will become the single control user with full control of the physical robot and any changes he makes will be reflected in all monitor users' virtual robots. If his application is not successful, he will be notified of the reason, e.g. a control user already exists or the username or password is incorrect. He will thus remain in monitor mode but may apply to be control user as often as possible.

An online control user may relinquish control to other users wishing to become control users. This is done by pressing the "Toggle Mode" button. Once a user disconnects, whether he was in control or monitor mode, he will be placed in offline control mode. If a user leaves the page on which the applet resides or closes the browser for some reason, the connection will automatically be terminated. If he was the control user, the server application will detect the connection was terminated and will enable other users to connect as control users.

5.2.6. Applet security

Java applets are normally loaded into a specific area of memory on the remote user's machine known as a sandbox and has limited or no access to system resources outside of this sandbox. For example an applet does not have access to system (i.e. on the client machine) resources such as printing, file access or networking. The class loader controls much of the security and access rights that an applet has. In order for an applet to be able to use system resources, the code must be made trusted (Microsoft, 1998).

Making applet code trusted means that the code is not confined to the sandbox anymore. A few things need to happen for Java code to become trusted. The code has to be placed within a cabinet file (*.cab) and the CAB file needs to be digitally signed (with a certificate). When the CAB file is created it must also specify the permissions required by the classes in the CAB file. Normally there are 3 predefined levels of permissions namely high (i.e. no permissions at all), medium and low (i.e. permissions to all resources) permissions. Custom permissions (also known as fine grained permissions) may also be set up to grant access to specific resources. The certificate that the applet is signed with is normally one that is bought from a Certificate Authority such as VeriSign. For development purposes a test certificate may be used.

When an applet has been digitally signed and is loaded onto a client machine web browser, a message alert will show the client who the applet belongs to and what permissions are required by the applet classes before the applet executes so that the client may decide whether the applet must be run or not.

For the client applet used in this project it was necessary to sign and package it because no socket connections to a server may be made unless permission was given by the client. A test certificate was used to sign the applet and a cab file (client2.cab) was created containing all the class files. MS Visual J++ 6 (that was used for developing this project) provides the ability to do all the above from within the development environment. The HTML file that contains the link to the applet is used to specify the applet name and the CAB file that must be used. The HTML file is shown below.

```
<p>
<centre>
<embed src="irb1400v2.wrl" width="760" height="360">
<applet code="mainApp.class" width="760" height="120"
mayscript>
<PARAM NAME="cabbase" VALUE="Client2.CAB">
</applet>
</centre>
</p>
```

5.2.7. Conclusion

The Java applet consists of a number of classes that contribute to the main goal of the applet, which is the establishment of remote control of the physical robot and communicating with the robot server application. The main application class of the Java applet is called the mainApp class. It performs the following functions:

- Creation and initialization of main application (including global) objects and variables including GUI elements and VRML specific elements;
- Creating functions to set up and update GUI elements;
- Set up the application to run on a separate thread of execution;
- Ensures that proper action is taken when the socket connection is lost or terminated by the robot server application or some external cause; and

- Establishing and maintaining a connection with the VRML browser object.

It can be seen from the above functions that the mainApp class provides important application-wide functionality. The applet forms an integral part of the system and provides a user-friendly interface to monitor and control the physical robot.

5.3. Robot Server Application

The Robot Server Application is a stand-alone Java application that runs on the PC that directly controls the physical robot, also referred to as the host PC (see Figure 4.1). The server application performs the following tasks:

- Controls the physical robot by making use of RobComm and RAPlink proprietary software;
- Acts as a master controller of the robot;
- Acts as a server to clients requesting data and those controlling the robot;
- Manages connections with remote clients;

The application consists of a number of Java classes, some of which are almost identical to those used in the Java client applet. The application is split into different areas where each area is implemented using one or more classes. It must be noted that because the proprietary hardware and software to control the robot was not yet available when this software was being developed, only an explanation and recommendation of how the proprietary wares can be linked with the software written for this project (this will be discussed in the physical robot control section). The following sections will explain details of how the GUI, kinematics and communications architecture for the robot server application were implemented.

5.3.1. Graphical User Interface (GUI)

The GUI is the front-end of the application and provides an interface for the master controller to:

- Control the physical robot;
- Set robot status;
- View network and users status;
- View server parameters; and
- View robot status.

Figure 5.6 shows the GUI layout. Each of these GUI elements will now be explained briefly and their implementation will be discussed.

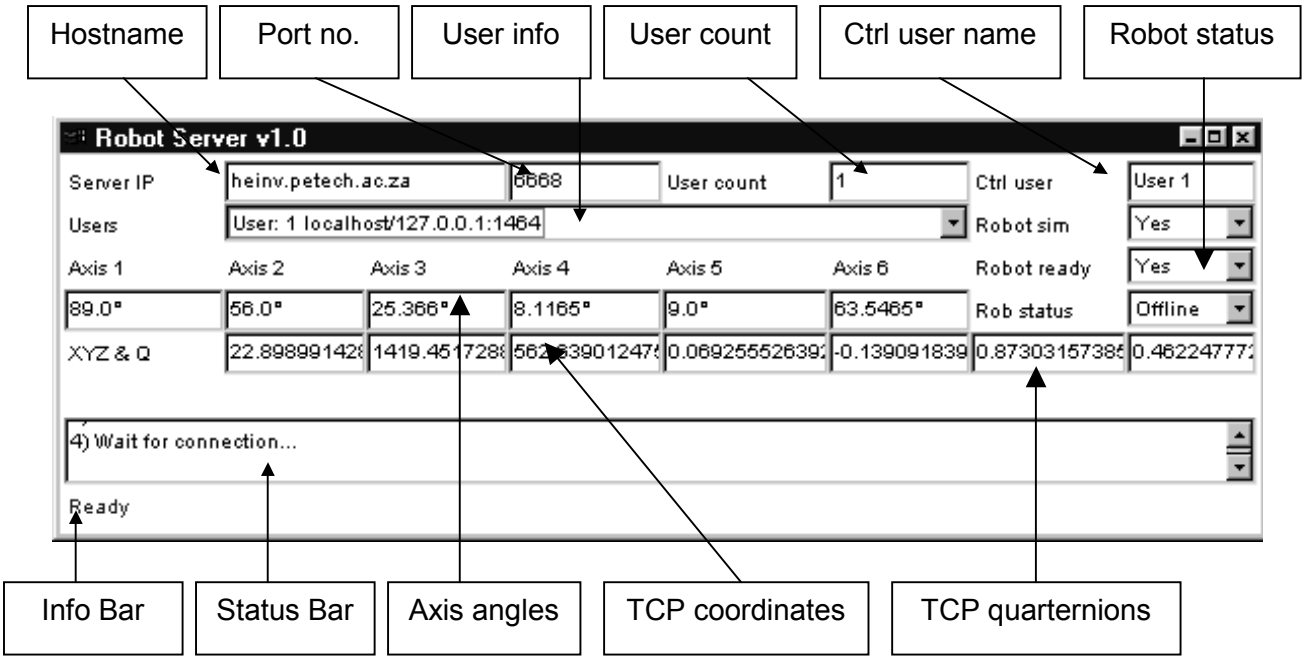


Figure 5.6 Robot server application GUI

- GUI elements

The server IP address is the hostname of the machine that is used to control the robot and that is running the robot application. The name “heinv” is internal to the petech.ac.za domain. The port that was used for establishing communications with the server was taken as an arbitrary value of 6668. This value is outside the area reserved for current protocols (e.g. FTP uses port 21 and HTTP uses port 80).

The user count shows the number of all users currently connected to the server. The user’s combobox shows the actual hostname of the connected users as well as the order in which they connected. It also shows the local port through which the specific user is connected to the server.

The control user specifies the user, if any, that is currently in control user mode. If no control user exists, this field will show “None”; otherwise it will display the user number correlating with the user entry in the users combobox.

The six axis angles of the robot is shown in six text fields. The coordinates of the tool centre point (TCP) is shown below the axis values. These coordinates include the positional cartesian coordinates (x, y, z) of the TCP in millimeters and the orientational quarternion values of the TCP with respect to the base coordinate system. All these values are changed as the control user (if any) or the master controller changes one or more of the robot axis angles.

A scrollable status bar at the bottom of the application window is used to give network and user related information, e.g. to indicate when a user connects or disconnects. An information bar below the status bar is intended to give more info

on text fields as the user (of the application) moves the mouse cursor over a text field.

- Implementation

The method of how Java typically handle GUI events, how the event model functions and how interfaces are created and used was discussed in the section on the client applet. In this section only the relevant changes to these will be mentioned and discussed.

Most of the GUI elements are defined inside the main application class called RobServer. They are also added to the application and updated by the addCompToScreen and UpdateGUI functions respectively.

There are two classes that handle events of the main application namely StatusBarHandler and MainHandler. StatusBarHandler is only used to display context sensitive information to the user as the user moves the mouse of text fields in the application window. The information is displayed in the information bar. MainHandler is responsible for updating the internal robot data and axis angles when these are changed by the master controller. It also sets a flag so that the main application thread can update all clients and the physical robot with the new values. Code examples of the above classes is shown below.

```
/*This class ensures that context sensitive info is
 displayed in the statusbar at the bottom of the frame*/
class StatusBarHandler extends MouseAdapter{
```

```
  private RobServer rs;
  public StatusBarHandler(Label lb, RobServer r){
    sb = lb; rs = r; }
  public void mouseEntered(MouseEvent e){
    //Set text in statusbar for each component that
    //the mouse moves over
    if(e.getSource()==rs.tfServerAddress)
      sb.setText("Server address");
    else if(e.getSource()==rs.tfServerPort)
      sb.setText("Server's port number");
    for(int i=0;i<rs.tfAngle.length;i++)
      if(e.getSource()==rs.tfAngle[i])
        sb.setText("Change axis " + (i+1)
          + " angle and press Enter");
    for(int i=0;i<rs.tfQ.length;i++)
      if(e.getSource()==rs.tfQ[i])
        sb.setText("Quarternion value #" + (i+1));
  }
  public void mouseExited(MouseEvent e)
    { sb.setText("Ready"); }
}

/*Main event handler of the robot server. Handles common GUI
events */
class MainHandler implements ActionListener, ItemListener
{ RobServer rs;
  public MainHandler(RobServer r) { rs = r;  }
  public void itemStateChanged(ItemEvent ie) {
    if(ie.getSource()==rs.chRobotReady)
      rs.robot.RobReady("Yes"==
        rs.chRobotReady.getSelectedItem()?true:false);
    else if(ie.getSource()==rs.chRobotSim)
      rs.robot.RobSimulate("Yes"==
        rs.chRobotSim.getSelectedItem()?true:false);
    else if(ie.getSource()==rs.chRobotStatus)
      rs.robot.RobConnected("Online"==
        rs.chRobotStatus.getSelectedItem()?true:false);
    rs.NewDataFromRobot = true;
    rs.UpdateGUI();
  }
  public void actionPerformed(ActionEvent e) {
    for(int i=0;i<rs.tfAngle.length;i++)
      if(e.getSource()==rs.tfAngle[i])
        rs.robot.ChangeAxisAngle(i+1,
          Float.valueOf(
          rs.tfAngle[i].getText()).floatValue());
    rs.NewDataFromRobot = true;
    rs.UpdateGUI();
}}
```

- Kinematics

The theory of kinematics was covered in Chapter 2 whereas the direct kinematical solution for the IRB1400 was done in Section 5.2.3. The implementation of the

kinematical calculations is exactly the same as that covered in the above-mentioned section, to the extent that the same class (Kinematics.class) was used by both the client applet and the robot server application. This is a good example of code reuse in object-oriented programming. Because of this reason, kinematics will not be covered for the robot server application.

- Server communications architecture

The server part of the robot server application is based on classes that implement the server, where the server can be defined as a section of the program that establishes, maintains and terminates network connections, which exist between the application running the server and remote clients. The architecture of the server communications is clearly shown by the class hierarchy below in Figure 5.7.

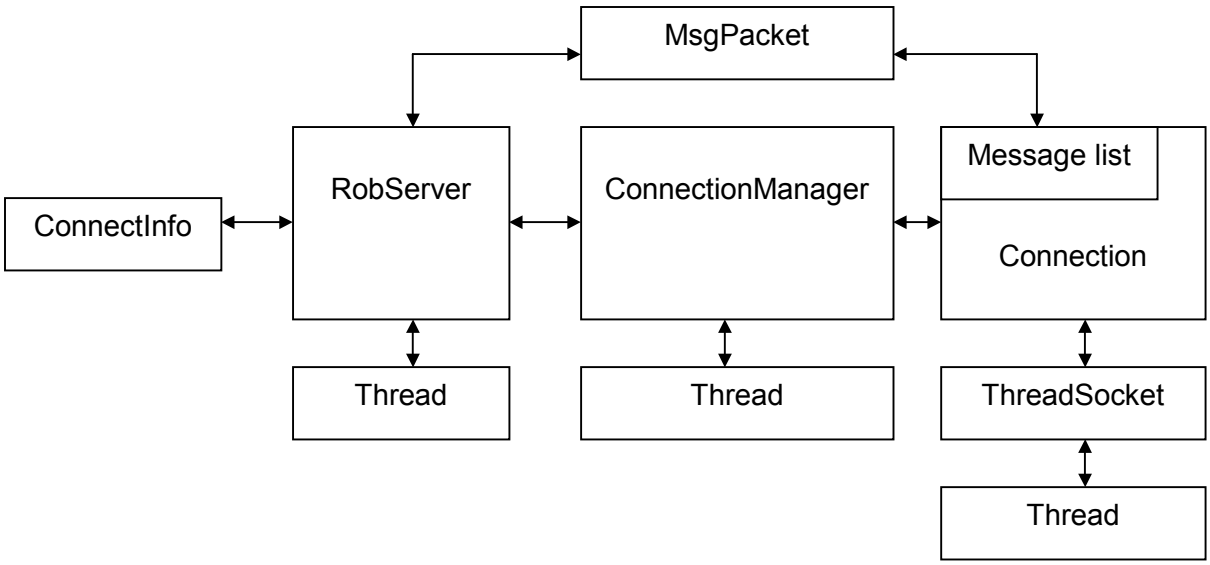


Figure 5.7 Communications architecture

There are three main classes forming the communications framework, i.e. RobServer class, ConnectionManager class and Connection class. The other classes only have supporting functions.

The data flow within the architecture differs significantly from the observer-listener event-based mechanism that was used by the client applet. The server application must be able to handle many client connections simultaneously as well as messages or requests from each connection. This task was broken down into three sections, each implemented by a specific class:

- Sending and collecting messages to and from one connection (Connection class);
- Establishing connections with clients wishing to connect to the server (ConnectionManager class); and
- Handling messages from all clients and maintaining all connections (RobServer class).

Each of the above classes will now be discussed with their supporting classes. Code examples will also be given.

The **Connection** class is used to maintain a connection with a remote client. It stores information regarding the client, e.g. whether the client is a control or monitor user. It also stores client information relating to the socket connection indirectly in the ThreadedSocket class. It continually listens for messages or requests from the client to which it is connected. Each of these messages are packaged in a MsgPacket class and these MsgPacket objects are placed inside a container class (similar to a linked list or an array of objects) for manipulation by the RobServer class.

The **ThreadedSocket** class is an abstract base class providing threading capabilities, among others, to each connection. It is very similar to the ThreadedSocket class used for the client applet.

The **MsgPacket** class, mentioned above, is the same one that was used for the client applet. It provides a structure that keeps a message together with its command code. It also provides functions to extract information from the message intelligently. Code examples of the Connection class is shown below.

```
public class MsgPacket{
  public MsgPacket(){msg = new String();}
  //USED FOR INCOMING MESSAGES
  public MsgPacket(String s) {
    msg = new String();
    processNewMsg(s);
  }
  //USED TO PROCESS ALL INCOMING MESSAGES AND
  // SORT THE MSGTYPE FROM THE MESSAGE
  private void processNewMsg(String s){
    //Check if msg is in the correct format
    if((i=s.indexOf(' ')) != -1){
      //Type is 1st string before space
      str = s.substring(0,i);
      msgType = Integer.valueOf(str).intValue();
      //Copy whatever is after 1st space to msg
      msg = s.substring(i+1);
    }
    else{
      msgType = PACKET_FORMAT_ERROR;
      msg = "";
    }
  }
  public String getMsg() { return msg; }
  public int getMsgType(){ return msgType; }
  public void setMsg(String s){ msg = s; }
  public void setMsgType(int i){ msgType = i; }
  /*Returns the set of new angles; @param1 An array in
  // which the angles must be placed; @return Flag to
  // indicate if packet format is bad*/
  public boolean getNewPosition(float angles[]){
    //FORMAT OF MSG: "val1 val2 val3 val4 val5 val6 "
    s = msg;
    for(int i=0;i<6;i++){
    if((cnt=s.indexOf(' ')) != -1){
      angles[i] = Float.valueOf(
        s.substring(0,cnt)).floatValue();
```

```
      s = s.substring(cnt+1);
    }
    else return false;//format incorrect
  }
  return true; //all is well
}
public boolean getNewRobotStatus(boolean status[]){
  //FORMAT OF MSG: "val1 val2 val3 "
  s = msg;
  for(int i=0;i<temp.length;i++){
    if((cnt=s.indexOf(' ')) != -1){
      temp[i] = Integer.valueOf(
        s.substring(0,cnt)).intValue();
      status[i] = (temp[i]==1);
      s = s.substring(cnt+1);
    }
    else return false; //format incorrect
  }
  return true; //all is well
}
public String getStringFromPacket(){
  return msgType + " " + msg + " ";}
public boolean getUNPW(String username[]){
  //FORMAT OF MSG: "username password "
  s = msg;
  for(int i=0;i<username.length;i++){
    if((cnt=s.indexOf(' ')) != -1){
      username[i] = new String(s.substring(0,cnt));
      s = s.substring(cnt+1);
    }
    else return false; //format incorrect
  }
  return true; //all is well
}}
```

The **ConnectionManager** class has the primary goal of establishing new connections and placing these connection objects in a container class for processing by the RobServer class. A ServerSocket object that is assigned a specific port number listens on that port for clients wishing to connect to the server. Once a connection request is received on that port, the ConnectionManager creates a new Connection class object, assigns it a unique ID number, makes sure the client connects as a monitor user initially and adds it to the client container. The ConnectionManager class is a subclass of class Thread in order for it to continually listen for new connection requests. Code from the class is shown below.

```
public class ConnectionManager extends Thread{
  /*Connection ID counter; An ID is inserted into each
  .connection object to make it easily identifiable*/
  protected int IDCounter = 1; //start at 1
  public ConnectionManager(Vector vc, ServerSocket ss,
    ExTextArea ta, Choice ch, RobServer rs){
    server = ss;
    users = vc;
    status = ta;
    chUsers = ch;
    theserver = rs;
    //Start thread
    start();
  }
  public void run(){
    while(true){
      status.append("Wait for connection...");
      socket = null;
      try{ socket = server.accept(); }
      catch(IOException ioe){
        status.append("Problem with accepting new
          connection");
      }
      if(socket!=null){ //check if socket exists
        //Always connect user as monitor
        connection = new Connection(socket,0,status);
        status.append("Connection with client created");
        connection.setConnectID(IDCounter++);
        //Add connection to container class
        synchronized(this){
          users.addElement(connection);
          chUsers.add("User: " +
            connection.getConnectID() +
            " " + connection.getURL() + ":" +
            connection.getPort());
          theserver.UpdateGUI();
}}}}}
```

The **RobServer** class is the main application class of the server application. It performs application level tasks of which communication with clients and physical robot control are the most important. The class runs on a separate thread of execution enabling it to function in parallel with other tasks. The communication tasks performed by the RobServer class will now be discussed making use of code examples.

Task 1) Connections that have been lost (either by a bad TCP/IP link or by the specific client terminating the connection)are removed. It does this by polling through a list of currently connected users and deleting the user entry from the list. The following code extract shows the implementation of this task.

```
//Create enumeration of copy of users to easily iterate
//through it
Enumeration UsersEnum = UsersConnected.elements();
//Temp connection variable for use with enum
Connection tempCon;
/*TASK 1 - Remove connections whose socket connection
  have been lost. a) Go through list of users, b) Remove
  connection from list; c) Remove connection from list of
  currently connected users*/
while(UsersEnum.hasMoreElements()){
  //Get temp connection variable
  tempCon = (Connection)UsersEnum.nextElement();
  if(tempCon.getSocket()==null){
    String s = new String();
  int j,k,id;
  for(int i=0;i<chClientsConnected.getItemCount();i++){
    //Extract the ID from the connection entry
    s = chClientsConnected.getItem(i);
    j = s.indexOf(' ');k = s.indexOf(' ', j+1);
    s = s.substring(j+1,k);
    id = Integer.valueOf(s).intValue();
    //Check if this is the right connection entry
    if(id==tempCon.getConnectID()){
      chClientsConnected.remove(i);
      taOutWin.append("Removed user: " +
      tempCon.getURL() +
      ":" + tempCon.getPort());
      if(tempCon.IsController()){
        Controller=false;
        cInfo.setControlUserName("None");
      }
    UpdateGUI();
    break;
  }}
//Remove connection object from list
UsersConnected.removeElement(tempCon);
}}
```

Task 2) The list of connections is checked to see whether or not a control user exists. If one exists, all messages and commands from this connection are processed and executed. If robot data is changed by the control user, a flag

indicates that all monitor users must be updated with the latest robot data. Any other requests by the control user are also processed, e.g. if the control user wishes to relinquish control of the robot so that another monitor user may able to become a control user. Following is an extract of the code implementation of this task.

```
/*TASK 2 - Receive input from controller, if any; a) Go
through list of users; b) Get controller, if any; c) Get msg
from controller; d) If new data, set flag and set new data; e)
Set flag indicates that new data must be sent to monitors;*/
//Reinitialize enumeration
UsersEnum = UsersConnected.elements();
while(UsersEnum.hasMoreElements()){
  //Get temp connection variable
  tempCon = (Connection)UsersEnum.nextElement();
  //Check to see if this connection is the controller
  if(tempCon.IsController()){
    if(tempCon.AnyNewMsg()){
      //Read and display new messages from contoller
      Enumeration enum =
        tempCon.NewInMessages.elements();
      MsgPacket tempMsg;
      while(enum.hasMoreElements()){
        tempMsg = (MsgPacket)enum.nextElement();
        MsgPacket mpacket = new MsgPacket();
        switch(tempMsg.getMsgType()){
          case ConnectInfo.C_NEWPOSITION:
            float Ang[] = new float[6];
            if(tempMsg.getNewPosition(Ang))
              robot.setAxisFromAngles(Ang);
              UpdateGUI();
            }
      //Indicate that new data has arrived and must be
      //sent to monitors
            NewDataFromController = true;
            break;
          case  ConnectInfo.I_ROBPOSITION:
            mpacket.setMsgType(ConnectInfo.I_ROBSTATUS);
            mpacket.setMsg((robot.IsRobConnected()?1:0) +
            " " +(robot.IsRobSimulated()?1:0) + " " +
            (robot.IsRobReady()?1:0));
            tempCon.send(mpacket.getStringFromPacket());
            break;
          case ConnectInfo.C_SETMONITORUSER:
            mpacket.setMsgType(
            ConnectInfo.C_SETMONITORUSER);
            mpacket.setMsg("");
            tempCon.send(mpacket.getStringFromPacket());
            tempCon.setController(false);
            Controller = false;
            cInfo.setControlUserName("None");
            UpdateGUI();
```

```
            break;
          default:
            taOutWin.append("Corrupt message from: " +
            tempCon.getURL() + ":" + tempCon.getPort());
            break;
        }
      }
      //Clear all messages from 'buffer'
      tempCon.NewInMessages.removeAllElements();
    }}}
```

Task 3) The list of connections is iterated and the messages from each connections message buffer is processed one by one. After each message is processed a response is immediately sent to the user, if a response is required by the message. The code implementation of this task is given below.

```
/*TASK 3 - Process messages from all users; send feedback; a)
Go through list of users; b) Respond to requests;*/
//Reinitialize enumeration
UsersEnum = UsersConnected.elements();
while(UsersEnum.hasMoreElements()){
  //Get temp connection variable
  tempCon = (Connection)UsersEnum.nextElement();
  if(tempCon.AnyNewMsg()){
    //Read and display new messages from contoller
    Enumeration enum = tempCon.NewInMessages.elements();
    MsgPacket tempMsg;
    while(enum.hasMoreElements()){
      tempMsg = (MsgPacket)enum.nextElement();
      MsgPacket mpacket = new MsgPacket();
      switch(tempMsg.getMsgType()){
        case ConnectInfo.C_SETCONTROLUSER:
          if(Controller==false){
            String user[] = new String[2];
            if(tempMsg.getUNPW(user)){
              if(user[0].equals(USERNAME) &&
                user[1].equals(PASSWORD)){
                mpacket.setMsgType(
                  ConnectInfo.C_SETCONTROLUSER);
                mpacket.setMsg("");
                tempCon.send(
                  mpacket.getStringFromPacket());
                tempCon.setController(true);
                Controller = true; //set global flag
                cInfo.setControlUserName("User " +
                  tempCon.getConnectID());
                UpdateGUI();
              }
            else{
    //Tell user that the username/password is incorrect
```

```
                mpacket.setMsgType(
                  ConnectInfo.F_NOTVALIDUSERNAME);
                mpacket.setMsg("");
                tempCon.send(
                  mpacket.getStringFromPacket());
              }
            }
          }
          else{
      //Tell user that controller already exists
            mpacket.setMsgType(
              ConnectInfo.F_CONTROLUSEREXISTS);
            mpacket.setMsg("");
            tempCon.send(mpacket.getStringFromPacket());
          }
          break;
        case ConnectInfo.I_ROBSTATUS:
          mpacket.setMsgType(ConnectInfo.I_ROBSTATUS);
          mpacket.setMsg((robot.IsRobConnected()?1:0) + "
            " + (robot.IsRobSimulated()?1:0) + " " +
            (robot.IsRobReady()?1:0));
          tempCon.send(mpacket.getStringFromPacket());
          break;
        case ConnectInfo.I_ROBPOSITION:
          mpacket.setMsgType(ConnectInfo.I_ROBPOSITION);
          mpacket.setMsg(robot.getAxisAngle(1) + " " +
            robot.getAxisAngle(2) + " " +
            robot.getAxisAngle(3) + " " +
            robot.getAxisAngle(4) + " " +
            robot.getAxisAngle(5) + " " +
            robot.getAxisAngle(6));
          tempCon.send(mpacket.getStringFromPacket());
          break;
        default:
          taOutWin.append("Corrupt message from: " +
            tempCon.getURL() + ":" + tempCon.getPort());
          break;
        }
      }
      //Clear all messages from 'buffer'
      tempCon.NewInMessages.removeAllElements();
    }}
```

Task 4) If either the control user or the master controller changed any robot data, all users are sent an update of these changes. The code implementation of this task is shown below.

```
/*TASK 4 - Send new robot info to all users; a) Go through
list of users; b) Send info only*/
//Reinitialize enumeration
UsersEnum = UsersConnected.elements();
```

```
while(UsersEnum.hasMoreElements()){
  //Get temp connection variable
  tempCon = (Connection)UsersEnum.nextElement();
  if((NewDataFromController && !tempCon.IsController())
    || NewDataFromRobot){
    //SEND NEW ROBOT STATUS
    MsgPacket mpacket = new MsgPacket();
    mpacket.setMsgType(ConnectInfo.I_ROBSTATUS);
    mpacket.setMsg((robot.IsRobConnected()?1:0) + " " +
      (robot.IsRobSimulated()?1:0) + " " +
      (robot.IsRobReady()?1:0));
    tempCon.send(mpacket.getStringFromPacket());
    //SEND NEW ROBOT POSITION
    mpacket.setMsgType(ConnectInfo.C_NEWPOSITION);
    mpacket.setMsg(robot.getAxisAngle(1) + " " +
      robot.getAxisAngle(2) + " " +
      robot.getAxisAngle(3) + " " +
      robot.getAxisAngle(4) + " " +
      robot.getAxisAngle(5) + " " +
      robot.getAxisAngle(6));
    tempCon.send(mpacket.getStringFromPacket());
  }}
//Clear flags only after all users have been updated
NewDataFromController = false;
NewDataFromRobot = false;
//Gives other threads a chance to CPU time as well
try{ AppThread.sleep(50); }
catch(InterruptedException ie){}
}
```

The **ConnectInfo** class is similar to the one used by the client applet. It contains command IDs, fault IDs and information regarding the control user.

5.3.2. Conclusion

Communication between the server and its connected users is one of the most important tasks that it has to manage. The method used to handle communications could have been one of the many. For example, with this system a connection oriented mechanism was used to maintain connections. The system could also have used a connectionless mechanism. By using a connection-oriented mechanism and maintaining the connection for the lifespan of the client applet, the overheads

associated with setting up a new socket connection for each data transmission was eliminated.

The method of passing messages from a client connection to the main server application class is different from the one used by the client applet. The former method uses container classes to store message packets while the latter uses the observer-listener mechanism. The container class method seems to be the preferred method as there are less overheads.

5.4. Physical Robot Control

Controlling the IRB1400 is another primary task of the robot server application and is the main reason for having a server. As was stated in Chapter 4, no hardware or software for controlling the robot was available when this project was done. The physical robot data was simulated as if a robot was connected. This section aims to explain how the physical robot hardware and software must be integrated with the developed system to provide true remote monitoring and control of an industrial robot. It will be assumed (for this section) that all the required hardware and software are available.

5.4.1. Required hardware

The hardware required by the controller is a proprietary Ethernet card (the DSQC 336) from ABB. The card plugs into a slot on the controller and has a standard 10BASE-T twisted pair connection. The card connects to an Ethernet hub (on the LAN) via twisted pair cable.

The controller PC should be connected to the same network (LAN) to which the controller is connected by means of a standard network card. Figure 5.8 shows the network outline.

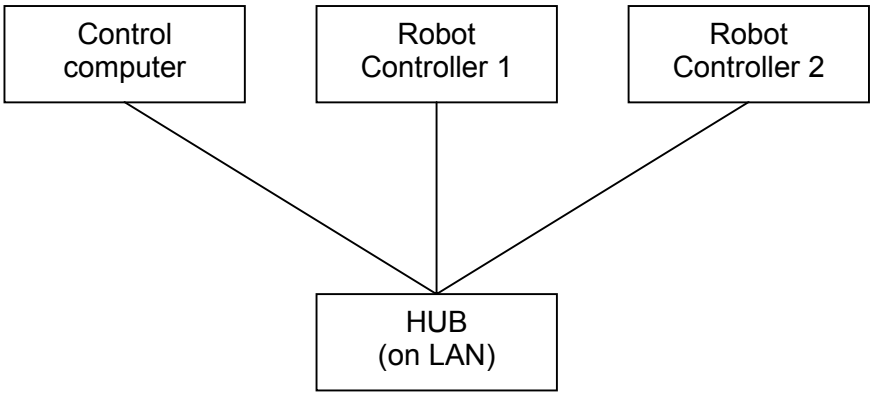


Figure 5.8 Ethernet network outline

5.4.2. Required software

Section 4.3 outlined a few different software options that are available for the robot. The software that was required for this project can be divided into that loaded on the controller and those loaded on the PC.

The controller is loaded with a BaseWare operating system. The FactoryWare Interface 3.1 is loaded on top of the operating system and it includes the Robot Application Protocol (RAP), which is used to communicate with a PC. Ethernet Services 3.1 are loaded to support the proprietary Ethernet card. This service also enables the robot to run a program directly from the PC's harddrive.

The PC must be loaded with the RobComm 3.1 ActiveX control, which enables development systems such as MS Visual C++, MS Visual Basic, MS Visual J++ and

WonderWare InTouch 7 to incorporate robot control and monitoring into custom applications. The correct network drivers must be loaded to communicate with the robot via the Ethernet network.

5.4.3. Integration of RobComm into the robot server application

After the hardware and software have been installed and set up properly on the robot controller and PC, the integration of RobComm into the robot server application can begin. Although the server application have been created with MS Visual J++, no use have been made of the Visual J++ specific functionality. In other words, the application could be run on any platform other than MS Windows and would operate as it should.

ActiveX controls are MS Windows specific and programs that rely on them cannot be executed on another platform. There are two scenarios namely the Java robot server application that can be run on any platform and the RobComm ActiveX component that can only be used in a MS Windows environment. Visual J++ allows a trade-off to allow the integration of the RobComm component into the Java robot server application but then the Java application is not platform independent anymore. This is a not really a problem because all the client applets are independent of the robot server application in terms of what platform they are run on. Only the server application needs to be run on a Windows platform.

Visual J++ makes it relatively easy to use ActiveX components in a project. Once the ActiveX control is registered (this will be done when RobComm is installed and loaded) it may be dragged onto the appropriate form (window), its properties must be

set and then it can be used in the code by referring to its name as specified in the objects properties. This is appropriate when Windows Foundation Classes (WFC) are used to create the application forms (see chapter 2 for details on WFC). For this project, however, no use was made of WFC (Microsoft, 2000).

As mentioned above, use was made of non-Windows specific, pure Java apart from the inclusion of the ActiveX control. The control cannot be placed on a form because the forms are created using code only. The control must be imported using the `import` statement similar to the way packages were imported to use them in a class implementation. A class of type RobComm must then be instantiated in the main application class and the properties of the class must be set manually via set and get functions provided by the ActiveX control. To enable events to be generated by the RobComm class, event handlers and observers must be set up. A sample implementation is shown below.

```
import com.ms.activex.robcomm.*;

//Main application class
public class RobServer extends Frame implements Runnable{
  RobComm RobotCtrl(this);
…
  RobotCtrlHandler handler;
  handler = new RobotCtrlHandler(this);
  RobotCtrl.AddActionListener(handler);
…
  x = RobotCtrl.getAxis2Position();
  n = RobotCtrl.getAxis4Position();
…
  RobotControl.setTCPOrientation(12.3,45.7,89.6);
…
}
//Listener class
class RobotCtrlHandler implements RobotListener{
  RobComm rc;
  public RobotCtrlHandler(RobComm r){ rc = r;}
…

  public void RobotStateChanged(StateEvent ie){
```

```
    if(ie.getSource()==rc.Axis1Position)
      { … }
    else if(ie.getSource()==rc.Axis2Orientation)
      { … }
…
}}
```

The above code shows an example of how the RobComm ActiveX control may be added to the existing codebase without fundamentally changing the structure of the code and ensuring code reuse. It also shows that although the code is changed to run on Windows systems only (due to the inclusion of the ActiveX control), the client-server communication and operation is unaffected by the changes.

5.5. Conclusion

This chapter covered the four major components of the project in detail. This consists of the VRML robot model, the client control and communications applet, the robot server application and the RobComm ActiveX control, which links the robot with the robot data contained in the server.

The VRML model is the main front end of the system giving a three dimensional virtual representation of the physical industrial robot. It was constructed in an object-oriented-like fashion to ease its use and provide reusable code.

The client control and communications applet mainly serves as a graphical user interface for clients to control the physical robot. It provides and maintains the important link to the robot server and thus provides a data communications pipe. The applet can also be run on practically any machine or operating system because no platform specific code were used.

The robot server application is the program responsible for controlling the physical robot. It is also in charge of which clients may connect to the robot and its associated data and thus which client has the right to control the robot. The application ensures that all connected client are immediately updated of new robot data as it becomes available.

The RobComm ActiveX, together with its support software on the PC and the robot controller, provides the robot server application with the ability to control and monitor the physical robot via a local network.

The concluding chapter will show some test methods and the results that were obtained from these tests.

CHAPTER 6 CONCLUSION

This chapter will show the final results of this project. It will also compare these results to the objectives of the project that was set out in Chapter 1. Problems that were encountered during the project's development will then be discussed followed by possible future extensions to the project.

6.1. Project results

The model of the IRB1400 robot manipulator that was created in VRML for this project together with the client communications applet is shown in Figure 6.1.

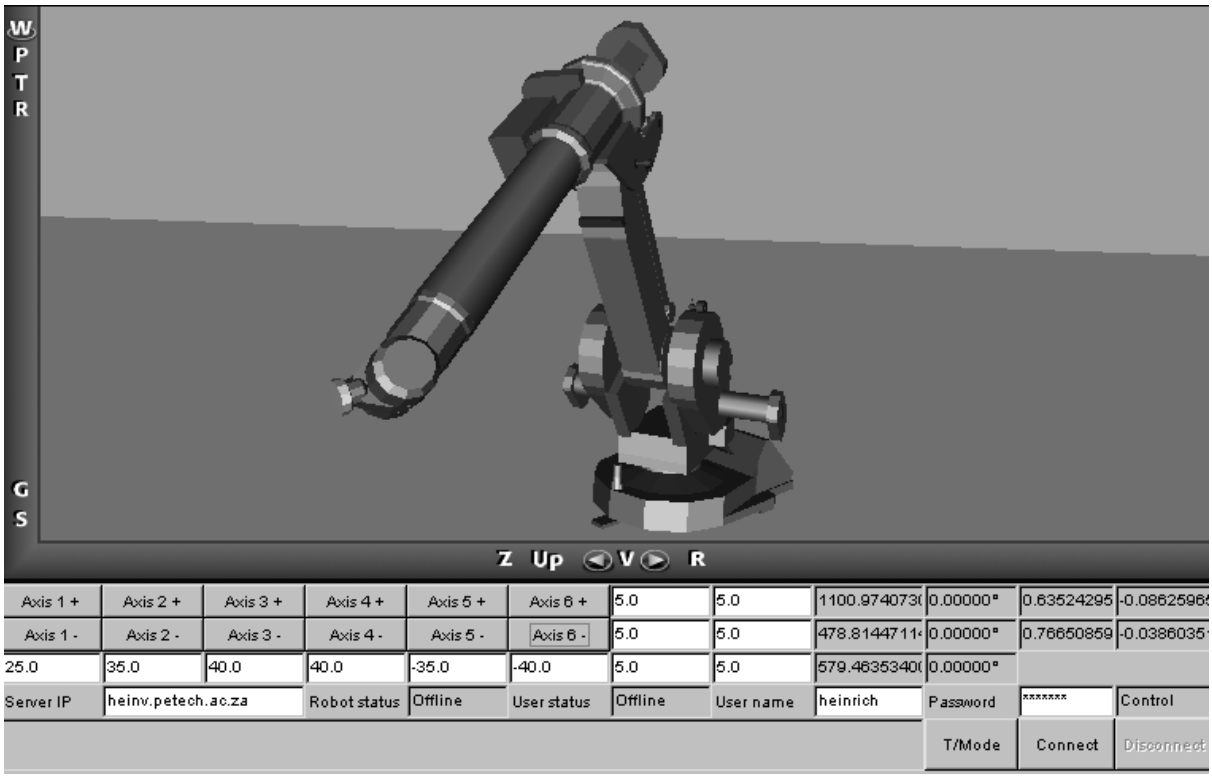


Figure 6.1 The VRML model of the IRB1400 robot manipulator

Two personal computers on the same LAN were used to test the project results. The one computer (host PC) was used to set up the robot application server. This PC is connected to the physical robot via an Ethernet connection. The other PC that is

connected runs the client communications applet and VRML model inside a Web browser. The PCs were physically in different rooms so that the operator of the client computer could not have visual contact with the robot. The host PC also had the client communications applet and VRML model running inside a Web browser but this client only acted as a monitor user while the other machine was logged on as the control user of the robot.

The following test procedure was performed with this setup:

- The operator of the client computer must open the Web page, on which the VRML and client communications applet resides, into a Web browser.
- The operator is confronted with the Java security dialog box shown in Figure 6.2 and must respond by pressing 'Yes' to allow the applet access to system functions such as networking (see Section 5.2.6 – Applet security). The applet is then started. At this stage there is no access to the robot yet.
- The operator must type in the correct server address and click on 'Connect'. The client will then be connected to the robot server application and receive the update of the current status of the robot. The client will now be a monitor user. The login will not be successful if 20 users are currently connected (this is a preset maximum number).
- The operator must type in the correct login and password and click on 'T/Mode'. If the login and password is correct and no other control user is connected the client will be made the control user. All changes that are made to the robot model will be sent to the robot server application. This server will move the robot to this new position and send the updated robot status to all other clients

connected. Thus these changes will also be reflected by the client running on the host PC.

- When the control user client has completed robot control or the link to this client is broken, another client will be allowed to become the control user.
- The operator can also change the robot position from the robot server application's user interface that runs on the host PC. These changes will be handled in the same manner as when a control user controls the robot.

Figure 6.2 Java security dialog box

The above test procedure was successfully performed. The control user successfully controlled the robot and all clients connected were correctly updated with the robot status and their models reflected these changes.

6.2. Comparison of final results to expected results

This research project investigated existing hardware and software tools for VR modeling so as to identify appropriate components needed for this research. It also examined currently available VR models and found that VRML was very well-suited for the task with extensions to design custom VR applications as has been shown in this project.

VRML also provided a generic user interface that can be used to display VR models of manufacturing processes. VRML is well established as the de facto standard for transferring 3D graphics over the Internet making it all the more useful as a generic user interface to manufacturing data.

The VR model shown in Figure 6.1 gives an excellent representation of VRML's ability to produce VR models of manufacturing processes. Interactivity of the model is an inherent part of VRML and the model that was created for this project. Although various other input devices to the VR model were studied, only the use of a mouse and keyboard was implemented, as this was not part of the main focus of the project.

The interface that was chosen between the VR model and the robot data, was a Java interface. This interface proved to work well with both VRML and the data stored in Java robot server application. Java provided a very good application basis especially because of its ease in connecting applications via the Internet.

The client communications applet also provided a means of simulating the manufacturing processes (control an industrial robot in this case) while another user (control user) was controlling the robot. This simulation was by using the VR model and it was integrated with the robot data derived from the robot server application.

6.3. Problems encountered

Problems that were encountered in the project were mostly of a practical nature. It is a pity that no hardware was available to have physical access to the robot or its controller at the time when this project's implementation was done. This aspect may have made the research all the more concrete and significant for industry. This also inhibited testing of the complete system.

The system was tested in a local area network that is connected to the Internet via a firewall (for security reasons). This, however, made it impossible for outside users to access the Web page with the client communications applet and VRML model. The testing could therefore only be done on the LAN. Another option would have been to connect the computer that was connected to the robot, directly to the Internet and provide it with an IP and firewall but this was beyond the scope of this project.

It is a pity that the robot software that is available for the Windows operating system only provides access to the robot through an ActiveX control, which is Microsoft specific. This made it difficult to use with an open platform language such as Java.

Although VRML has been standardised, it is still an evolving standard with additions being made to it. Many browsers do not fully comply with the standard or may

interpret the standard in different ways making it difficult for content creators to design content that will be the same in different browsers and platforms.

The VRML model that was created is a true representation of the IRB1400 robot. It could, however, have contained more inherent VRML behaviours that would have simplified the communications and control applet. For example the problem that was encountered in controlling axis 2 and 3 of the robot could have been done using a VRML behaviour.

6.4. Possible extensions and conclusion

Many extensions are possible from this basic setup. This project proved the viability of using VR models of an industrial robot to control the robot via the Internet. By the same token many other manufacturing processes can be modelled in VRML and linked to the physical manufacturing process via the Internet as described in this project.

It is also possible to create a number of models of objects that work together in a common workcell and all these objects can be included in one virtual environment. One example of this would be two robots performing an arbitrary task on one ore more objects in a common workcell.

The TCP/IP socket links between the server and each client can be made secure so that unwanted third parties will not be able to manipulate the data. This will be more of an issue for high-risk applications.

An Inverse kinematic model of the robot manipulator can designed to allow users to point to objects (if they exist) in the virtual environment and let the robot pick them up or place them at a specific location intelligently.

This project has shown the use of Virtual Reality to visualise, monitor and control an Industrial Robot via the Internet. The research proved that it is viable to make use of VR models to accomplish this task and has also shown its applicability for other manufacturing processes in general.

The aim of this research was to make a contribution to the manufacturing industry by laying a foundation for remote access of manufacturing processes and by identifying the ability to control and visualise manufacturing processes remotely, interactively and in a 3D format.

APPENDIX A REFERENCES

ABB Flexible Automation. (1998a). Product Manual – IRB 1400.

ABB Flexible Automation. (1998b). RAPID Reference Manual – RAPID Overview.

Aspragathos, N.A. & Dimitros, J.K. (1998). A comparative study of three methods for robot kinematics. IEEE Transactions on Systems, Man and Cybernetics. Part B, Vol. 28, no. 2, pp.135-145.

Beier, K.P. (1999). "Virtual Reality: A short introduction". Article available from Internet URL http://www-vrl.umich.edu/intro/.

Bolmsjo, G.S. (1988). "Robotic arc welding – Present status and future trends". Lund Institute of Technology, Lund. Article available from Internet URL http://www.robotics.lu.se/Robotics/publications/index.html.

Broll, W. (1998). "DWTP – An Internet Protocol For Shared Virtual Environments". Proceedings of VRML98 Third Symposium on the Virtual Reality Modeling Language, Monterey, pp.49-56, CA USA.

Brutzman, D. (1997). "Virtual Reality Transfer Protocol (VRTP) Design Rationale". Workshop on Enabling Technology: Infrastructure for Collaborative Enterprises: Sharing a Distributed Virtual Reality, pp. 179-186, Massachusetts, CA.

Brutzman, D. (June 1998). The Virtual Reality Modeling Language en Java. Communications of the ACM, 41, pp.57-64.

Bryson, S. (May 1996). "Virtual Reality in Scientific Visualisation". Communications of the ACM, 39, pp.62-71.

Buchanan, W. (1997). Software Development for Engineers. London : Arnold Publishing.

Campbell, B. (1997). "Extending VRML 2 with Java." VRML Developers Journal vol. 1 (1), pp. 42-47. Article available from Internet URL http://www.sys-con.com/vrml/.

Carpenter, I.D., Ritchie, J.M., Dewar, R.G. & Simmons, J.E.L. (1997). Virtual Manufacturing. Manufacturing engineering, 76, pp.113 – 116.

Coors, V. & Jung, V. (1998). "Using VRML as an Interface to the 3D Data Warehouse". Proceedings of VRML98 Third Symposium on the Virtual Reality Modeling Language, Monterey, pp.7-13, CA USA.

Detert, R. (August 1999). "The Amazing ActiveX - Part 1". Article available from Internet URL http://www.irt.org/articles/js178/index.htm.

Djurcilov, S. & Pang, A. (1998). "Visualisation Products On-Demand Through the Web". Proceedings of VRML98 Third Symposium on the Virtual Reality Modeling Language, Monterey, pp.121-127, CA USA.

Dowling, K. (1996). "What is robotics?". Article available from Internet URL http://www.frc.ri.cmu.edu/robotics-faq/1.html.

EAI-WG (External Authoring Interface Working Group). (August 1998). "EAI Development History". Article available from Internet URL http://www.vrml.org/WokingGroups/vrml-eai/history/history.html.

Finlayson, R.A. (1999). "VRML 2.0 EAI Frequently Asked Questions". Article available from Internet URL http://members.xoom.com/muratak/eaifaq.htm.

Fu, K.S., Gonzalez, R.C., Lee, C.S.G. (1987). Robotics – Control, sensing, vision and intelligence. USA : McGraw-Hill, Inc..

Krick, J. (May 1997). "Site building: Using JavaScript and VBScript". ZDNet Developer. Article available from Internet URL http://www.zdnet.com/devhead/stories/articles/0,4413,2136910,00.html.

K-12 Outreach. (1997). "A short history of robots". Article available from Internet URL http://tc.engr.wisc.edu/zwickel/Outreach/histrob.html.

Laird, C. & Soriaz, K. (October 1997). "Choosing a scripting language". SunWorld. Article available from Internet URL http://sunsite.uakom.sk/sunworldonline/swol-10-scripting.html.

Laursen, J.V. (1997). "The Internet: Past, Present and Future – Internet & WWW History." Article available from Internet URL http://www.vissing.dk/inthist.html.

Lemay, L., Couch, J. & Murdock, K. (1996). 3D Graphics & VRML 2.0. USA : Sams.net Publishing.

Loftin, R.B., Engelberg, M. & Benedetti, R. (1993). "Applying Virtual Reality in Education: A Prototypical Virtual Physics Laboratory". IEEE Symposium on Research Frontiers in Virtual Reality.

Loeffler, E.C. & Anderson, T. (1994). The Virtual Reality Casebook. Thomson International Publishing : New York.

Marrin, C. (1996). "Anatomy of a VRML browser." Article available from Internet URL http://www.marrin.com/vrml/Interface.html.

Marrin, C., Kent, J., Immel, D. & Aktihanoglu, M. (1997). "Using VRML Prototypes to Encapsulate Functionality for an External Java Applet". Article available from Internet URL http://www.marrin.com/VRML/papers/InternalExternal/Chris_Marrin_1.html.

Marsh, D. (1998). "History of the Internet." Article available from Internet URL http://www.netvalley.com/netvalley/archives/mirrors/davemarsh-timeline-1.htm.

Matsuba, S. & Roehl, B. (1996). Using VRML – Special Edition. Indianapolis : Que Publishing.

McKerrow, P.J. (1991). Introduction to Robotics. Addison-Wesley Publishers : Singapore.

Megahed, S.M. (1993). Principles of Robot Modeling and Simulation. John Wiley & Sons : Chichester.

Microsoft. (1998). Make your Java code trusted in Internet Explorer. Microsoft Developer Network.

Microsoft. (2000). Importing ActiveX Controls. Microsoft Developer Network.

Venners, B. (1998). "The 'event generator' idiom". Article in JavaWorld, September 1998. Article available at URL http://www.javaworld.com/javaworld/jw-09-1998/jw-09-teq-chniques_p.html.

Milgram, P., Zhai, S., Drascic, D. & Grodski, J.J. (1993). "Applications for augmented reality for human-robot communication". Proceedings of International Conference on Intelligent Robots and Systems, Japan. Article available from Internet URL http://vered.rose.toronto.edu/people/david_dir/iros93/iros93.full.html.

Mitra, Marrin, C. & Walsh, A.E. (1999). "VRML, URNs and Universal Media. Recommended Practice Proposal". Article available from Internet URL http://www.web3d.org/WorkingGroups/media/proposals/urn.html.

Nadeau, D.R. (1997). "VRML Feature summary". Article available from Internet URL http://www.sdsc.edu/doct/publications/a6/intro_vrml.html.

Netscape Communications Corporation. (1997). "JavaScript Guide". Article available from Internet URL http://developer.netscape.com/docs/manuals/communicator/jsguide4/.

Newman, A. (1996). Using JAVA – Special Edition. Indianapolis : Que Publishing.

Parisi, T. (1997). "Continuing commitment to a growing technology." VRML Developers Journal vol. 1 (1), pp. 38-41, 47. Article available from Internet URL http://www.sys-con.com/vrml/.

Perl Mongers. (2000). "Perl Fast Facts". Article available from Internet URL http://www.perl.org/press/fast_facts.html.

Ressler, S., Wang, Q., Bodarky, S., Sheppard, C. & Seidman, G. (1997). "Using VRML to Access Manufacturing Data". Article available from Internet URL http://www.itl.nist.gov/div894/ovrt/projects/mfg/sima/vrml97/vimhtml.fm.html.

Ressler, S., Godil, A., Wang, Q. & Seidman, G. (February 1999). "A VRML Integration Methodology for Manufacturing Applications". Proceedings of VRML99 Fourth Symposium on the Virtual Reality Modeling Language, Paderborn Germany.

Robot Encyclopaedia. Article available from Internet URL http://www.trueforce.com/encyclopaedia/robot_encyclopaedia.htm.

Robot Institute of America, (1979). Article available from Internet URL http://www.frc.ri.cmu.edu/robotics-faq/1.html.

Roehl, B., Couch, J., Reed-Ballreich, C., Rohaly, T. & Brown, G. (1997). Late Night VRML 2.0 with Java. Emeyville, CA : Ziff-Davis Press.

Ropp, K.L. (1993). "Robots in the operating room". USA Food and Drug Administration. Article available from Internet URL http://www.fda.gov/bbs/topics/consumer/con00242.html.

Roskothen, M. (1997). "VRML-Bär." VRML Developers Journal vol. 1 (1), pp. 48-50. Article available from Internet URL http://www.sys-con.com/vrml/.

Scouten, M. (February 1999). "Evolutionary trends in Industrial Robotics". Electricity and Control, pp.41,43.

Stallings, W. (1997). Data and computer communications (5th ed.). New Jersey : Prentice-Hall Inc.

Sun Microsystems. (1997). “The Java 3D API. Technical White Paper”. Article available from Internet URL http://java.sun.com/marketing/collateral/3d_api.html.

Tanenbaum, A.S. (1996). Computer Networks. New Jersey : Prentice-Hall.

VRML Consortium. (1997). “The Virtual Reality Modeling Language. International Standard ISO/IEC 14772-1:1997. Part 1: Functional specification and UTF-8 encoding.”. Article available from Internet URL http://www.vrml.org/Specifications/VRML97/.

VRML Consortium. (1999). “The Virtual Reality Modeling Language. International Standard ISO/IEC 14772-2:1997. Part 2: External Authoring Interface”. Article available from Internet URL http://www.vrml.org/WorkingGroups/vrml-eai/eai_iso_cd_spec.pdf.

Walsh, A.E. (1999). “Universal Media Working Group”. Article available from Internet URL http://www.web3d.org/WorkingGroups/media/.

Warmkessel, B. (1999). “Introduction to Active Server Pages. Understanding ASP basics”. Article available from Internet URL http://www.ahref.com/guides/technology/199806/0601buz.html.

Yergeau, F. (October 1996). Request for Comments 2044. Article available from Internet URL http://www.faqs.org/rfcs/rfc2044.html.

Zakon, R. (1999). Hobbes' Internet Timeline v5.0. Article available from Internet URL http://info.isoc.org/guest/zakon/Internet/History/HIT.html.

APPENDIX B PAPERS PRESENTED AT CONFERENCES

The following papers (see attached) relating to this project was presented at international conferences:

- Vermeulen, H., Van Niekerk, T.I., Huang, J. & Hattingh, D. (1999). VRML to Monitor and Control an Industrial Robot via the Internet. Fifth AFRICON Conference in Africa, Vol. 1, p.561-564, September 1999.