

42

A Simulation Environment for Multi-User Telerobotics

by

Özgü Tokgöz

B.S., Middle East Technical University (1996)

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

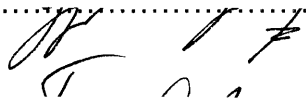
Master of Science

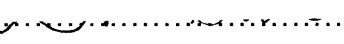
at the


MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1998

© Massachusetts Institute of Technology 1998. All Rights Reserved

Author
 Department of Mechanical Engineering
May 22, 1998

Certified by
 David Brock, Research Scientist
MIT Artificial Intelligence Laboratory
Thesis Supervisor

Accepted by
 Ain A. Sonin
Mechanical Engineering
Chairperson, Department Committee on Graduate Students

Eng

A Simulation Environment for Multi-User Telerobotics

by

Özgü Tokgöz

Submitted to the Department of Mechanical Engineering on May 26, 1998,
in partial fulfillment of the requirements for the degree of
Master of Science

Abstract

Networked telerobotic systems require the optimum use limited resources with operational efficiency. Providing access to a specific group of users and restricting access to the others during operation along with simulation tools for both training and real-time operation are the basis of this efficiency. A three dimensional virtual environment allowing multiple users simulate a collaborative task over the network has been developed. The simulation has been enhanced with a control interface allowing real-time control with embedded entity and task locking mechanisms allowing the access of a single user or a specific group of users to selected resources.

<http://ferrari.ai.mit.edu/demo/demo.html>

Thesis Supervisor: David L. Brock

Title: Research Scientist, MIT Artificial Intelligence Laboratory

Acknowledgements

I would like start with my advisor, Dave, who has sparked the fire in this thesis with his deep vision and brilliant ideas. The freedom he provided helped me to pursue my ideas confidently.

I am grateful to my partner Loma, who with his passion, enthusiasm, and never-ending energy motivated me to stay focused. Collaboration with him turned the most challenging problems into a “piece of cake”. Daniel, my ex-officemate, thank you for being such a fun person to hang out and enter the 50K competition with.

Güvenç, my “virtual” brother, no matter what I write down, it will not convey the full meaning, thanks for everything.

Hale, thanks for giving me the chance to see how marvelous life can be at MIT.

And all the lovely people, I had the chance to meet, thanks for the fun we had together.

Finally, I would like to thank my parents, who made a dream come true with their never-ending love, and support.

Table of Contents

1 INTRODUCTION	6
1.1 INTRODUCTION	6
1.2 BACKGROUND	6
1.3 OUTLINE	8
2 STRUCTURE OF THE SYNTHETIC ENVIRONMENT.....	10
2.1 INTRODUCTION	10
2.2 ENTITY DEFINITION AND CLASSIFICATION	10
2.2.1 <i>Deterministic Entities</i>	11
2.2.1.1 Static Entities	11
2.2.1.2 Animated Entities.....	11
2.2.2 <i>Dynamic Entities</i>	11
3 LOCKING MECHANISM	15
3.1 INTRODUCTION	15
3.2 CLASSIFICATION OF LOCKING TYPES.....	15
3.2.1 <i>Access Based Locking</i>	15
3.2.1.1 Explicit Locks	16
3.2.1.2 Implicit Locks	16
3.2.2 <i>Operation Based Locking</i>	16
3.2.2.1 Active Entity Locking	17
3.2.2.2 Passive Entity Locking.....	22
3.2.2.3 Task Locking	24
4 CLIENT PRIORITIZATION.....	32
4.1 INTRODUCTION	32
4.2 DEFINITIONS	32
4.3. SERVER AND CLIENT WAITING TIMES	33
4.4 CLIENT PRIORITIZATION	34
4.4.1 <i>Determination of the Threshold Value</i>	35
5 IMPLEMENTATION	37
5.1 INTRODUCTION	37
5.2. SYSTEM DESCRIPTION	37
5.3 VRML MODELS	38
5.4 JAVA CONTROL INTERFACE	41
5.5 VIDEO IMAGE INTERFACE.....	47
6 CONCLUSION	49
6.1 INTRODUCTION	49
6.2 REVIEW.....	49
6.3 FUTURE DEVELOPMENTS IN THE FIELD.....	50
6.3.1 <i>Developments in Network Connections</i>	51
6.3.2 <i>Developments in PC Hardware</i>	51
6.4 FUTURE WORK	52
6.4.1 <i>Hardware Design for Networked Telerobotics</i>	52
6.4.2 <i>Programmable Interface Design for Networked Telerobotics</i>	52
6.4.3 <i>Predictive Simulations for Space Exploration</i>	53
6.4.4 <i>Failure Detection and Prevention for Networked Machinery</i>	53

APPENDIX.....	54
A.1 THE VIRTUAL REALITY MODELING LANGUAGE	54
<i>A.1.1 Advantages of VRML as a Simulation Tool</i>	<i>54</i>
A.2 NETWORK SPECIFICATIONS.....	56
REFERENCES	59

Chapter 1

1 Introduction

1.1 Introduction

Computer networks have long existed to allow users to transfer information to one another. Operating machinery remotely over the network is an extension of this information transfer. It requires reliable data transfer back and forth via an interface, which can function equally well on multiple computer platforms allowing increased accessibility.

The types of machines operable over the network ranges from very high level complicated systems like the Hubble Space Telescope [21] and scanning electron microscopes to home automation systems as simple as turning on and off household devices [17].

This thesis work, although not covering the extreme ends of the machinery range specified above, defines standards for a control interface for multiple machines that can be operated collaboratively by multiple users over the network. The interface provides a locking mechanism, which selectively gives access to users over the network while placing the others in a waiting queue.

A three dimensional virtual environment representing the real world is provided as a simulation tool for the users to test their inputs to the machines in advance. Since remote operation can be collaborative, the virtual environment is also designed to receive multiple input from different users and simulate the outcome.

Section 1.2 provides a background by introducing the definitions of the terms frequently used in the context of this thesis and by giving information about similar research in the field. The chapter concludes with section 1.3, presenting the outline of the thesis.

1.2 Background

Networked Telerobotics is simply defined as supervisory control of robots via the network. Teleoperation is the control of an object or a process remotely by a human operator regardless of the autonomy of the object or complexity of the process. Supervisory control is defined by Sheridan; "in the strictest sense, supervisory control means that one or more human operators are intermittently programming and continually receiving information from a computer that itself closes an autonomous control loop through artificial effectors to the controlled process or task environment." [38].

Telerobotics has found application in space program, undersea environments, hazardous environments, medical operations, manufacturing, and training [11],[20].

The Internet, which is originally defined as a “distributed heterogeneous collaborative multimedia information system”, provides the most readily available network for teleoperation. Controlling robots from the Internet took start with independent works of Ken Goldberg at the University of Berkeley [13], and Ken Taylor at the University of Western Australia [42]. The work of Kevin O’Brien and Dr. David L. Brock at MIT Artificial Intelligence Lab established the first telerobot with a virtual interface [28].

The term Virtual Reality (VR) used by many different people with many meanings. There are some people to whom VR is a specific collection of technologies, that is a Head Mounted Display, Glove Input Device and Audio. Some other people stretch the term to include conventional books, movies or pure fantasy and imagination. For the purposes of this thesis VR will be restricted to computer mediated systems. The selected definitions of VR [17] in parallel to this text are:

“Artificially stimulated simulation” by Marjan Kindersley

“A way for humans to visualize, manipulate and interact with computers and extremely complex data” , from the book "The Silicon Mirage"

"A tool for design and forward simulations of the real world".

The last definition is anonymous and closest to the way it is utilized in this thesis.

The recent attention to VR over the last decade stems mainly from a change in the perception of the accessibility of the technology. Though its roots can be traced to the beginnings of flight simulation and telerobotics displays, drops in the cost of interactive 3D graphics systems and miniature video displays have made it realistic to consider a wide variety of new applications for virtual environment displays [10], [27],[38].

The development of Internet Oriented Virtual Reality tools such as Virtual reality Modeling Language (VRML) introduced the terminology “ Distributed Virtuality” as Bernie Roehl defines it: "a simulation environment running on multiple computers connected over the Internet, allowing users sharing the same virtual world interact in real time” [36].

There are recent applications of multi-user virtual reality such as Distributed Interactive Simulations (DIS) over the Internet which is led by Don

Brutzman of Naval Postgraduate school [5],[6] and virtual socializing environment, Living Worlds by Tim Regan of BT labs [32].

The virtual environments designed for telerobotics must allow translation of the viewpoint, interaction with the environment, and interaction with the autonomous agents. All this must occur through sensory feedback and control. The human-machine interface aspects of telerobotic systems are, therefore, highly relevant to VR research and development from a device, configuration, and human perspective.

A large part of the supervisor's task is planning, and the uses of computer-based models have a potentially critical role. The virtual environment is deemed an obvious and effective way to simulate and render hypothetical environments to pose "what could happen if" questions, run the experiment, and observe the consequences. Simulations are also an important component of predictive displays, which represent an important method of handling large time delays.

Telerobotics and VR should therefore not be viewed as disparate but rather as complementary endeavors whose goals include the exploration of remote environments and the creation of adaptable human-created entities.

This thesis work utilizes the existing interface technology to provide architecture for a multi-user networked telerobotics. A virtual environment allowing the users to simulate the actions of intelligent machines is presented along with a locking mechanism, which regulates the access to the machines in order to provide the optimum use of limited resources available for networked operation.

The locking mechanism introduces object and task locking and provides a probabilistic analysis to solve The interface is designed specifically for multi-user operation and provides means to the clients of the system interact, share data and plan and simulate collaborative tasks.

1.3 Outline

This thesis describes a simulation environment for multi-user networked telerobotics. This chapter presented an introduction to the topic and made some definitions relevant and essential to the thesis, as well as a review of some literature and current research. Chapter two introduces the properties of the simulation environment, by defining and classifying the concept of entity. This chapter concludes with the discussion of entity behavior to illustrate how the simulation environment receives input and provides user-based output for multiple users in the network.

Chapter three presents the locking mechanism, which monitors and arranges the access to machinery in the telerobotic environment. This mechanism is discussed in sections namely, machinery, object, and task locking. The problems that arise with task locking are solved with a probabilistic analysis based on the user's access time to entities.

Chapter four provides optimization techniques and user prioritization for the locking mechanism presented in Chapter three.

Chapter five is the implementation chapter where the theory in the preceding three chapters are implemented over the Internet using the available programming tools such as Java, VRML, and Hypertext Markup Language (HTML).

Chapter six concludes with a review of the thesis. Predictions on software and hardware development that will increase the efficiency of the implementation are made. The thesis is concluded with future work suggestions in the field.

Chapter 2

2 Structure of the Synthetic Environment

2.1 Introduction

This chapter defines the properties of the synthetic environment created for multi-user networked telerobotics. The “synthetic” environment represents both the real and simulation environment. Since the aim of simulation is to represent the real environment as closely as possible, both of these environments are designed to share the same properties. Section 2.2 introduces the definition of an “entity” and presents different entity categories. The data transmission mechanism, which allows the modification of the synthetic environment along with the concept of “task” execution, is then introduced. These definitions provide a background for the locking mechanism that is presented in the next chapter.

2.2 Entity Definition and Classification

Entities are objects in the synthetic environment. They are described by vectors that identify their position, orientation, velocity and acceleration in the environment space. They also have other distinguishing characteristics such as shape, color, and texture.

The classification of entities in the simulation environment is depicted in figure 2.1.

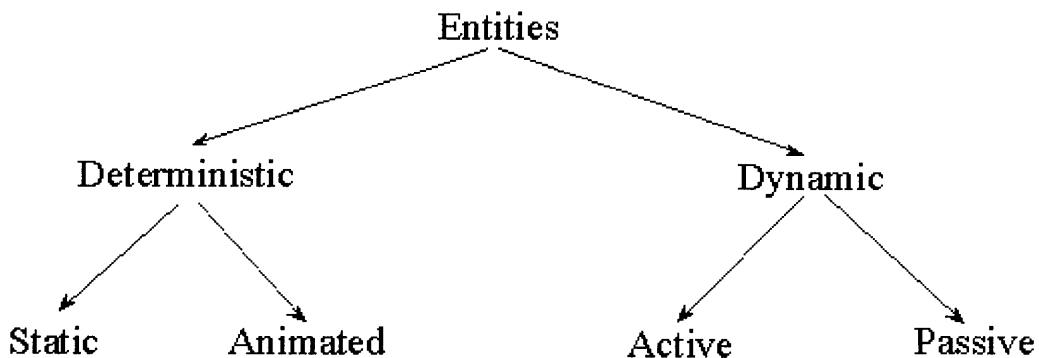


Figure 2.1: Entity Classification in a Synthetic Environment

The classification is mainly based on the entity states, which are the physical properties of the entity such as position, orientation, velocity, or temperature.

2.2.1 Deterministic Entities

The deterministic entities are built into the system during initial setup and do not change their states by user input. They are further classified into static entities and animated entities.

2.2.1.1 Static Entities

They are basically the building blocks of the surrounding environment such as the workspace the machines and other entities are located. In the simulation environment, these models have fixed states, which have been programmed at initial setup. This static environment provides a navigation option to the user so that the user can move around the world and observe it from many different viewpoints.

2.2.1.2 Animated Entities

Animated entities do not have fixed states as opposed to static entities: their states change as a function of time however the changes are predictable and already programmed into the system. The most descriptive example for this category is the movement of a clock's hand. During simulation, increasing the number of animated entities helps reducing the network load. This issue will be explained more clearly in the implementation chapter.

2.2.2 Dynamic Entities

Dynamic entities form the most important category in the synthetic environment. Like animated entities, the state of dynamic entities change as a function of time. However these changes occur as a result of the data input sent over the network. The change of states in response to this data occurs based on the laws of dynamics. Data transmission is achieved by means of a server-client system. Server is a set of instructions programmed into a host machine or a group of host machines, which listens to and receives connections over the Internet. The intelligence built into the server will be presented step by step within the context of this thesis. Additional information is also provided in the appendix.

Clients are defined as the agents who get connected to the user interface and make requests to submit data to an active entity. The server then selects

one client to be the user who is allowed to make inputs into the synthetic environment. This is the distinction between the client and the user. Clients can only observe the tasks specified by the users; they cannot send their own data. As an insanely simple analogy, the user can be thought of as the driver of the car capable of giving the inputs, whereas the clients are the passengers who only observe the outcome of the inputs specified by the driver. This concept is illustrated in figure 2.2.

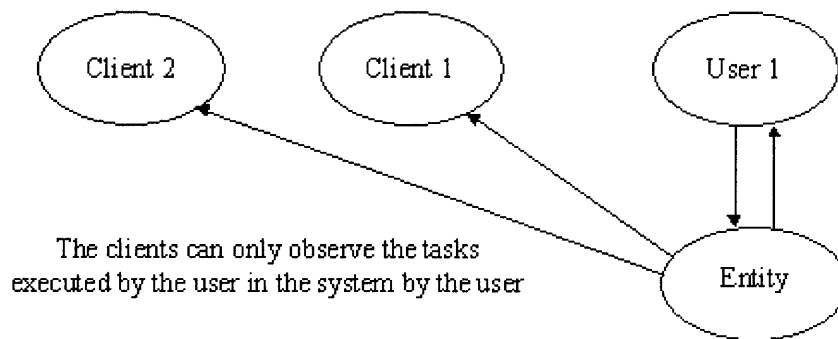


Figure 2.2: The Distinction between the User and the Clients in the System

The dynamic entities can further be divided into another category such as active dynamic entities and passive dynamic entities. Active entities are the direct recipients of the data. Their states are observable, and modifiable at all times via data input from the network.

The interaction between an active and a passive entity occurs with physical contact. The states of the passive entities are just observable initially, and they become modifiable when contact between two entities is established. This interaction is shown in the figure 2.3.

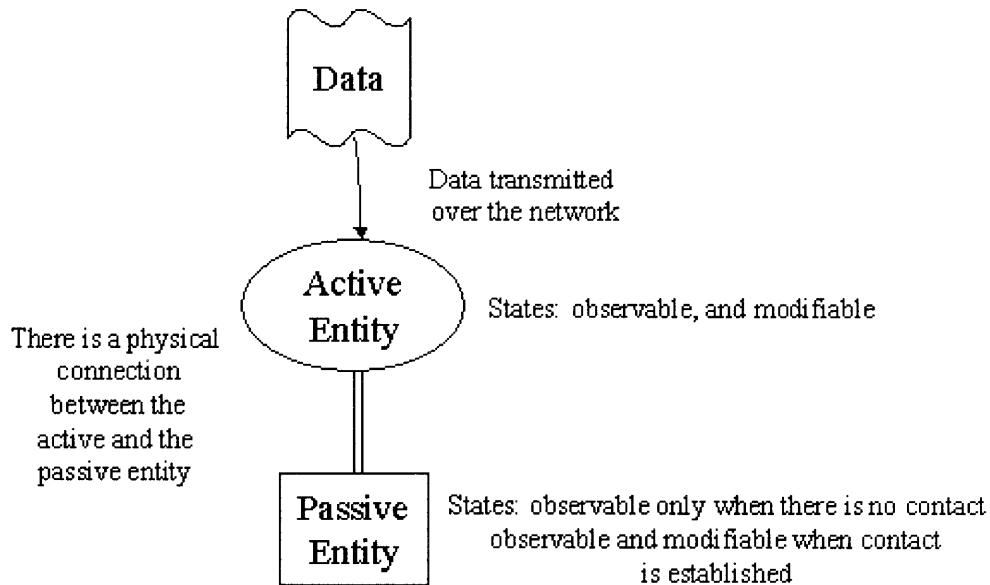


Figure 2.3: Interaction between Active and Passive Models

Consider a robot arm, which is moving a block from one position to another. The robot is the active and the block is the passive entity in this example. Initially the states like position velocity of the block are observable but not modifiable. Whenever the robot grabs the block, the physical contact is established and they start moving together, and thus the states of the passive object now become modifiable. Besides, at this point the passive model can be considered as a sub-entity of the active entity in the sense that they share the same states such as position, velocity and acceleration. Thus, this sub-entity assumption reduces the number of variables required for dynamic calculations.

However, this is not necessarily the only and the best example. An active entity can be a refrigerator trying to keep the temperature, of several passive entities at a certain value. The state of the passive entity is changed via the change of the state of the active entity. In this case, the refrigerator temperature is set to a specific value; this eventually changes the temperature of the stored items in the refrigerator.

The availability of active entities for data input introduces three new states, namely on, off, or down. On state means that that entity, whether passive or active is being used by a selected user, in other words locked by a user, and is going to be manipulated in the simulation environment. The details of the locking mechanism are going to be discussed in the next section. Off means that there is no lock imposed on the entity, however it is available for this locking mechanism. Down means that a mechanical failure has occurred in the entity, In the simulation environment, this entity changes its color to a predefined color, which is selected as red in the implementation. If this state occurs during the simulation, the simulation stops, this entity is no longer available for locking, and cannot be manipulated until the failure is fixed.

The interaction of active entities with passive entities in response to data input over the network is defined as a task. In cases, where there are multiple active and passive entities are involved, tasks are divided into subtasks, each dealing with a single state change of a passive entity. This task definition is further going to be used in the next chapter to establish a task locking mechanism which will grant the execution right of a task to a selected user over the network.

Chapter 3

3 Locking Mechanism

3.1 Introduction

In this chapter, a locking mechanism for multi-user telerobotics has been developed. Locking an entity in a telerobotic environment is defined in the simplest sense as preventing all other users from accessing an entity while a user is using it. The locking mechanism developed in this thesis accepts requests from clients over the network, and gives one client the right to send data to the entity and still keeps the other clients in the network queue where they can do simulations before gaining access to the entity.

Using the simplest analogy, locking mechanism serves as a way to determine who is the driver and who are the passengers in a car. As soon as one driver completes the lap, one of the other passengers is given the right to drive. The mechanism makes sure that nobody sits in the driver seat passenger without gaining the proper access right.

3.2 Classification of Locking Types

Locking in a multi-user environment can be classified under two categories, namely access based locking and operation based locking. These categories are not mutually exclusive but are used to develop the theory in this thesis.

3.2.1 Access Based Locking

Access based locking determines how access rights to entities are given to selected users in the system. The access right is either controlled by the user or the server as illustrated in the figure 3.1.

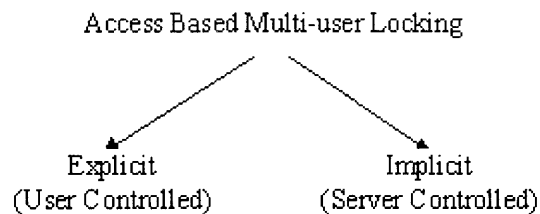


Figure 3.1: Classification of Access Based Multi-User Locking

3.2.1.1 Explicit Locks

These are the locks established by the users of the system. The user explicitly locks the machine he wants to work with as long as it is available, and when the task is over the user can at, his own will, releases the lock.

3.2.1.2 Implicit Locks

These locks are the ones, which are controlled by the server. They are specifically included in the classification so that server has the ultimate right to decide on which user locks a specific entity. The following example will clarify this point:

Consider a user who has established a lock on the machine he is currently using. During the operation, a network loss occurs and the user gets disconnected. If there are only explicit locks are the system, then this machine would not be recoverable until that user reconnects to the system. Therefore the server is given the superior right to remove the previous locks in these cases and grant them to new clients. The waiting time for the server to decide if the user has got disconnected from the system is defined in Chapter 4.

3.2.2 Operation Based Locking

The second classification, shown in figure 3.2, is based on the types of operations locks are used for. It is named as Operation-Based Locking. The classification is given in the figure below. Entity locking includes the locks established on the dynamic entities discussed in section 2.2.2. Active locking is defined for locks on the active dynamic entities whereas passive locking is defined for the ones on the passive dynamic entities. During task execution,

passive entities can become sub-entities of active ones. However, as far as locking is concerned, all the locks are established by the user, in other words an active entity cannot lock a passive entity but rather establish physical contact with it as required within the context of the task specified. Task locking is defined under a new classification due to the fact that it embodies both active and passive locking and introduces its unique design problems, which will be discussed and resolved in section 3.4.

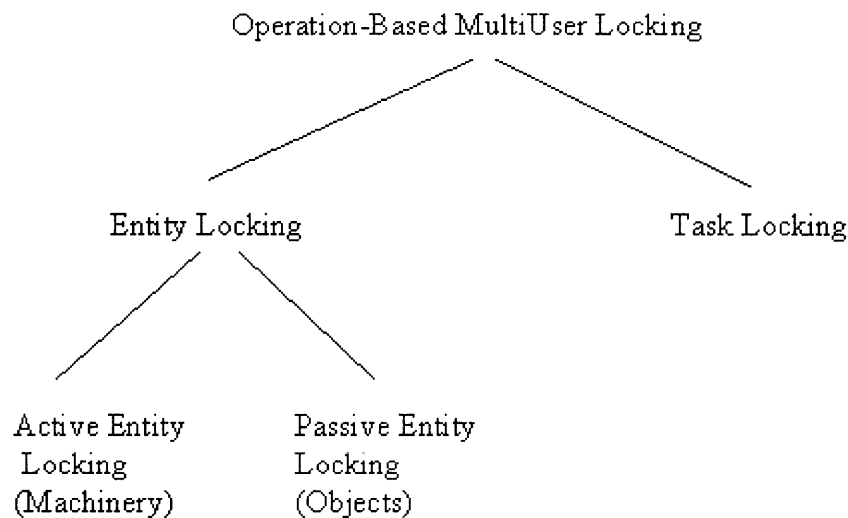


Figure 3.2: Classification of Operation Based Multi-User Locking

3.2.2.1 Active Entity Locking

Based on the locking mechanism described above, entity locking is simply giving access right to a selected user in the network. When there are new users assigned to an active entity, the server waits for a specified amount of time t , and accepts connections from the clients as shown in figure 3.3.

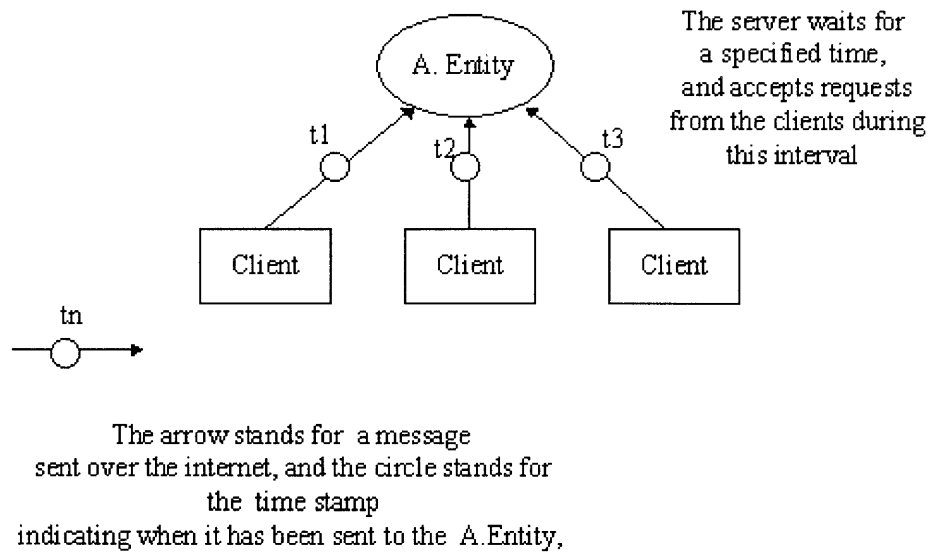


Figure 3.3: Clients Trying to Access an Active Entity over the Network

At the end of this time interval, the server gives access to one of the clients and puts the others in a waiting queue as shown in figure 3.4 by using a client prioritization mechanism. Each client placed in the waiting queue is sent a network message indicating his approximate waiting time. The calculation of the waiting time for the server to wait for client connections, user prioritization mechanism, and waiting times for each client, will be discussed in detail in Chapter 4.

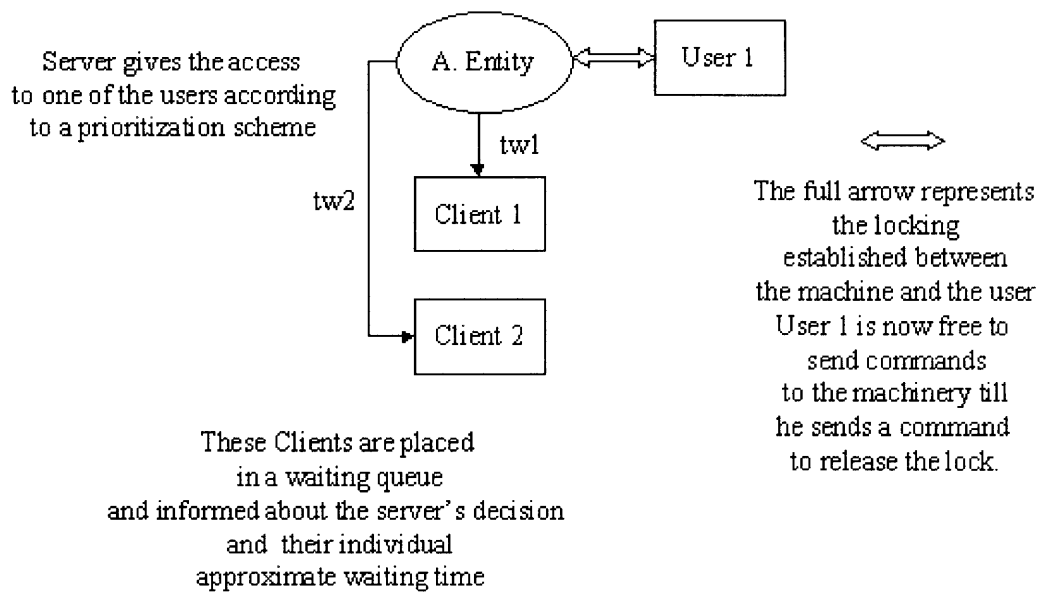


Figure 3.4: User Selection by the Server

During the time interval, T_{ws} , which will be defined as the server waiting time in section 4.3, the current user has the access to the active entity, the server keeps accepting new clients over the network as shown in figure 3.5.

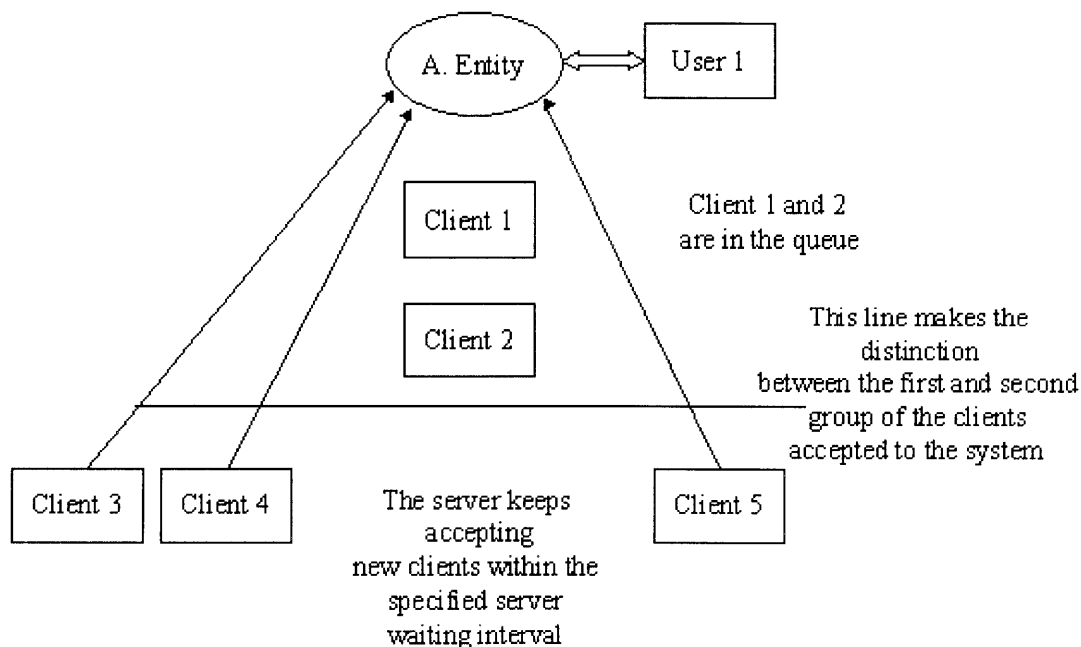


Figure 3.5: Accepting New Clients into the System

After accepting these new clients in fixed intervals of T_{ws} the server places them in the queue, and these clients are also notified about their approximate waiting time in the system as shown in figure 3.6.

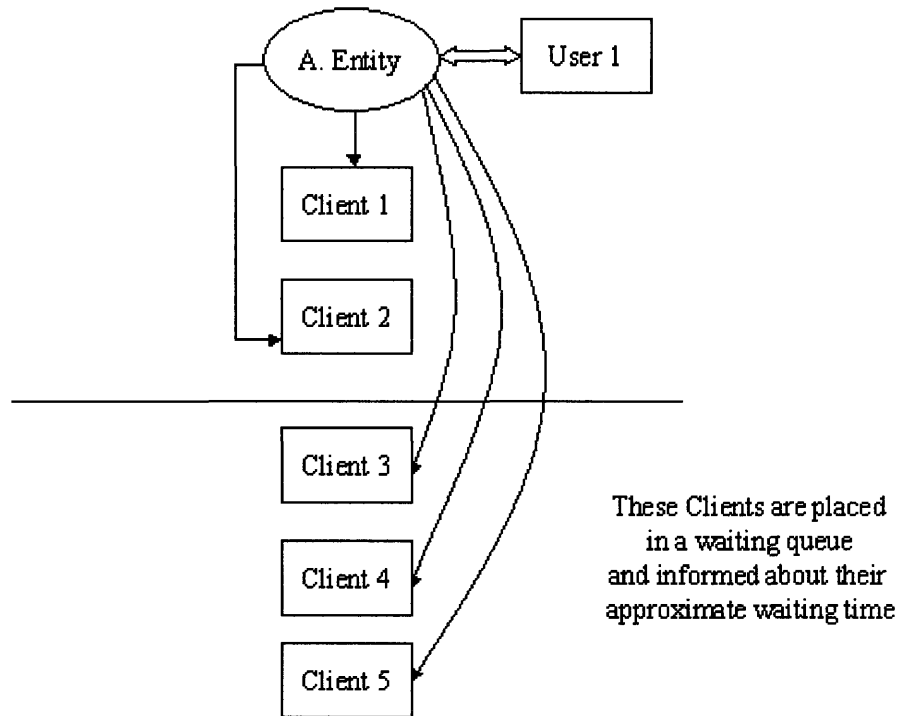


Figure 3.6: Average Waiting Times for Each Client in the Queue

All of the clients in the system have the chance to observe the actions of user1, but will not have access to operate this machine. However, they can use the simulation tool to test the tasks they are planning to perform with the entity. Whenever, user1 is finished with his operations, he removes his lock by clicking on the “release” control button, as depicted in the figure 3.7.

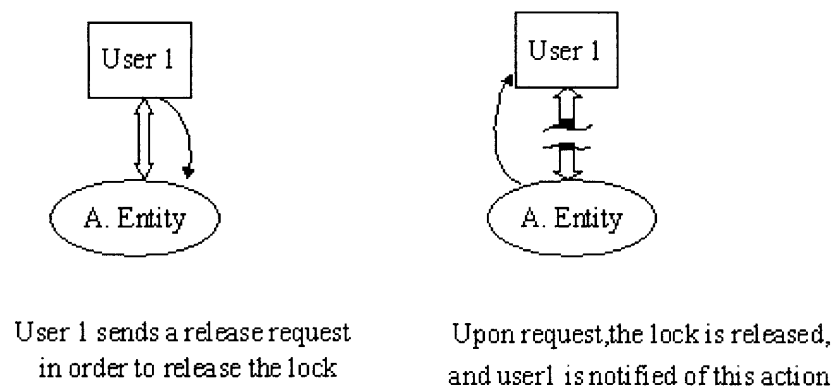


Figure 3.7: User 1 Trying to Release the Entity

As soon as the current user releases the lock, the access right is automatically given to the next client in the waiting queue. The server keeps track of all the clients in the queue, and if one of the clients does not want to keep waiting and leaves, the queue, then the server recalculates the waiting time. The following figure 3.8 shows the transfer of the access right to the next client.

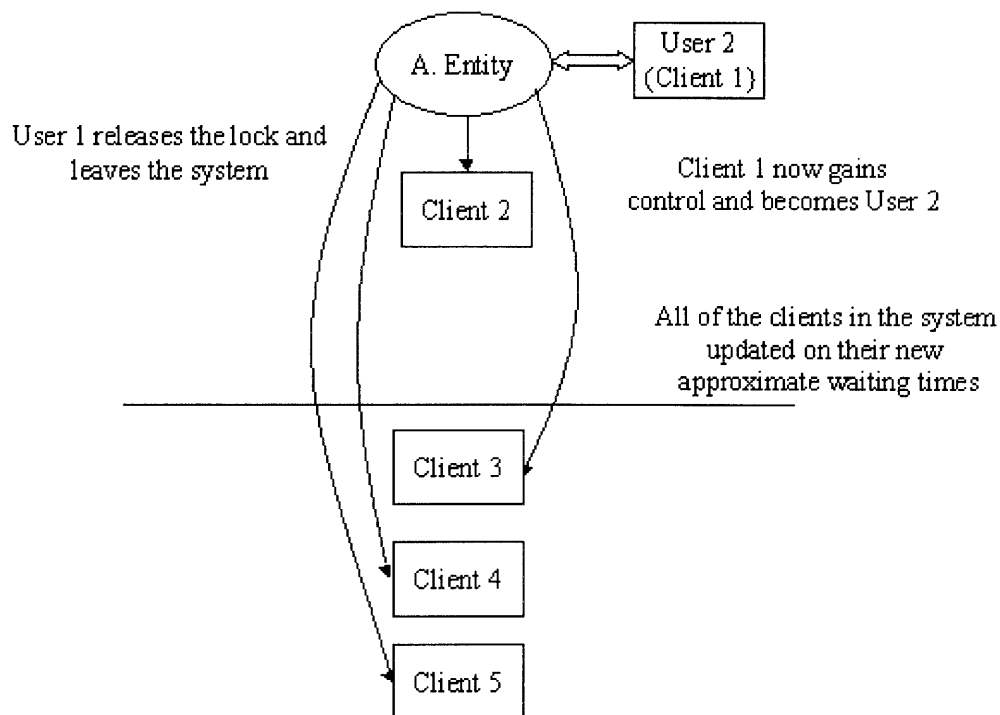


Figure 3.8: Assigning a New User to the Entity

It should also be noted that the clients connecting to the system are given priority only within their own group. In other words, their best location in the queue is the first place in their own group. This principle is depicted in figure 3.9.

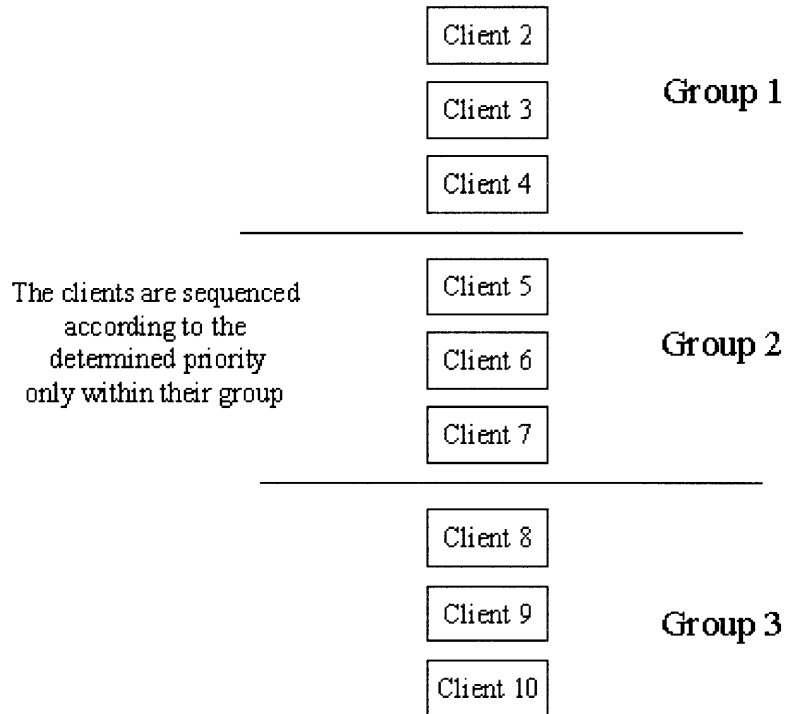


Figure 3.9: Grouping Clients depending on the Connection Interval

This type of grouping allows the users with less priority to still have the chance to access the machinery, because these users will still be ahead of the user who have connected to the system later on in the successive groups. Otherwise the sequencing would be unstable, in other words, the users with more priority would keep accessing the system while the others wait indefinitely in the queue.

3.2.2.2 Passive Entity Locking

The previous chapter determined how active entities are assigned to users in the system. However, since active entities are used together with passive entities to accomplish tasks as indicated in the task definition in section 2.2.2, without an additional locking mechanism, the following situation could have occurred: Consider a refrigerator in a chemistry lab, that is storing two solutions. The refrigerator is the active entity and the solutions are two passive entities. If the user has locked one of those passive entities, then another user with an active entity like a robot is not allowed to pick up and remove this solution that is

supposed to stay within the refrigerator. This second user can, however, access the other solution if any other user in the system did not lock it previously. The problem that occurs without a locking mechanism is also illustrated in the figure 3.10.

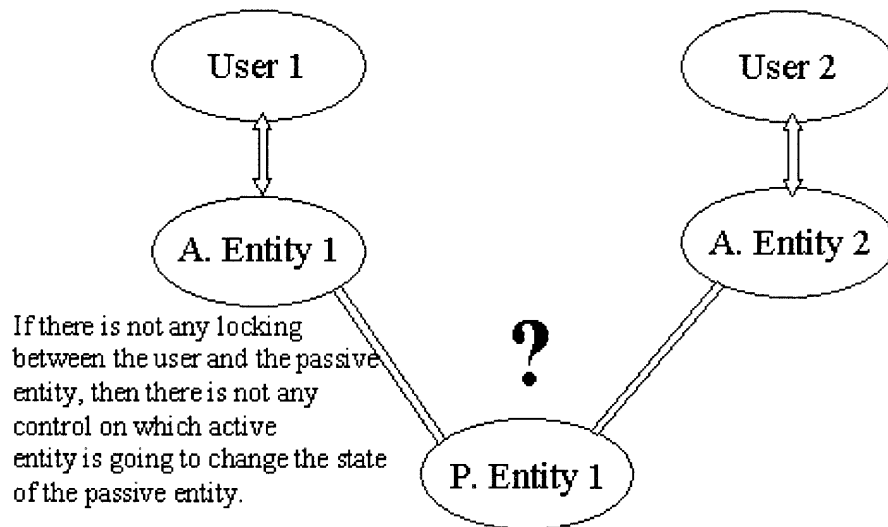


Figure 3.10: Object Locking Problem in a Multi-User Environment

Therefore, the user interface also provides a list of objects that can be used during the operation of the machine. Using the interface, the user controlling the machine can lock one or several objects as depicted in figure 3.11.

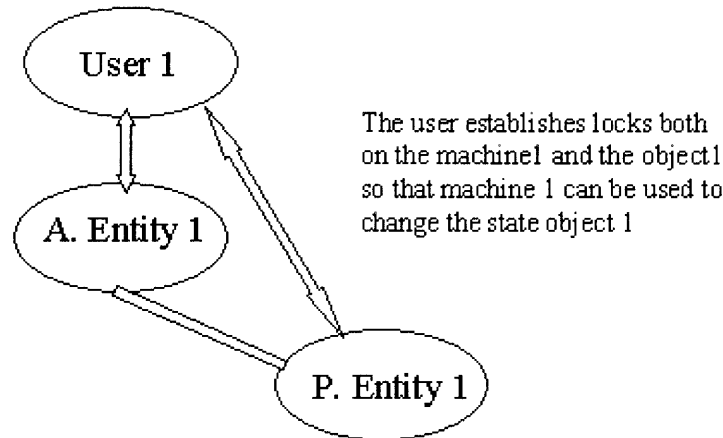


Figure 3.11: User 1 Locking both Machine 1 and Object 1

Thus, when multiple users try to access the same object, the user who has established object locking gains access to the object and the other user is notified of this locking established. This locking does not necessarily have to be established during the initial login to the system. The availability data of the objects are continuously displayed to the user so that new object locking and releasing can be made at all times during operation. Due to the delay existing between the time data is sent by the user and the time it is received by the server, an object that has just been locked might temporarily seem available to another user. Therefore the user always has to wait for the server's confirmation on the locks he tries to establish. In addition, the server requests the user to confirm previously established object locks at the end of each task so that objects that are not going to be used any further are released from that user.

3.2.2.3 Task Locking

The third type of operation based locking is task locking. In order to minimize the operation times, and release the workload from the user we can have pre-programmed tasks built into the interface. These tasks can be the most frequent operations that the users perform. Creating these tasks also decreases the network load, and hence the transmission delay because instead of sending data, observing the output, and sending another set of data, this sequence can be programmed into a task and can be executed by sending data to the system only once. However, unlike object locking, the user can only select one task at a time. Tasks, as expected, have the list of machines and objects to be utilized and

therefore task selection inherently takes care of both locking types discussed above. This is illustrated figure 3.12.

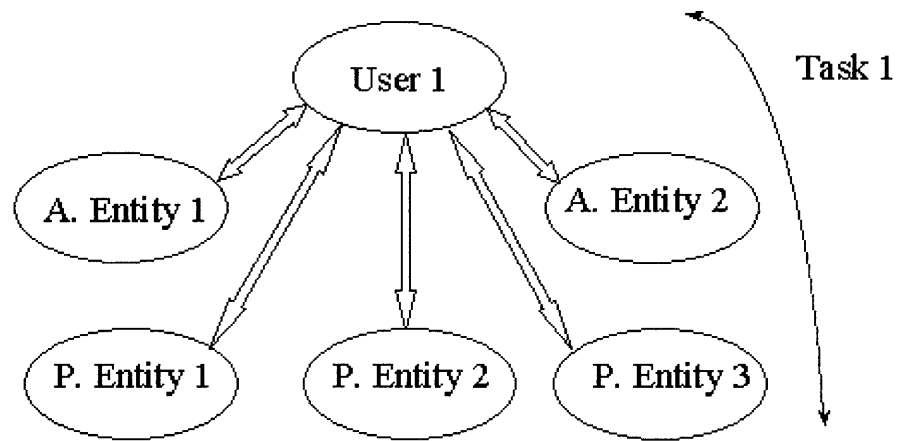


Figure 3.12: Task Locking in a Multi-User Environment

Once a user selects a certain task, until that task is completed, the other users are prevented to perform the very same task. However, two clients can choose different tasks, which might access to same objects during operation. This problem is illustrated in figure 3.13.

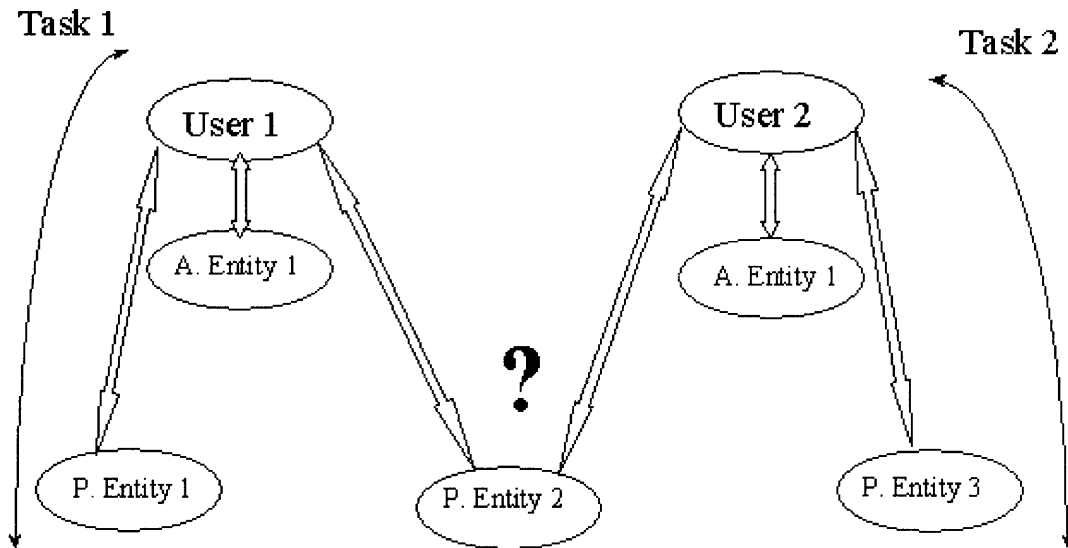


Figure 3.13: Task Locking Problem in a Multi-User Environment

The simplest yet the most unpractical solution is to reject access to task 2, however then the list of available tasks would be very limited because they have to provide access to totally different objects in the system. The solution implemented into the server is that, the task is divided into a couple of subtasks.

A subtask T_i is designed to be as compact. It can be as simple as changing only one state, S_{ij} of an active entity A_i . When this active entity is in contact with a passive entity, then the most compact subtask T_i becomes changing a single state S_{ij} of this passive entity P_i . In addition, for each subtask the average completion time and the probability of successful execution are also defined. Decomposition of tasks into simpler subtasks is adapted from the work of Brock [3] who introduced task level control subdividing tasks until, at some level, the decisions to be made are intuitive and programmed simply. The server simply allows the other tasks to be accepted and then it checks which subtask is currently being executed. If the shared object is never going to be used by task 1 again, the access and therefore the lock are directly given to the task 2. If both of the tasks will have access to that object in the future, the server calculates the access time to that object for each task, and gives the access to the task with the shorter access time. Whenever that task is completed, the object locking is automatically passed to the other task in execution.

Each subtask T_i is associated with an average completion time, tc_i , and a probability of successful completion, p_i .

There is a set of required inputs to achieve a subtask T_{i+1} after a preceding subtask T_i has been achieved.

In trivial case, a task may have x unique input variables, which are mutually exclusive from one another. That is to say, the changes in one or many variables do not influence the change in another variable. In this case, the set of different inputs has v elements.

If these inputs are not mutually exclusive, the set contains 2^x elements. For instance the accomplishment of a task may require the movement of the machine to a specific location which requires 3 dimensional variables. This creates a set of 2^3 possible inputs. At initial system setup, the subtasks have been tested for all these inputs for reliable execution and thus have an initial successful completion probability of

$$p_i = 2^x / 2^x = 1.0 \quad (3.1)$$

For each task, the total number of successful subtask executions s_s and the number of attempted subtask executions, s_a are recorded for each user. This data is then used to update the previous probability value so that the final probability p_i becomes:

$$p_i = (2^x + s_s)/(2^x + s_a) \quad (3.2)$$

Coming back to the previous example of moving to a location, if a user, for example has made 4 successful subtask executions out of 8 attempts, then the final probability of successful completion $p_f(n)$ for that subtask becomes:

$$p_i = (2^3 + 4)/(2^3 + 8) = 0.75 \quad (3.3)$$

It should be noted that these attempts may either be the operation of real machinery or simulation carried out in the waiting queue. This probability value is unique and has to be computed for each user. Other user customization attributes are discussed in section 4.2.

Then one way to estimated the task completion time is:

$$ETC_i = TC_i/p_i \quad (3.4)$$

If the probability of the event is one, then there will not be any failure during operation at all and therefore average completion time becomes equal to the estimated completion time, if the probability is less than one, then there is the chance of failure. Additional time might be spent trying to fix the failure occurred and repeat the task. If the task is impossible to do, that is $p(\text{task}) = 0$, then the estimated task completion time becomes infinity, as expected.

The execution of one subtask can begin only after a preceding subtask has been completed successfully. This relation is simply described in figure 3.14

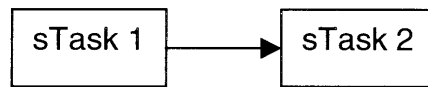


Figure 3.14: Two Tasks Connected in Series

Therefore a conditional probability definition should be made.

$P(2|1)$: probability of successful completion of task 2 given that task 1 is successfully completed.

N subtasks, which are serially connected, are shown in figure 3.15. The entity becomes available at the end of subtask n.

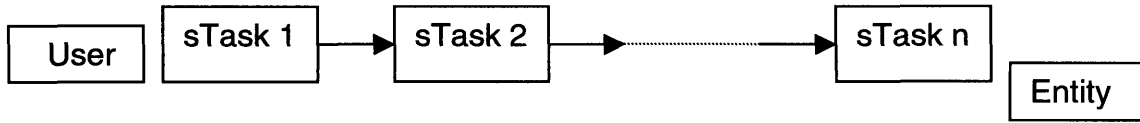


Figure 3.15: N Tasks Connected in Series

The time to reach that entity is given by :

$$T_e = \sum_{i=1}^n \left(TC_i / \left(\prod_{j=1}^n (p(j|j-1)) \right) \right) \quad (3.5)$$

Using this relation, entity access time in the following subtask layout shown in figure 3.16.

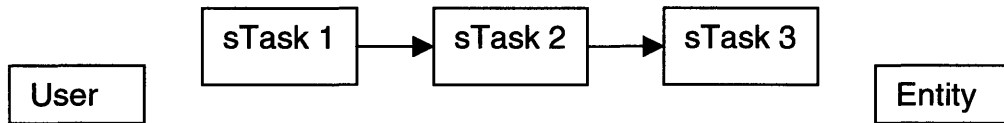


Figure 3.16: Calculating Entity Access Time for Three Tasks Connected in Series

can be calculated as :

$$T_e = TC_1/p(1) + TC_2/(p(2|1)*p(1)) + TC_3/(p(3|2)*p(2|1)*p(1)) \quad (3.6)$$

However, serial layout is not the only way subtasks are connected to one another.

A subtask can be executed only after a number of subtasks have been successfully completed, as illustrated in figure 3.17.

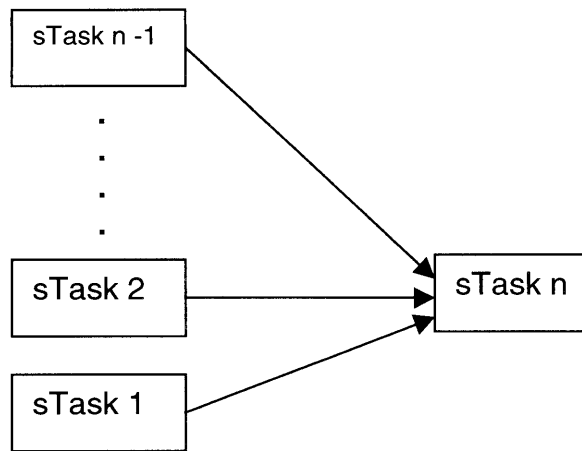


Figure 3.17: N -1 Tasks Combined to Trigger Task n

In this case, the probability of successful execution of task n, provided that all preceding tasks are accomplished successfully is defined as

$$p(n|n-1 \& n-2 \& n-3 \dots \& 1) \quad (3.7)$$

The other possibility is that a subtask may lead to multiple of subtasks. This case is depicted in figure 3.18.

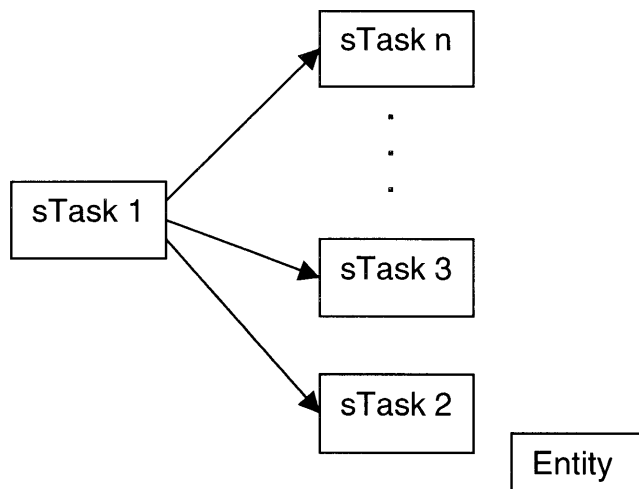


Figure 3.18: One Task Triggering Multiple Tasks

The entity in this case is reached only after these n tasks have been successfully completed. The time to reach this entity is then defined as:

$$T_e: \max (TC_n/p(n+1), TC_{n-1}/p(n), \dots, TC_3/p(3), TC_2/p(2)) \quad (3.8)$$

Since all of the tasks must have been accomplished before accessing the entity, the maximum completion time among these tasks is selected.

The following example, depicted in figure 3.19, will make use of these principles explained above to calculate the entity access time, T_e .

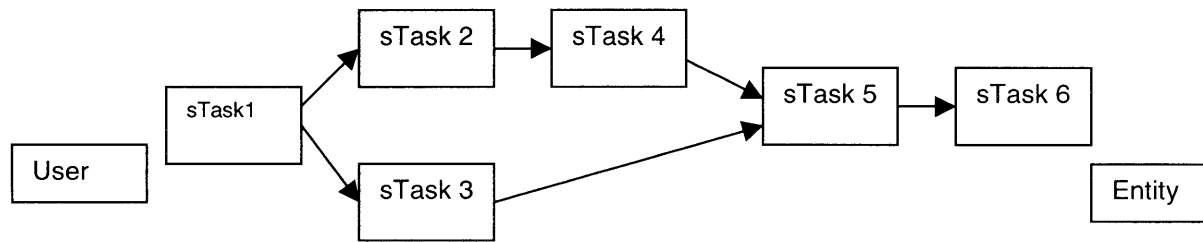


Figure 3.19: Subtask Layout for the Example Problem

In this example, the completion of subtask 1 triggers the start of both subtask 2 and subtask 4, and subtask 5 can begin only after both subtask 4 and subtask 3 are completed successfully. The entity becomes available at the end of subtask 6.

The time to reach entity, T_e , in this case is given by:

$$T_1 = TC_6/(p(6|5)*p(5|4\&3)*p(3|1)*p(4|2)*p(2|1)*p(1)) \quad (3.9)$$

$$T_2 = TC_5/(p(5|4\&3)*p(3|1)*p(4|2)*p(2|1)*p(1)) \quad (3.10)$$

$$T_3 = TC_4/(p(4|2)*p(2|1)*p(1) + TC_2/(p(2|1)*p(1)) + TC_1/p(1) \quad (3.11)$$

$$T_4 = TC_3/(p(3|1)*p(1)) + TC_1/p(1) \quad (3.12)$$

$$T_{e1} = T_1 + T_2 + T_3 \quad (3.13)$$

$$T_{e2} = T_1 + T_2 + T_4 \quad (3.14)$$

T_e , is then taken to be the greater of the two terms T_{e1} and T_{e2}

$$T_e = \max(T_{e1}, T_{e2}) \quad (3.15)$$

In this equation $p(5|4\&3)$ is the probability of success of subtask 5 given that both subtask 4 and subtask 3 are accomplished properly. T_e is selected to be the greater term among T_{e1} and T_{e2} due to the branching between subtask 2 and subtask 3 after the completion of subtask 1.

As the subtasks are completed, T_e is recomputed. For instance whenever subtask1 is completed successfully, TC_1 is set to zero, simplifying the calculations. If there is a mechanical breakdown during the execution of a subtask, or the network connection gets broken, completion time for the specific subtask is set to infinity (actually a very high number in the actual code) so that the sequence does not keep on locking an entity. Completion time is set back to its original value when the problem is fixed. Besides tasks might take actually take longer than the expected completion time due to a possible hardware or a network error, therefore the initial calculations have to be updated. The users are locking tasks with shared objects are continuously notified about the current object locks.

Chapter 4

4 Client Prioritization

4.1 Introduction

In the previous chapter, a locking mechanism, which determined the access to a selected entity has been introduced. Simply restated, the server waited for a fixed amount of time to accept requests from clients over the network. Afterwards the group of clients connected was arranged in a network queue, and one client in the group has been given the priority to access the selected entity. The other clients are informed of their waiting time, which is determined by their position in the queue formed. This chapter presents the prioritization method utilized along with the definitions of the server and client waiting times. The prioritization scheme is based on selecting the more experienced client, and therefore a discretization of experience, which depends on the input types sent, has been developed using the prior client data gathered in the system. Section 4.2 will introduce the definitions of the variables used to formulate the server and client waiting times that are going to be defined in section 4.3, and Section 4.4 will conclude the chapter by describing the prioritization scheme.

4.2 Definitions

In this section, we will introduce variables needed to compute the server wait time T_{ws} and individual client wait time T_{wc} .

Initial login time, T_l is defined as the time it takes for the client to login and to the interface site. T_l is dependent on the network load at the time of the day, complexity of the interface and the user's hardware capabilities. Complex interfaces that require larger data transmission will take longer time to load. T_l cannot be directly measured by server, and therefore client feedback is required for precise measurements. However initially an average value is determined by login tests from different sites.

Transmission time is defined as the time it takes for the data to travel between the client and the entity. T_{ft} is forward transmission time, which is the time from client to server, and T_{rt} is the reverse transmission time, which is from server to client, as shown in figure 4.1. Transmission time is directly dependent on the environment. It can be almost instantaneous for local area networks and a

few minutes or longer for space telerobotics (15 minutes of time delay between Earth and Mars).

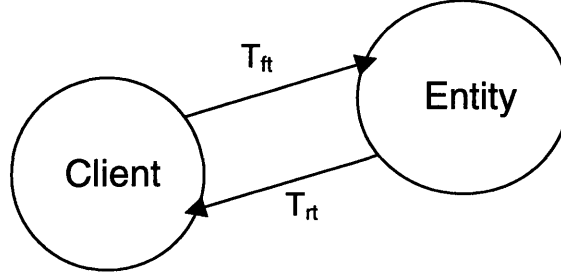


Figure 4.1: Forward and Reverse Transmission Times

Each task is associated with an average task completion time, TC_n as discussed in the previous text.

Each task also has a planning time, T_p which is the time it takes to decide to perform a certain task. For collaborative tasks, where there is n number of users T_p is defined as:

$$T_p = f(n) * T_p \quad (4.1)$$

In other words, it is a function of time and increases as the number of users involved, n , increases.

4.3. Server and Client Waiting Times

We set the server waiting time to be independent of the clients in the system. Since T_l , T_{ft} and T_p are not fixed quantities and are client dependent, mean values are used in the definition of the server waiting Time T_{ws} , which is:

$$T_{ws} = T_l + T_{ft} + TC + T_p \quad (4.2)$$

The server accepts new clients continuously at fixed intervals of T_{ws} . During the “off-hours”, the time intervals where the number of clients logging into the system is comparatively low, the server waits for an interval of T_{ws} only if the entity is currently being used by another user. Else, “first come first serve” principle is applied, and the client who gets connected to the system first gets the access to the entity. Otherwise, a single client would have to wait for an amount of T_{ws} before having access to the entity at a time with low connection rate. Thus, unnecessary waiting times are prevented. For the time being, the “off-hours” are

set to be between 11pm and 8am when the connection rate to the system is comparatively very low.

The server waiting time, T_{ws} is also used to accept users who were disconnected from the system during task execution. Whenever a user is disconnected from the system without releasing the entity, this means that there is a network connection loss. The server waits for a duration of T_{ws} without granting access to any other clients in the queue. If the disconnected user achieves to login to the system in this time interval, then he regains control, else the access is granted to the next client in the queue.

The waiting time for each client in the queue is calculated individually. The first client should wait for the time interval in which the current plans his task, (T_p), executes it (TC), and finally notifies the server that he is releasing the lock (T_{ft}). Then the server notifies the first client in the queue that access has been granted to him (T_{rt}). Therefore the waiting time for the first client is:

$$T_{wc1} = T_{ft} + T_{rt} + TC + T_p \quad (4.3)$$

Generalizing this expression, the client having the p th position in the queue will have an approximate waiting time of

$$T_{wcp} = p \cdot (T_{ft} + T_{rt} + TC + T_p) \quad (4.4)$$

When compared to the server waiting time T_{ws} , the client waiting time T_{wc} is not constant, it is an approximate value. The entity access right is passed to the next client only after the current user releases the lock. In other words, the actual waiting time can either be smaller or higher and that is why it is recalculated for each client in the queue every time a new user is assigned to the entity.

4.4 Client Prioritization

The missing part in the discussion is how we arrange these clients in this queue that we are talking about. The aim is to give priority to more experienced and successful clients yet, still allow the less experienced clients have considerable amount of access and allow them to build up their experience by using the simulation environment.

Before the discussion, it should be made clear that users are not necessarily human beings but may also be other entities in the system. This gives a chance for machines to communicate and execute tasks together. In this perspective, the prioritization scheme, especially evaluating the experience of the user, becomes applicable to only users. Since humans do not like to wait a lot in a queue, when the server accepts requests from the clients in time interval of T_{ws} ,

the machines are placed at the end of the queue in the group. The following principles are then used to arrange the human being remaining in queue.

The total number of attempted moves, and the number of accepted moves are recorded to determine the success rate of each client, S_r that is defined as:

$$S_r: \text{Number of accepted moves/Number of attempted moves} \quad (4.5)$$

The definition requires a definition of a threshold number of moves so that the calculation of the success rate becomes justifiable. Otherwise perfect records with small number of moves would have an overweighed value.

4.4.1 Determination of the Threshold Value

Each entity has certain pre-defined control values specified as inputs over the network. These inputs can be classified to be either binary or analog.

Binary inputs are numerical values specified within a range with cause a change in one or more states of the entity. A three degree freedom robot for example requires three binary input values for full operation.

The client is gaining logging to a system with multiple entities with different number of binary input variables. For the client to gain experience with an entity, he is at least expected to know the effect of each variable on the operation of that entity. In the trivial case, as discussed in section 3.4, for x number of mutually exclusive binary input variables, there are v different inputs, and if the variables are not mutually exclusive, then the number of different inputs becomes 2^x

Since the user has the option to collaborate with different users and use multiple machines, he is expected to know the operation modes of all of the machines in the system, therefore the number of different possible inputs, X , in the system becomes:

$$X = \sum_{i=1}^y 2^{x_i} \quad (4.5)$$

where y is the number of entities in the system. However, this equation is only valid for entities, which are mutually exclusive from each other, so that the changes in the states of one entity do not affect the states of another entity. If there are dependent entities in the system then they should be grouped together and their combined number of different inputs, x_c becomes:

$$x_c = \prod_{j=1}^k 2^{x_k} \quad (4.6)$$

where k is the number of dependent entities. Then the result x_c should be inserted in equation (4.5) to find the total number of different inputs, X . We have decided to set this number to be the threshold value to calculate the success rate S_r for each client. As long as the number of attempted moves exceed the total number of different inputs in a system, then the calculation of S_r becomes justifiable.

The clients in the waiting queue can be defined as either “veteran” of “rookie”. This is achieved by discretization of experience. If the number of attempts exceed the defined threshold value, the client is categorized as veteran, else as rookie. In a group of clients accepted to the system, priority is given to the veteran clients with the highest efficiency percentage. When there are no veteran clients left in the group, the priority passes on to the rookie clients, who are sequenced according to the login time and the client with the earliest time stamp gains access to the entity.

Another input type is the analog input, which can best be defined by the following analogous example. A driver in a car gives provides analog inputs to the car, and based on his expertise level, these inputs might end up in driving at a steady cruise speed of 20mph up to 200mph depending on the type of the car. The drivers who can drive smoothly at very high cruise speeds are expected to be the more experienced users, and less experienced users are expected to drive much slowly. Based on this definition, the analog inputs are divided into several operational ranges and each client is expected to use the simulation environment and have experience on the entity response before moving into the next range of inputs. The server keeps track of these operation ranges for each client and gives priority to the client who has reached a higher range of operation.

In order to make this prioritization scheme as stable as possible for both type of the inputs, the clients are accepted to the system within certain time intervals, T_{ws} , as defined in the previous section, and are arranged only within their group. A client, no matter how experienced might be, will not be placed anywhere above his own group within the queue. This provides a protection for the less experienced clients who have previously connected to the system. Besides, the clients are allowed to make simulations while they are in the waiting queue and thus can increase their experience and success percentage by making successful inputs. Client success rate is updated every time they leave the system.

Chapter 5

5 Implementation

5.1 Introduction

The purpose of this chapter is to present the implementation of the simulation environment and the locking mechanisms based on the design principles discussed in the chapters 2, 3, and 4. Section 5.2 will present the overall system description and will compare it with previous work in the field. Section 5.3 will present the multi-user simulation environment developed using the existing software tools, and finally Section 5.4 will present the user interface developed and explain how the locking mechanism is implemented using the existing programming tools.

5.2. System Description

We developed originally a control model for operating machinery over the Internet has been shown in the figure 5.1. Similar models also exist in the literature [13],[42], [28].

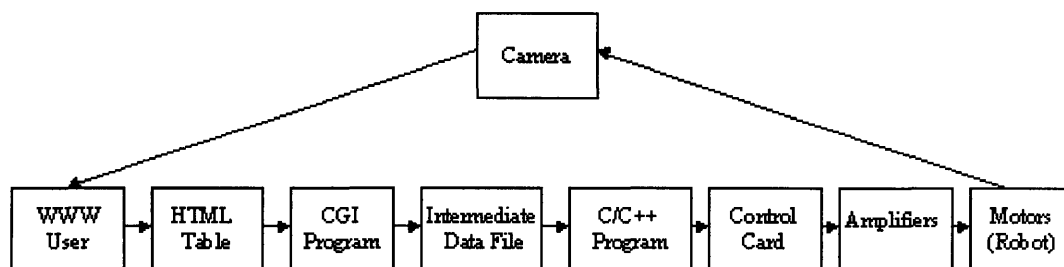


Figure 5.1: Telerobotic Control over the Internet (Previous Model)

The user logged into the homepage, and submitted data to the machine through html tables whose data is received and saved to an Intermediate data file by a Common Gateway Interface (CGI) program. The C or C++ program in the host machine read the data file and sent commands to the controller card which converted this data into the control signals. These signals are amplified and sent to the motors on each axis of the machine, and thus remote operation over the

network has been achieved. The user has seen the result of his action by camera images transmitted at regular intervals and text messages displayed on the html page. This system could only handle a single user at a time, and the other users trying to connect to the interface were simply blocked by an html page indicating that the machine is currently being operated by another user.

The interface developed in this thesis on the other hand has the following layout, as shown in the figure 5.2.

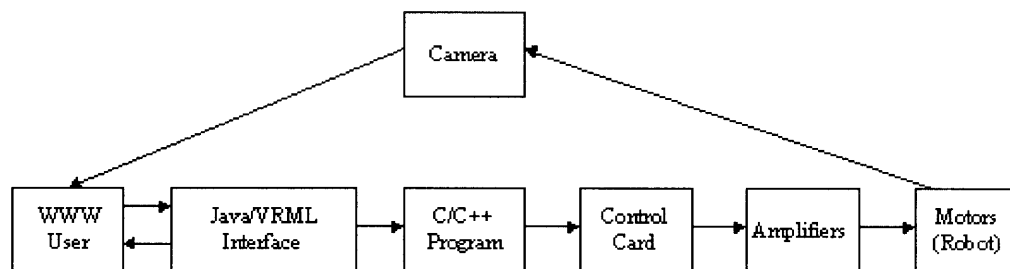


Figure 5.2: Telerobotic Control over the Internet (Current Model)

This model uses a Java/VRML interface, which directly calls the functions in the C/C++ program by the use of Native Methods in the Java language. A server-client architecture has been programmed in Java, and this allows multiple users to login to the system simultaneously. The interface consists of a 3D virtual model programmed in VRML and a Java applet. The networking details of the server-client system embedded in the applet have been described in detail in the appendix section. The VRML world presents a 3D model of the real system that is being operated remotely. This world is used to simulate future tasks before sending data to the actual machine. Camera image is still used as a feedback to show the user the operations being performed. The Java applet provides the control interface used to send data to the machine. It is also embedded with the locking mechanisms described in chapter 3.

5.3 VRML models

Several VRML models have been developed to implement the system discussed in the preceding chapters. The animation of the models has been programmed using the External Authoring Interface (EAI) [24], which allows manipulation of the VRML world with an external program such as a Java applet.

The world shown in figure 5.3 is the control interface for a Cartesian robot designed to be used in a chemistry lab.

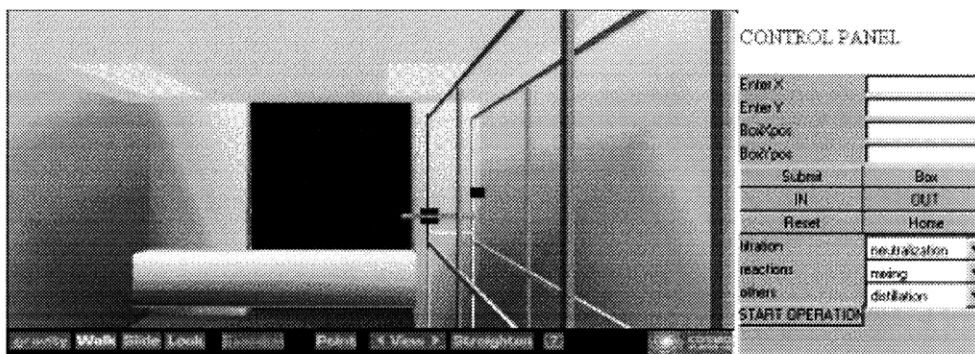


Figure 5.3: Virtual Interface of the Cartesian Robot

This virtual world has been designed to simulate the operations of the Cartesian Robot which is programmed to perform “pick and place” tasks in a chemistry lab to take objects to various lab machines such as refrigerator, oven, and pHmeter.

Event triggers in the simulations were directly programmed into Java instead of the VRML code and a continuous simulation sequence is obtained. For instance as soon as the Cartesian robot brought the object into the workspace, the machine in that workspace started operating on it. This very much resembles the real operating environment where the presence of the approaching object is detected with a sensor, and a second machine starts operating on it.

The VRML model of the Cartesian robot was built as simple as possible, yet identical to the real one in terms of operation. This is one of the main features of the models made so that the file size is reduced as much as possible.

The interface has various task options such as moving the robot to specific x, and y coordinates, moving the arm in and out along the gripper axis, grabbing and taking the object from its current location and delivering it to a specific work cell. There is also a task menu, which allows the simulation of tasks such as neutralization, mixing, and titration. Programming of these tasks, as indicated in chapter three, reduced the amount of data transfer over the network.

The second VRML model has been designed to present a multi-user simulation environment. A simple client server is prepared for this purpose. It accepts two clients and denies any other requests until one of the users gets out of the system. Each user can control one robot and observe the motion of the other robot. The control scheme is depicted in figure 5.4

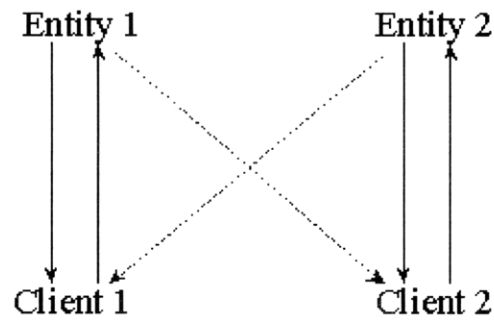


Figure 5.4: 1 to 1 Match up Simulation

Two models of 3 degree of freedom robots with end-effectors serving as a gripper are designed using V-Realm Builder. The interface is shown in the figure 5.5 below.

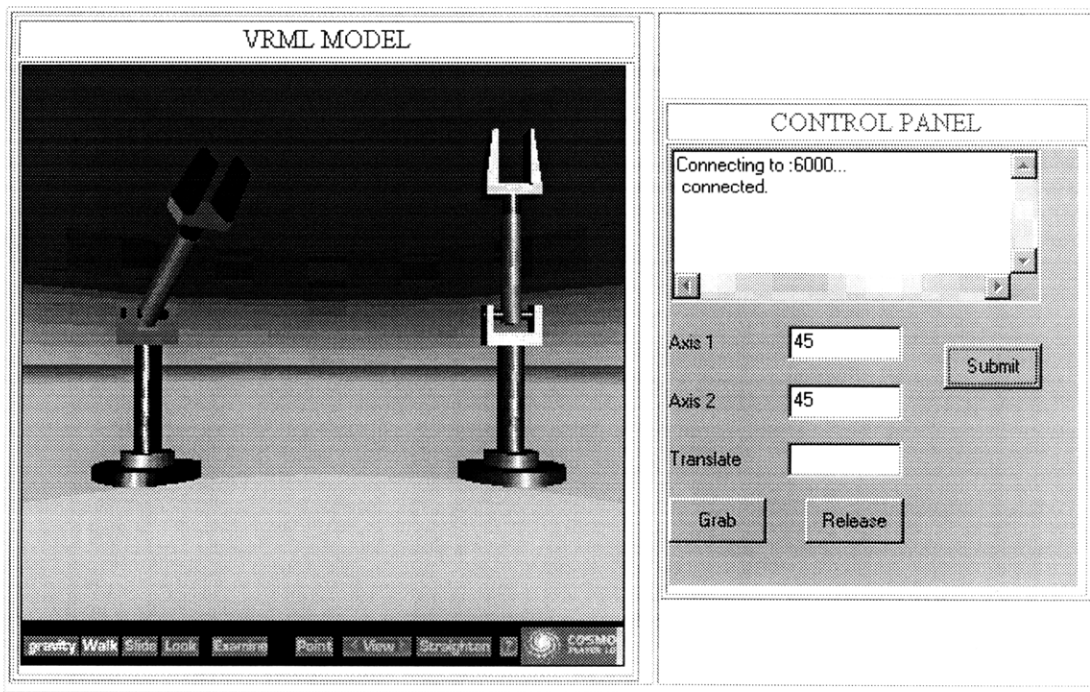


Figure 5.5: User Interface

5.4 Java Control Interface

The Java applet serving as the control interface is shown in the figure 5.6.

MULTIUSER TELEROBOTIC CONTROL INTERFACE

Started.
Connecting to ferrari.ai.mit.edu:6000... connect

Mach.ID	Time	Cur.User
Arm1	0:0:51	

Attempts Accepted

Data1 Data2 Data3

Submit Home Reset

Grab Release Simulate

DATA TYPE

☒ Angular
☐ Cartesian
☐ On
☐ Off

Other Machines

Name	Status	User
Arm2	Off	
Arm3	Off	
Arm4	Off	

Object Locking

☐ Object 1
☐ Object 2
☐ Object 3
☐ Object 4

SetLock

Task Locking

☒ Null
☐ Task 1
☐ Task 2
☐ Task 3

The other available machines are: **ARM2** **ARM3** **ARM4**

Figure 5.6: Java Applet Serving as the Control Interface for Multi-User Operation

For the initial log on we apply the “first come, first serve” principle. If no one is using the machine at that moment, the first user who clicks on the “grab” button gets the access to the machine, and all of the users that are connected are notified through a text message.

Unless the current user hits the “release” button, shown in figure 5.7, the other users cannot gain access to the machine. They are informed with a warning text message saying that the robot is being used by another user. Actually the person who is controlling that machine is continuously displayed as one of the text fields in the interface.

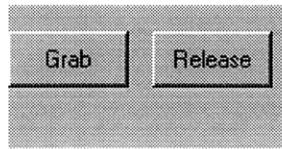


Figure 5.7: Grab & Release Buttons

Whenever the current user clicks on the “release” button, the other users are informed that the machine is available and again the user who clicks on the grab button first gets hold of the machine. The programming is achieved by using two flag values namely “control and precontrol. The precontrol values correspond to the current state of the machine. If the machine is occupied, its value is one, else it is zero. The control value stands for the control request from another client. If it is one it means that another user has clicked on the grab button, and if it is zero it means that there is no request from the users. To demonstrate this principle, let's assume that user A has the control of the machine, and the server is getting requests from user B. The possible outcomes are listed in the following table.

Precontrol value for A	Control value for B	Result
0	0	null, no one is in control
1	0	A keeps control
0	1	B gains control
1	1	A stays in control, the request from B is rejected

Table 5.1: Decision Mechanism for the Server

The total usage time, the number of moves attempted, and the number of accepted moves are also being kept track of. This data is then used to calculate the user efficiency as discussed in chapter 4.

Mach.ID	Time	Cur.User
Arm1	0:6:51	lamborghini.a
Attempts	Accepted	
7	5	

Figure 5.8: Current User Data Display

The interface also gives the status and user information about other machines in the environment. The server creates a data file for each machine in the environment, and continuously reads from the file to send the proper values to the clients. The use of threads for each client has been mentioned before. If one thread is reading from the file at the same time another thread is writing to it, we would have a bug in our client-server system. However, using synchronized methods in Java handles this problem and assures that only one thread accesses a file at a time. The architecture of data sharing is depicted in the figure 5.9. The machine status is designed to be “on”, off”, and “down”. “On” means that the machine is being currently operated by a user. The user is also specified in the near by text field. If the status is off, the machine is available and waiting for a user to get connected. The “down” status indicates that there is a problem with the machine, therefore it is not operational. This down signal will also be used by other machines in the system so that they will not provide any references to it. The interface is show in Figure 5.10.

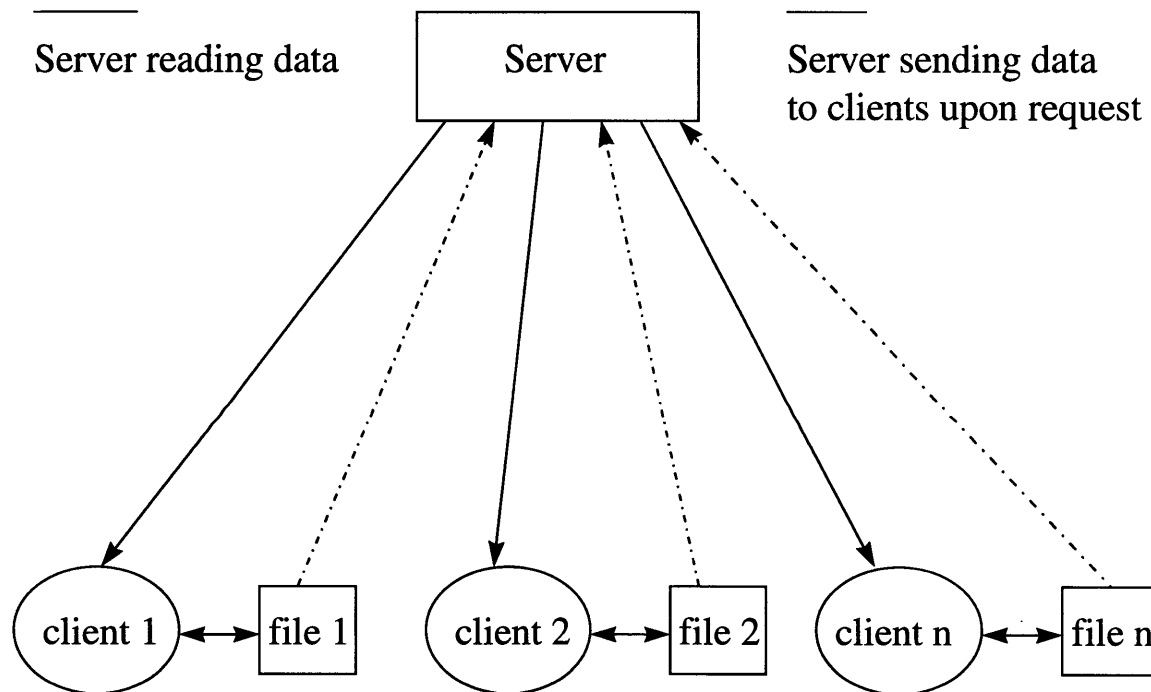


Figure 5.9: The Architecture of Data Sharing

Name	Status	User
Arm2	On	ferrari.ai.mit.e
Arm3	Off	
Arm4	Off	

Figure 5.10: Status of Other Machines in the System

The user can also try to access other machines by click on the hyperlinks at the bottom of the interface, which is shown in figure 5.11. If the user gets the user interface for arm3, then arm1, arm2, and arm4 will be displayed in other machine interface, as expected.

The other available machines are: **ARM2** **ARM3** **ARM4**

Figure 5.11: The Hyperlinks to Other Machines in the System

There are four checkboxes in the user interface implemented to provide the object locking discussed in section 3.3. When the client decides to use a certain machine, he also decides if he is going to use any of these objects during his operation, and clicks on these checkboxes. As described previously, there are data files for each machine. The first line includes the state and user of the machine. The second line has the list of objects associated with this machine. Therefore when a new user tries to lock an object, the server simply checks these data files, and if the object is not included in any of these files, then the access is given to this user. The content of the data file is presented in the figure below.

On (status) *(username)*
Object 1, Object 2 *(objects allocated by the user)*

As seen from the file, the user has the right to lock more than one object. The internal timer mentioned above, which decides whether the user is still connected to the machine also takes care of the object locking. As soon as the user releases the control of a machine, all the locks that user has placed on the objects in the system are disabled.

At the end of each task, the user has to confirm the locks he has established on the objects. The user also has the option to lock new objects, provided that they are available, or release previously locked objects once he is

done using them. This feature becomes really helpful when one object is manipulated for a short time during the operation and is never used again. After the usage, the user can simply release that lock, simply by setting the new locks and clicking on the “SetLock” button. The interface is shown in figure 5.12

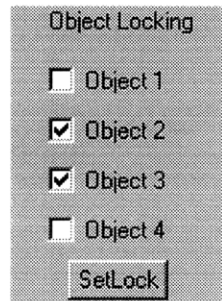


Figure 5.12: Object Locking Interface

The user interface designed for task locking is also embedded in the control panel as shown in figure 5.13. The default value is the null selection, which means that no task locking is established.

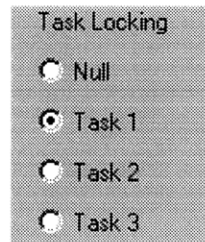


Figure 5.13: Task Locking Interface

Tasks, as expected, has the list of machines and objects to be utilized and therefore the task selection inherently takes care of the hierarchical locking mentioned in the previous section. When a task is selected in the interface, the data file for the machine takes the following form:

On (status) *(username)*
Task 2 *(name of the task selected)*

If another client is trying to select a task or lock an object, then the server simply checks the data files for each machine. If the client is trying to select the same task, or lock an object, which is hierarchy of the task listed in the file, then his request is rejected by the server.

The structure of the data file is presented in the figure below.

On (status) (username)
Task 2 Subtask 3 (the number of the subtask that is being currently executed).

The access time calculation should be calculated at the end of every subtask for each task due to the fact that subtasks might actually take longer than the expected completion time due to a possible hardware or a network error. In addition, the clients executing the tasks are notified by the following text message. "Object 1 is a sharable entity, currently being used by client X "

In a telerobotic control panel, the users should not be restricted in the type of data they should send. They should be given the freedom to pick the type of data, which suits best to their purpose. That is why our interface provides multiple options such as angular data, and Cartesian data (this corresponds to the tip position of the mechanism) to illustrate this principle. In order to provide general functionality for the interface, there are also on and off check buttons so that the interface can also be used to turn on and off simple machines such as an over or a refrigerator. The interface is shown in figure 5.14.

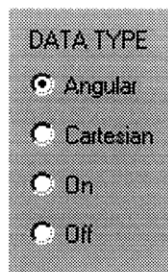


Figure 5.14: Interface Allowing Different Types of Data Input

5.5 Video Image Interface

Video has been the preferred client feedback since the first Internet telerobotics demonstration in 1994. Some applications sent back camera images taken at certain update intervals, like 15 seconds [28]. Other applications, which have more complicated environments transmitted multiple camera images [42]. Recent cameras have the server push capability [19] to perform live image transmission.

There is also research going on to combine virtual reality and video imaging on the same frame. This approach is called Augmented Reality. There are researchers trying two different approaches, either using graphical image to augment a graphical interface or using computer graphics to augment a video display [25].

In every approach, video transmission is a necessity and a major part of the interface. In the interface described in this work, the camera image is not a necessity but rather complimentary, assuring the user that he is really operating the real machines over the Internet. The virtual simulation does exactly what the machine should do in real life, however there might be unexpected changes which the virtual interface that cannot represent properly such as dropping an object. Future work is expected to find solutions to some of these random, error-based motions that might occur in the real system. Another interesting area to investigate is how to create a virtual image of an environment from video input.

The hardware used in our implementation was WebCam 32 V4 [19]. The interface provides a default Java class, namely, `javacam.class` which enables the camera image to be used as an applet on the Internet. This applet obtains the image from WebCam 32 and replaces the previous image after configurable number of seconds have elapsed by using a sleep function inside the thread which obtains the image from the camera.

This file is modified and combined with the client-server interface so that it refreshes the image only if the server gets a request from the client. A request indicates that the machine will change its previous location, and therefore the camera image needs to be updated. After the data is sent to the server, the camera keeps refreshing the image for 15 seconds, this threshold time can easily be changed in the server code, depending on the nature task. This time interval accounts for the possible network delays that might occur during data transmission and the duration of the actual operation. If no operation is being made, the camera displays the static previous image so that the data transmission load is decreased.

Another advantage of the Java applet is that its image refreshing approach is superior to the refresh meta tag approach as only the image will be refreshed and the rest of the web page will be left alone and hence no flicker. Here is how the Java applet is included in html.

```
<TD><APPLET CODE = "javacam.class" width = 250 height = 215><PARAM  
name = "URL" value =http://128.52.37.96:81 ">  
<PARAM name = "interval" value = "1"></APPLET></TD>  
</TR>
```

Parameter value refers to the URL of the address the machine that the camera is connected to. The JavaCam applet must be loaded from the same machine as the target image, or else an error will occur. Java requires that any applet dynamically downloaded from the Internet only connect back to the machine from which it was loaded. The interval value is the multiplication factor in the sleep function in the Java program. Therefore the user can simply change this value from the html file without changing and recompiling the Java code.

```
try  
{  
    // Put the refresh thread to sleep for the supplied interval  
    Thread.sleep(int_Interval*10);  
    // Code to repaint image when data is submitted to the server...  
}  
  
catch (Exception e) {}
```


Chapter 6

6 Conclusion

6.1 Introduction

This chapter concludes the thesis, starting in section 6.2 with a brief review of the work done. Section 6.3 makes prediction on the future developments in this field, and finally section 6.4 concludes with several research avenues that can be taken from this thesis.

6.2 Review

The development of platform independent programming languages and simulation tools in compliance with the current Internet browsers provided the platform to develop a robust multi-user interface for networked machines which are gaining more popularity in the science community since their first applications in 1994.

This thesis work used these programming languages, namely Java 1.1 and VRML97 in order to develop a user interface for networked telerobotics. The simulation has been made by providing a linkage between Java and VRML thorough VRML's External Authoring Interface. Simply stated, VRML served as the media providing 3D environment over the Internet, and Java provided a client-server architecture with an embedded intelligence.

The server-client system provided a locking mechanism so that two users could not send data to the same machine at the same time. Whenever a client logged in to the system, it has been notified about the status of each machine in the system. In addition messages indicating the availability of machine that was being used are provided for the users waiting in the queue to operate machine. The locking mechanism has also been extended to include the objects that these machines are interacting with so that no two machines such as two robots tried to access the object simultaneously.

The client-server system has been programmed to support up to 50 clients, and was tested with up to 14 clients successfully. The server used broadcasting to send data to each client. Due to broadcasting, the load on the server increases linearly with the number of clients, because the server has to send data separately to each client. The future version of the interface is expected to use multicasting to overcome this problem. Multicast can be simply defined as the ability to send one message to one or more nodes in a single operation. Therefore no matter what the number of users in the system is the

network load on the server is at the same minimum level. There is currently an experimental Multicast Backbone, called the MBone [7], which is acting as a test bed for multicast application and protocol design and refinement. The use of multicast will definitely enable the server to distribute the simulation and real operation images to larger number of clients over the network.

The 3D-simulation interface gave the users the advantage of testing their commands before they submitted it to the real interface. For collaborative tasks, involving more than one user, the simulation environment provided a medium where each user simulated his task and observed the task execution of others. The simulation environment is intended to replace the common camera image interface, which requires large data transmission and thus creates problems due to bandwidth limitations. However the camera images were still used only when the task has been executed so that the clients had the feeling that they were really operating real machines over the Internet.

The time delay problem has become evident in multi-user simulations even for users accessing the system from the similar domains such as ai.mit.edu, and mit.edu. Therefore in order to minimize the data submitted over the Internet for complex collaborative tasks, the interface provided a series of pre-defined tasks which are coded into the system. This has allowed the users to execute a series of operation simply by sending a 4-byte integer data. The inclusion of these tasks introduced a hierarchical locking mechanism where a user selecting a task gained the access to all the machines and objects associated with this task.

For collaborative environments, the use of the chat applet has proven to be successful. The clients also had the chance to chat with the system administrator to ask their questions about the interface.

The clients, when asked, found the presence of the camera image, psychologically reassuring. That is to say, they wanted to really observe that they were operating a machine at the remote site. This indicates that future work should come up with an interface that will incorporate the VRML model with the camera image. Currently MPEG4 provides linkage between these two displays. The inclusion of the real image should be kept minimum to decrease the size of data transferred over the network, but still be indicative to the user regarding the real operation.

6.3 Future Developments in the Field

Since "multi-user networked" telerobotics is assumed to be widely practiced in the next coming years, we have to look into the probable technological developments of the future. The selected categories are:

- a) Developments in the Simulation Tools Used (i.e. Java, VRML)
- b) Developments in Network Connections
- c) Developments in PC Hardware.

The developments in the simulation tools are discussed in detail in the appendix.

6.3.1 Developments in Network Connections

Fast Ethernet or 100BASE-T is the leading choice among the high-speed LAN technologies available today. Building on the near-universal acceptance of 10BASE-T Ethernet, Fast Ethernet technology provides a smooth, non-disruptive evolution to 100 MBPS performance. The growing use of 100BASE-T connections to servers and desktops, however, is creating a clear need for an even higher-speed network technology at the backbone and server level. The next level seems to be the Gigabit Ethernet, which will provide 1 Gbps bandwidth for campus networks with the simplicity of Ethernet.

In a graphical simulation environment, as the file size that has to be transmitted over the network increases, a larger bandwidth and therefore a network that is capable of providing this for desktops, and the servers is required.

6.3.2 Developments in PC Hardware

In this field there is a continuous development for increased memory, increased graphic capabilities and increased storage capacity. Increased memory will allow us to make real time computations faster. (This especially becomes important for simulation applications for training purposes in which the user continuously sends real time data back to the simulation, and in return the simulation makes the necessary computations to respond to that.) In addition the simulation will have faster response and update capability to unexpected disturbances from the real system (such as falling of the object from the robot's hand, or an unexpected collision). Besides currently most of the CAD programs started to allow file conversions into VRML so that engineering drawings can be displayed and in future animated over the web, having high ram will help file conversion and animations to become widespread.

Increased graphic capabilities will make the virtual worlds as realistic as possible. One of the leading companies in this field is 3Dlabs, which is specialized in developing 3D graphics processors to meet the demand for 3D on the PC. This means that ordinary desktop PCs will have sophisticated graphics and this will increase the interest for the user interface provided.

As it can easily be seen, these three categories are almost inseparable and will depend on each other most of the time for future development. Here is a simple example that clarifies that. As the software becomes more sophisticated, it will require sophisticated PCs and the PCs interacting with each other will require a sophisticated network.

6.4 Future Work

Networked telerobotics is a broad topic, which is prone to exponential development in the nearest future. Based on the work in this thesis, there are a couple of further research topics, which will motivate many, including me.

6.4.1 Hardware Design for Networked Telerobotics

The concept of operating machines over the Internet is not definitely limited to robots. Considering a chemistry lab, there are numerous instruments that can be operated via web such as a centrifuge station, or a simple pH-meter. There are some automated laboratory applications (give reference), but they still require the presence of a human operator, which defeats the purpose of networked machinery. This brings up the question of how do we use these machines without a single human being. Do we need the classical doors and handles on machines such as a oven or a refrigerator? What is the ideal design of a networked pH-meter? Only after design modifications that will be flexible enough to fit both remote and local operation, will networked automated machinery become practical to operate in large scales.

6.4.2 Programmable Interface Design for Networked Telerobotics

Although the interface developed in this thesis provides a collaborative environment with its simulation tools and locking mechanism, it uses a server-client code, which does not allow the client to make programming changes when necessary. This becomes of crucial importance for space telerobots such as the Sojourner. If a failure occurs during operation, or a situation that has not been considered and programmed into the interface comes up, the supervisory users should have the chance to simply reprogram the server-client architecture and upload it to the remote machine. This feature will function better than any artificial intelligence embedded into the system because the supervisory controller will have the flexibility to intervene and insert “real intelligence” into the machine. This system will work fine as long as these clients are the professional scientists can perfectly debug these “to-be-inserted” programs by knowing the capabilities of their design. However if a machine which provides access to the general community is placed on the Internet, programming should be as simple as

possible. Besides, it should have a debugger and warning mechanism to give feedback to the client before his program is uploaded to the machine. In addition it should prove inter-operability between multiple machines so that the client can include multiple entities in his program. This interface will definitely simplify the interface designs which currently become complicated by trying to present the user with more task options.

6.4.3 Predictive Simulations for Space Exploration

Another future work related with space explorations is to make an intelligent simulation environment, which can gather sensory data from the real world and create a virtual environment. This work is closely related to vision and object recognition studies. For example, a device like Sojourner can be equipped with sensors capable of object recognition, and the gathered data along with prior data can be used to build the virtual workspace of this machine. Afterwards the planned tasks can be executed using this simulation and with the feedback received, future tasks can be adjusted accordingly. However, building a virtual environment just with sensory data gathered from the environment seems far way from being doable. Therefore this sensory data should be supported with supervisory data received from the video images obtained. For example if the space robot can measure the distance to a nearby rock. Based on the video image, its virtual model can be created. The material composure can be estimated from prior data and its weight can thus be approximated. Therefore a preliminary virtual model to simulate the path of the robot or a simple task like bringing the rock back to the space ship can be simulated. The virtual models are continuously updated, as more real world data becomes available.

6.4.4 Failure Detection and Prevention for Networked Machinery

Another issue in networked machinery is to be able to detect and prevent failure so that the machine is available for remote operation at all times. Providing this reliability will eliminate the presence of a human operator at the site for failure recovery. This requires the placement of sensors on the machine that will be capable of detecting the causes of failure and warning the client. The data from the sensor can also be sent to the virtual model where detailed views of the predicted failure location are displayed upon the request of the client. More automated systems are predicted to have remotely controlled machines that will fix the machine by for example changing a malfunctioning part. This research field is expected to become attractive for those working on remotely controlled production in factories where savings in time corresponds to huge savings in cost.

Appendix

A.1 The Virtual Reality Modeling Language

The Virtual Reality Modeling Language (VRML) is a language for describing multi-user interactive simulations, virtual worlds networked via the global Internet and hyper-linked within the World Wide Web.

The latest version VRML 2.0 [1], [15],[23], introduced in 1995, provides the concept of behavior into virtual worlds and puts the static geometry into motion. Currently there are two ways to create animation in VRML 2.0. The first one is called VRML Scripting, which makes use of either Java, or JavaScript languages embedded in the Script node [34].

The second method is the use of External Authoring Interface (EAI) [24], [32], [35]. EAI provides the modification of the static VRML world by means of data input from an external interface such as a Java applet [2],[12],[22], [33]. In the examples shown in the Implementation section EAI has been used to provide the simulations.

A.1.1 Advantages of VRML as a Simulation Tool

Displaying 3D graphics is very different from displaying 2D images. Objects are positioned in three dimensions but must be viewed on a two dimensional computer display. Because the objects exist in three dimensions, it must be decided from which direction they should be viewed. To do this, I will use the analogy of a camera. When you shoot a picture with a camera, you first aim at the desired scene. Then you snap the picture, which places an image of the three dimensional scene onto the flat surface of the camera film. Similarly, a scene in the computer has a camera that is placed by the author to view the desired part of the scene. Then the computer draws what the camera sees onto the two dimensional computer display. This operation is known as rendering.

This is a very powerful notion. To look at the scene from any angle, you simply place your virtual camera at a different viewpoint, render the image, and show it onscreen. If you move the camera little by little and render the image again at each step, you can create the walk-through of your scene. This is something you could never do with a 2D image, and it is part of the power of 3D. But this operation is as difficult as it is powerful. Rendering quickly enough to make the movement fluid requires a powerful computer. Faster computer hardware has made it possible for PCs to do a good job of rendering simple scenes.

Embedded with this feature, the users can walk around the generated VRML scene and observe it from multiple viewpoints. The other advantages of using VRML as a simulation tool are listed as follows:

- VRML is platform independent, standard way to describe 3D objects that can be used on the Internet or locally.
- ISO has adopted VRML as a standard Virtual Reality tool for the Internet.
- Window 98 will come with a VRML Browser. Netscape and Internet Explorer are putting plug-ins for VRML therefore users will not go through the trouble of downloading the plug-ins themselves. This will expose the internet users to VRML at an enormous rate because currently whenever these users come up to a page with VRML such as a movie advertisement that lasts 15 seconds, they do not go through the trouble of selecting a specific browser and downloading 6MB just to enjoy a VRML movie. This widespread usage of VRML will also give the end user some experience to move through VRML worlds.
- In the next version , VRML 3.0, downloading worlds will become faster because input streams will have a binary format.
- Apart from its interface with Java, it has built in commands that enable multi-user capabilities (Shared Objects), and good multi-user browsers started to show up in the market.
- Multithreading capability of Java enables simultaneous animations with different animation speeds. A thread becomes responsible for each animated object. The system itself will uniformly distribute the load amongst different threads. However you might hit the other extreme. The time spent in setting up a thread is not negligible – taking the time to set up hundreds of threads might be a hindrance rather than a benefit.
- User has the freedom to explore the 3D environment. If you are working with multiple machines you immediately learn the environment they are working in. Besides it would require multiple cameras to represent a 3D environment and this would impose bandwidth restrictions. Besides it has these pre-defined viewpoints
- The embedded parent-child node relationship enables the digital control of complex robotics mechanisms in a very simple manner. The kinematics matrices are taken care of by defaults. All the user has the input are the relative or absolute link angles or positions. (We can support this by presenting a source code for one of our robot demos)

- VRML has the capacity to encapsulate and organize many media types, including image, animation, and 3D audio.

Since VRML enables large amounts of information to be displayed with a 3D graphical interface, people incorrectly conclude that VRML must be a bandwidth hog. This myth often stems from the fact that in some of the worlds on the web the content has been converted from some other 3D format to VRML inefficiently. Native VRML is considerably more efficient than other media types such as QuickTime movies and even static images. For example, 20sec animation in VRML might only be only about 50k, while that same animation might weigh several megabytes were it created as a GIF or MPEG file. This is because VRML worlds are rendered locally. VRML is text at download time, not images as in a QuickTime movie or a bitmap file. With this smaller content footprint, VRML enables quicker download time.

A.2 Network Specifications

The communication protocol used in the server-client design is TCP/IP. TCP/IP stands for “Transport Control Protocol/Internet Protocol”[9]. TCP and IP are really two different protocols, but together govern the basics of the Internet. TCP is all about ensuring that data is transmitted correctly between two computers. If any errors occur, these are detected and data is retransmitted. If data sent to a particular machine had to be divided into smaller pieces, called data packets, and sent separately, all of these packets are reassembled on the receiving end in the correct order. The Internet Protocol (IP) uses a four-byte value to identify computers over the Internet, this value is called the IP address. In addition to the IP addresses, there is one more piece of information that distinguishes where the data should go, which is the port number.

By convention port numbers are divided into two groups. Port numbers below 1024 are reserved for well-known system uses such as Web protocols, email, ftp, and so on. The numbers above 1024 up to 2^{16} minus 1 are all left to the user who can assign personal programs and services to these port numbers.

The combination of IP address and port number uniquely identifies a service on the machine. Such a combination is known as a socket. A socket identifies one end of a two-way communication. When a client requests a connection with a server on a particular port, the server identifies and keeps track of the socket it will use to talk to the client.

Even though the server is communicating over the same port with many clients, it uses these sockets to determine the destination and source of that communication. The figure A.1 illustrates this concept.

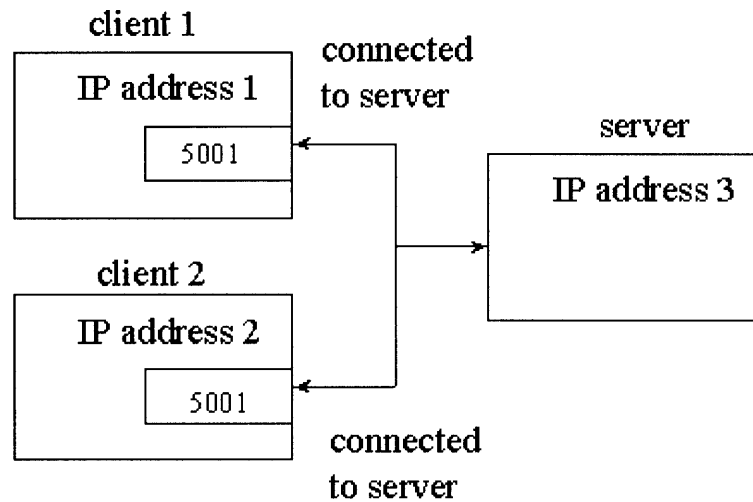


Figure A.1: Server – Client Communication via Sockets

The server developed listens for connections on a particular port; when it receives a connection, it obtains a socket to use for the communication and begins a dialog with the client at the other end of the socket. The client initiates the connection and communicates with the server via this socket as shown in figure A.2.

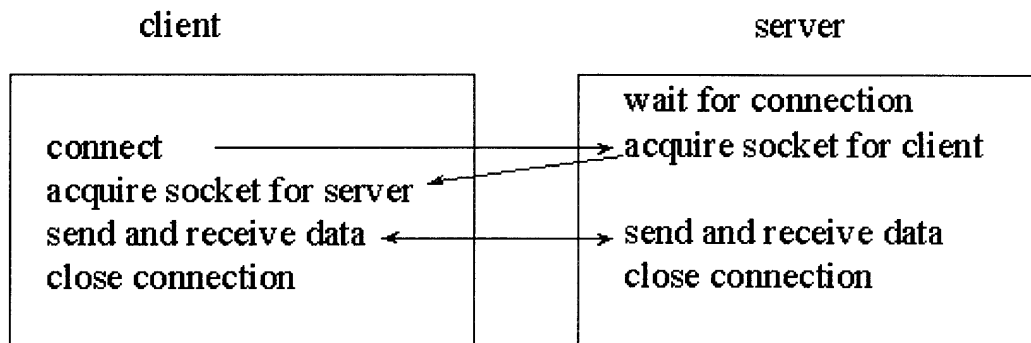


Figure A.2: Basic Interaction between a Client and a Server

The multi-user client server architecture is based on the very same principle with the additional use of threads to handle communication between the server and each client.

The term thread is the shorthand for thread of control, which is simply the path taken by a program during execution. This determines what code will be executed: does the code in the *if* block gets executed, or does the *else* block? How many times does the *while* loop execute? If we were executing tasks from a “to do” list, much as a computer executes an application, what steps we perform and the order in which we perform them is our path of execution, the result of our thread control. Having multiple threads of control is like executing tasks from a number of lists. That is how the server is set to handle different requests of multiple clients in multi client-server system. The basic flow chart of the multithreading server is shown in the figure A.3.

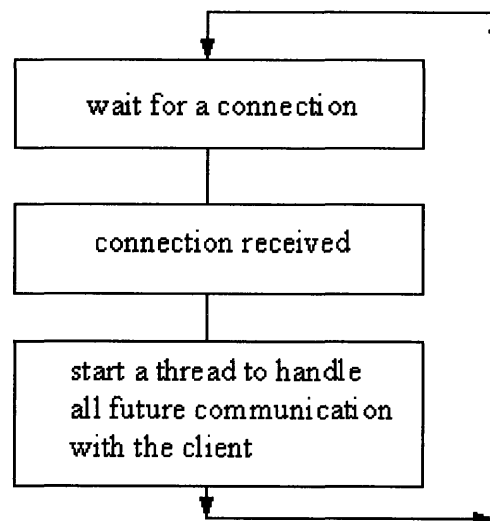


Figure A.3: Java Algorithm Handling Multiple Clients

Theoretically the number of clients that can connect to a server is unlimited but we can set a hardcore limit within the code, 50 in our case, with the intent of ensuring the server does not get swamped with requests. Otherwise, the server might get bogged down with keeping track of lots of network connections and performance could become slow.

References

- [1] Ames, L. Andrea, et al. *VRML 2.0 Sourcebook*. New York, NY: John Wiley & Sons, Inc. 1997.
- [2] Boone, Barry. *Java Certification Exam Guide for Programmers and Developers*. New York, NY: McGraw-Hill. 1997.
- [3] Brock, L. David. "A Sensor Based Strategy for Automatic Robot Grasping," Ph.D. Thesis, ME, MIT. 1993.
- [4] Brock, L. D. "Remote Controlled Machinery, Processes Over the Web Closer than You Think." *The MIT Report* 25.7(1997): 1-2.
- [5] Brutzman, Don, and Don McGregor. "DIS-Java-VRML: Streaming Physical Behavior into VRML Worlds." VRML 98 Third Symposium on the Virtual Reality Modeling Language Lecture Series, Monterey, CA. 17 Feb. 1998.
- [6] Brutzman, Don (1998, February 15). *Distributed Interactive Simulation, DIS-Java-VRML Working Group*. [WWW document]. URL <http://www.stl.nps.navy.mil/dis-java-vrml/>.
- [7] Byne, Magnus, et al. (n.d./1997) *An Introduction to IP Multicast*. [WWW Document]. URL <http://ganges.cs.tcd.ie/4ba2/multicast/>.
- [8] Campbell, Bruce. "3D Multi-User Collaborative Worlds for the Internet," M.S. Thesis, University of Washington, 1997.
- [9] Comer, E. Douglas, and David L. Stevens. *Internetworking with TCP/IP, Client Server Programming and Applications*. Upper Saddle River, NJ: Prentice-Hall, Inc. 1997.
- [10] Durlach, I. Nathaniel, and Anne S. Mavor, eds. *Virtual Reality Scientific and Technological Challenges*, Washington, D.C.: National Academy Press. 1995.
- [11] Ellis, R. Stephen. (n.d./1997). *Virtual Environments and Environmental Instruments*. [WWW document]. URL <http://duchamp.arc.nasa.gov/papers/veei/veei.html>.
- [12] Flanagan, David. *Java in a Nutshell*. Sebastopol, CA: O'Reilly & Associates, Inc. 1997.

- [13] Goldberg, Ken. (1996). *The Telegarden*. [WWW Document] . URL <http://www.usc.edu/dept/garden.html>
- [14] Gossweiler, Rich, et al. "An Introductory Tutorial for Developing Multiuser Virtual Environments." *Presence* 3.4 (1994): 255-264.
- [15] Intervista Software, Inc. (1998). *Virtual Reality Modeling Language*. [WWW Document]. URL <http://www.intervista.com/vrml/index.html>
- [16] Harold, R. Elliotte. *Java Network Programming*. Sebastopol, CA: O'Reilly & Associates, Inc. 1997.
- [17] Hoehnen, Dan. *Home Automation Index*. [WWW Document]. <http://www.infinet.com/~dhoehnen/ha/list.html>.
- [18] Isdale, Jerry (1993, October 8). *What is Virtual Reality?*. [WWW Document]. URL <http://www.cms.dmu.ac.uk/~cph/VR/whatisvr.html>
- [19] Kolban, Neil (1997, August 19). *Webcam32 and Java Applets*. [WWW Document]. URL <http://www.kolban.com/webcam32/java.htm>
- [20] Laguna, Glenn (n.d/1997). *Virtual Reality is ...* [WWW Document]. URL <http://www.sandia.gov/2121/vr/vr.html>.
- [21] Landis, Rob (n.d/1997). *Overview of the Hubble Space Telescope*. [WWW Document]. <http://opposite.stsci.edu/pubinfo/HSToverview.html>
- [22] Lea, Doug. *Concurrent Programming in Java, Design Principles and Patterns*. Reading, MA: Addison Wesley Longman, Inc. 1997.
- [23] Marrin, Chris, and Bruce Campbell. *Teach Yourself VRML 2 in 21 Days*. Indianapolis, IN: Sams.net. 1997.
- [24] Marrin, Chris (1996, August 2). *External Authoring Interface Reference*. [WWW Document]. URL <http://reality.sgi.com/cmarrin/vrml/externalAPI.html>.
- [25] Milgram, Paul, and Anu Rastogi. (n.d/1997). *Augmented Telerobotic Control: A Visual Interface for Unstructured Environments*. [WWW Document]. URL http://vered.rose.utoronto.ca/people/anu_dir/papers/atc/atcDND.html
- [26] Mosteller, Frederick, Robert Rourke, and George Thomas. *Probability with Statistical Applications*. Reading, MA: Addison-Wesley: 1970.
- [27] Natonek, E., et al. "Virtual Reality: an Intuitive Approach to Robotics" *Telemanipulator and Telepresence Technologies* Vol. 2351 (1994): 260-269.

- [28] O'Brien, M. Kevin. "Task-level Control for Networked Telerobotics, " M.S. Thesis, ME, MIT. 1996.
- [29] Oaks, Scott, and Henry Wong. *Java Threads*. Sebastopol, CA: O'Reilly & Associates, Inc. 1997.
- [30] Peck, B. Susan, and Stephen Arrants. *Building Your Own Website*. Sebastopol, CA: O'Reilly & Associates, Inc. 1997.
- [31] Pitman, Tim. *Probability*. New York, NY: Springer-Verlag. 1993.
- [32] Regan, Tim. "Taking Living Worlds Into Peoples Living Rooms" VRML 98 Third Symposium on the Virtual Reality Modeling Language (1998): 71-76.
- [33] Rice, C. Jeffrey, and Irving Salisbury. *Java 1.1 Programming*. New York, NY: McGraw-Hill. 1997.
- [34] Roehl, Bernie, et al. *Late Night VRML 2.0 with Java*. Emeryville, CA: Ziff-Davis Press. 1997.
- [35] Roehl, Bernie. "Channeling the Data Flood." IEEE Spectrum 34.3 (1997): 32-38.
- [36] Roehl, Bernie (n.d/1997). *Distributed Virtual Reality-An Overview*. [WWW Document]. URL <http://sunee.uwaterloo.ca/~broehl/distrib.html>.
- [37] San Diego Computer Center(SDSC). (n.d/1997). *The VRML Repository*. [WWW Document]. URL <http://www.sdsc.edu/vrml/>
- [38] Sheridan, B. Thomas. *Telerobotics, Automation, and Human Supervisory Control*, Cambridge, MA: MIT Press. 1992.
- [39] Stark, W. Lawrence, and Theodore T. Blackmon. "Model-Based Supervisory Control in Telerobotics." Presence 5.2 (1996): 205-223.
- [40] Stiles, Randy, et al. "Adapting VRML for Free-form Immersed Manipulation" VRML 98 Third Symposium on the Virtual Reality Modeling Language (1998): 89-94.
- [41] Story, David, Delle Maxwell, and Geoff Brown. "Authoring Compelling, Efficient VRML 2.0 Worlds." VRML 98 Third Symposium on the Virtual Reality Modeling Language Lecture Series, Monterey, CA. 16 Feb. 1998.

[42] Taylor, Ken, and Barney Dalton. (1997, December 8). *Issues in Internet Telerobotics*. [WWW document]. URL <http://telerobot.mech.uwa.edu.au/robot/anupaper.html>

[43] Turoff, Murray. "Virtuality." *Communications of the ACM* 40.9 (1997): 38-43.
Wagner, G. Michael. "Advanced Animation Techniques in VRML 97." VRML 98 Third Symposium on the Virtual Reality Modeling Language Lecture Series, Monterey, CA. 16 Feb. 1998.

[44] Waters, C. Richard, and John W. Barrus. "The Rise of Shared Virtual Environments." *IEEE Spectrum* 34.3 (1997): 20-25. 1996

[45] Vogel, Jorg.(Maint.) (1997, October 23). *VRML and Robotics Library*. [WWW Document]. URL http://dv.op.dlr.de/FF-DR-RS/STAFF/joerg_vogel/Vrml/lib.html