AN APPROACH TO ENHANCE A TRADITIONAL ERGONOMICS TOOL WITH ASSEMBLY CAPABILITIES AND ALGORITHMS FROM AN IMMERSIVE ENVIRONMENT

By

OKJOON KIM

A Thesis submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

WASHINGTON STATE UNIVERSITY
School of Mechanical and Materials Engineering

May 2007

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of OKJOON KIM find that it is satisfactory and recommend that it be accepted.

_____

Chair

_____

_____

ACKNOWLEDGMENTS

First, I am deeply grateful to the School of MME and PACCAR Inc. for providing me with this valuable opportunity for my research.

Especially, I would like to thank my advisor, Dr. Uma Jayaram, for constant attention, helpful guidance, and financial support. I also want to express my appreciation to Dr. Sankar Jayaram for continuous helpful advice and to Dr. Charles Pezeshki for serving on my committee. It has been fortunate that I could study and do research related to Virtual Reality in the Virtual Reality and Computer Integrated manufacturing Lab at WSU. In addition, I am thankful that my best friend, YoungJun, gave his hearty recommendation for me to study at Washington State University and thereby extend my experience. I also appreciate the intimate support and help from all the members in VRCIM lab.

Lastly, I would like to specially thank my wife, MiYoung for endless love, firm trust, and care. Furthermore, I am able to make a successful life because my parents, brother, and sister, and my wife's family trust me with confidence. I would like to thank all my friends in Korea and Pullman.

AN APPROACH TO ENHANCE A TRADITIONAL ERGONOMICS TOOL WITH ASSEMBLY CAPABILITIES AND ALGORITHMS FROM AN IMMERSIVE ENVIRONMENT

ABSTRACT

By OkJoon Kim, M. S.
Washington State University
May 2007

Chair: Uma Jayaram

This thesis presents an approach to link traditional, commercially available ergonomics evaluation tools with virtual environment tools for providing enhanced capabilities for engineering design. Ergonomic evaluation tools in engineering design are fairly mature and are used in important and specific ways to analyze human model postures in industry. The promising capabilities of immersive environment tools for assembly simulations such as realistic environments and interactions, constraint-based modeling, and physically-based modeling are attractive to industry but have so far been available only in environments separate from the traditional ergonomics analysis tools. This research seeks to create a well-integrated synergistic approach that will complement traditional ergonomics tools with a careful assimilation of capabilities and algorithms

from a virtual environment used for assembly simulations. The information exchange, representations, communication, and computational issues involved in achieving this connectivity are discussed in this thesis. A successful implementation was created and demonstrated. It is anticipated that this synergy between an ergonomics tool and a virtual environment will lead to breakthroughs and ease of use benefits similar to those that have now been obtained by the close integration of CAD and virtual environments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

An important tool used frequently in engineering design is the Ergonomics evaluation Tool, coined EGT in this work. This tool is particularly important for certain sectors such as the transportation and trucking industry. It is used primarily for the purpose of analyzing human performance such as posture, comfort, visibility, and accessibility. Human modeling itself is a complex and time-consuming task, because a human consists of many joints and segments and kinematics or constraints. Commercial ergonomics software such as JACK^{TM} and RAMSIS^{TM} provide a user-friendly interface to create virtual humans and to perform powerful analyses.

An immersive environment tool, coined IMT in this work, is a powerful planning, designing, and evaluation tool in the manufacturing and industrial sectors. The combination of a realistic environment and realistic interactions in a dynamic context provides important and unique design and evaluation capabilities. Initially, IMTs were created in stand-alone niche applications.

However, in traditional engineering design, the CAD model has been the master model and the CAD system has been the core component in the suite of product realization tools. Today, there are several examples of work in embedding IMTs in commercial CAD (Computer Aided Design) systems and documented benefits from this integration. Commercial systems such as CATIA even routinely provide this capability now. There is merit in learning from this and seeing how it applies to the use of IMTs and EGTs in engineering design.

While EGTs have a number of strengths in rendering multiple human models more easily and evaluating human performance more accurately, IMTs play an important role in displaying environments inside the immersive environment and providing realistic interactions. Considering the EGT and IMT systems separately is similar to the "over the wall" approach referred to in design/manufacturing. All the data is transferred manually between the two systems and analysis is performed sequentially. The concurrent use of the two tools that leverages the strengths of each, without significant overhead in either application, has the potential to provide some distinct new capabilities. Thus, instead of considering the ergonomics analysis tools and the immersive environment tools as being "isolated silos", there is merit to considering an integrated and synergistic capability [Figure 1].

Research in the VRCIM laboratory over the past few years has investigated preliminary methods to synchronize the two systems and make it possible to carry out the real-time ergonomic analyses immersed in an IMT system. Some of the work has investigated approaches to address this issue of integration of ergonomics analysis tools with virtual environments. However, that work was IMT-oriented and the user had access to ergonomic modules while all the time *working in the IMT environment*.

In contrast to this previous work which was IMT-oriented, the approach investigated and implemented in this thesis will be EGT-oriented. Specifically, the motivation of the work presented in this thesis is to consider and evaluate approaches to integrate carefully selected capabilities from the immersive environment into an ergonomics tool, thus enabling accurate measurement of human performances during assembly operations *being performed in the EGT environment.* This will allow the

monitoring of the postures during each frame automatically and will provide for a continuous stream of analysis data.

This thesis will briefly discuss the previous relevant work in the laboratory that forms a foundation for the current work, provide motivation and objectives for a new approach, and provide details of the new approach. An implementation is discussed, along with experiments and comparisons. Advantages and limitations are presented.

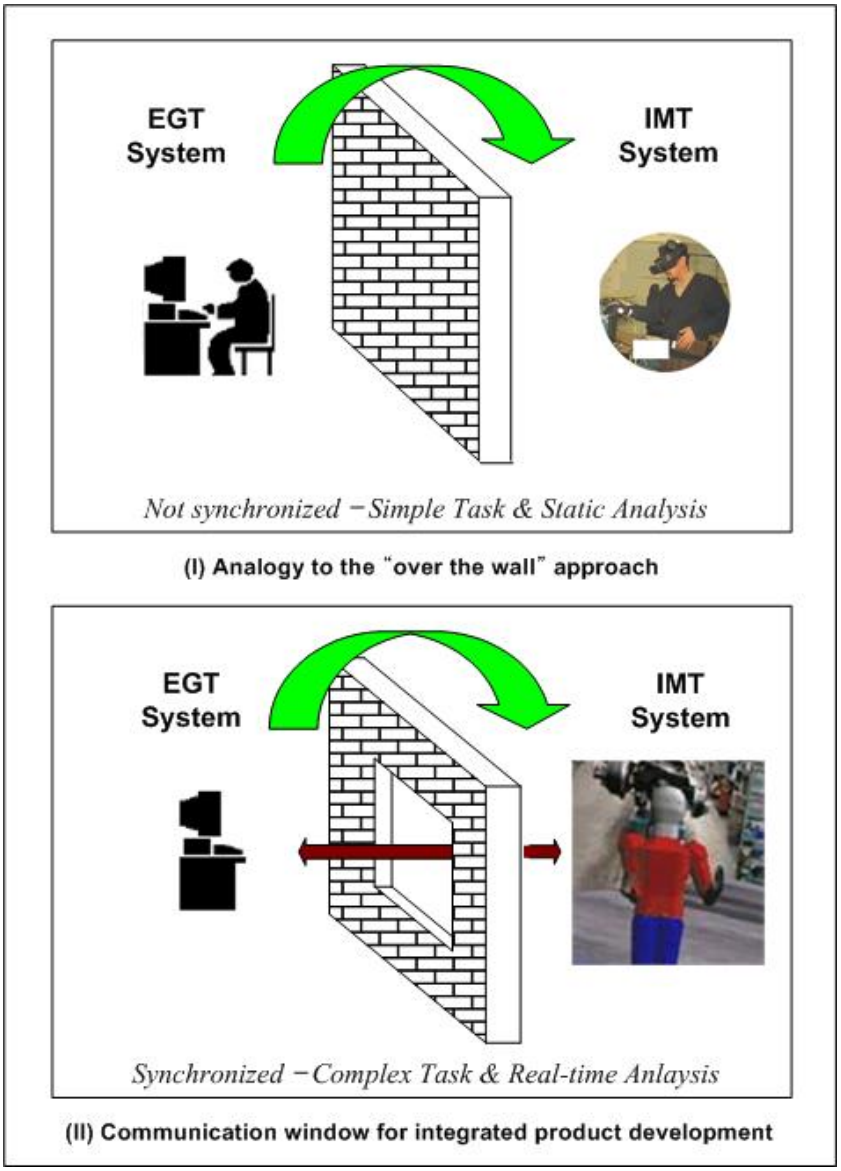

Figure 1 : Integration between EGT and IMT

CHAPTER TWO

BACKGROUND AND LITERATURE REVIEW

This chapter provides the background that supports the work in this thesis with emphasis on the work done at the VRCIM lab at WSU.

2.1. Ergonomics Evaluation Tools

Industrial ergonomics considerations are important to decrease occupational or job-related injuries and illnesses and increase human performance. The Injuries, Illness, and Fatalities(IIF) program of the U.S. Department of Labor has documented a number of U.S. workers that were exposed to occupational or job-related injuries and illnesses [1]. Some of the ergonomics studies have focused on human motion during a work activity. For example, in one study, information was captured related to the right-arm motion of a diverse group of about 3000 participants in the workplace and illustrated how stature, age, and gender have effects on reach and postures [2]. Some studies have approached the human motion modeling and analysis statistically while others have studied accessibility issues of various digital humans with different anthropometric[3],[4]data.

Ergonomics human modeling tools are primarily aimed at helping users in industrial sectors to create models of the environment and the humans, to assign certain tasks to the human, and to analyze human performance in the workplace. JACK^{TM} is one of the well-known ergonomics applications and enables users to locate fully-scaleable digital humans in an environment and improve the ergonomics of product designs and workplace tasks[5]. It builds upon a basic human model which is composed of 71 segments, 69 joints, and 135 degrees of freedom. Jack also provides a Motion Capture

Toolkit to configure and use virtual reality devices. However, it has been observed that loading the entire virtual environment in Jack slows down the application. RAMSIS™ (Realistic Antropological Mathematical System for Interior-comfort Simulation) is also a prominent ergonomics application and is developed by the Human Solutions Corp. in Germany. This software is a 3-D ergonomics CAD-tool for designing and analyzing vehicle interiors as well as working places. Similar to JACK™, it allows realistic visualization of body data and an efficient analysis of the human visibility and comfort and ergonomics formulations[6].

Figure 2 : Various Ergonomic Applications in Industry Sectors [5]

These tools also provide powerful capabilities to allow customization and access. For example, JACK™ provides a tool, called JackScript which is an interface to manipulate the behaviors of objects and adjust the relationship among objects. JackScript

is equipped with python-based API (Application Programming Interfaces) libraries and provides a link between JACK^{TM} application and C++ libraries. Python[7, 8] language is powerful for accessing well-developed resources such as the shared memory.

Research has shown that immersive environments are powerful tools [9, 10]. One of the applications is for presenting ergonomics analyses [11]. Whitman et. al. performed a task of moving a box in the virtual environment and compared lateral, sagittal, and twist velocity/acceleration data between real and virtual environments [12]. ERGONAUT is a tool that has been used for evaluating reach envelopes, visual fields, and comfort of drivers using an existing tractor driver's cabin as a model[13]. Researchers have experimented on how important interactive devices can be selected for measuring human motion in the virtual environment [14]. Realistic rendering and interactions is an important goal. Two important considerations have emerged to represent more realism in simulations, especially assembly simulations - constraint-based modeling and physically-based modeling. Constraints are important in assembly design because the assembly of two sub-parts can be established when the assembly relationship between them including contacts and alignments are fully satisfied. Fernando et. Al. [15] depict two constraint-based approaches: Equation-based approach and Geometric Constructive approach. An extensive survey of various types of constraints in assembly was done by Jones et al. [16]. Physically-based modeling enables physical characteristics to be incorporated into models and allows numerical simulations of their behavior. Focusing on how bodies move and change shape over time, it is applied to animation production, scientific visualization, and teleological modeling [17].

2.2. Immersive Environment Tools and Enhancements with Ergonomic Algorithms

The immersive environment tool that we have developed and have used in this research is VADE [18]. VADE has the capability to import models and constraints from CAD tools and makes the assembly process available through the combination of axis and plane constraints [19]. Physically-based modeling is used in VADE for representing the free motion in space, sliding on a plane on along an axis, and rotation about an axis [19]. It is a fairly mature tool [20, 21]. We have worked on methods to integrate CAD and immersive systems and had a fully functional prototype about 10 years ago [11, 22-25].

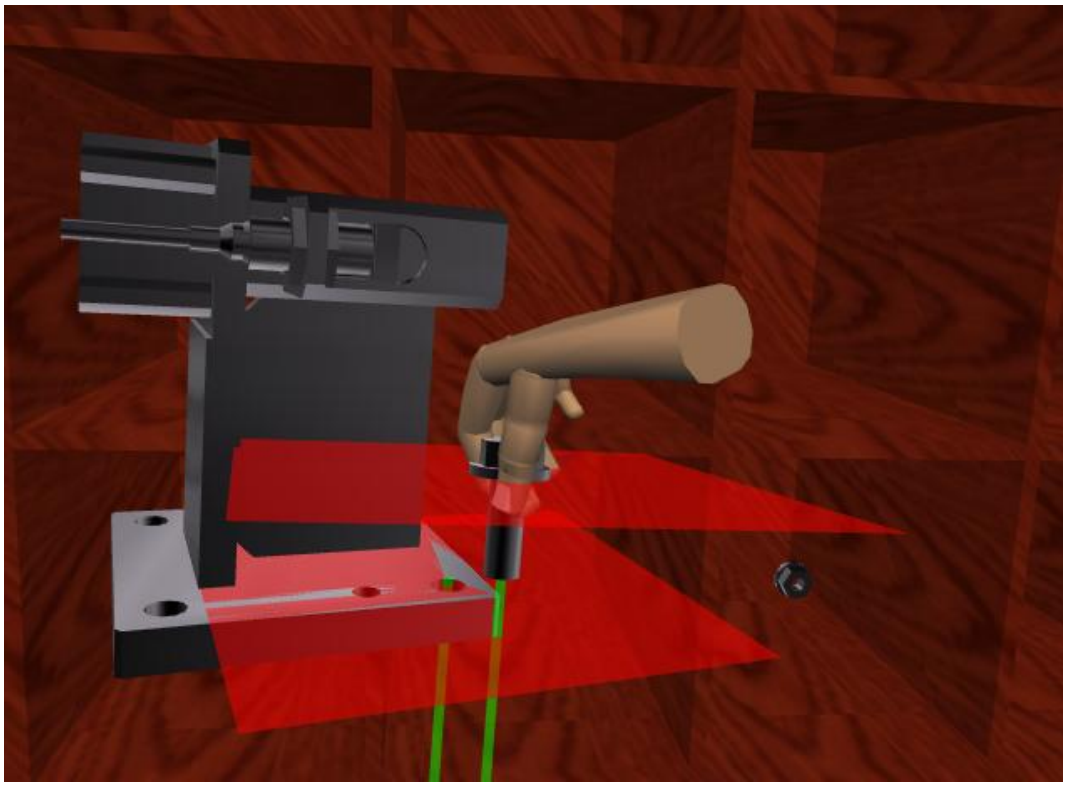

Figure 3 : An Assembly Simulation in VADE

Realistic rendering technique is a common goal in computer graphics. Two main principles have emerged to represent the realism in simulating the assembly process: constraint-based modeling and physically-based modeling. Constraint-based modeling is

an important factor in assembly design because the assembly of two sub-parts can be established when the assembly relationship between them including contacts and alignments are fully satisfied. Fernando et al. [15] depict two constraint-based approaches: Equation-based approach and Geometric Constructive approach. An extensive survey of various types of constraints in assembly was done by Jones et al. [16]. VADE has the capability to import the constraints from CAD tools and makes the assembly process available through the combination of an axis constraint and a plane constraint controlled by the type of constraints [26].

Physically-based modeling enables physical characteristics to be incorporated into models and allows numerical simulations of their behavior. Focusing on how bodies move and change shape over time, it is applied to animation production, scientific visualization, and teleological modeling [17]. Physically-based modeling is used in VADE for representing the free motion in space, sliding on a plane on along an axis, and rotating about an axis [26, 27].

For the past few years one of our focus areas has been to investigate methods to integrate ergonomics evaluation tools and virtual environment tools. Our initial approach was to make the solution IMT oriented. To achieve this, we first integrated an important algorithm used for ergonomics analysis, RULA, the rapid upper limb assessment algorithm, with VADE [28]. In this context a parametric human model was embedded into the immersive assembly system. The intent was to overcome the limitation of current commercial ergonomics systems that gave only a static analysis since the user had to define the posture and the tool would then analyze that static posture to check if the posture could lead to injury. Our approach was to track the position and orientation of the

different parts of the human that are of interest and feed these continuously to a RULA algorithm to evaluate the posture continuously as the user goes through the steps in a task. There were obvious limitations such as restricted movement because of the VR devices. However, it was very successful in demonstrating the capability to provide a continuously changing quantitative score that could be used to identify potential problem areas in an easy.

We then went a step further and integrated our virtual environment directly with the RULA functionality in the commercial ergonomics tool JACK^{TM}, instead of having a stand-alone module for the RULA algorithm [29]. The two strategies i.e. creating a built-in ergonomics analysis module in the IMT vs. creating methods to integrate the IMT with the required algorithms and functionality in EGT were evaluated using case studies and demonstrated the new emphasis of using COTS solutions for new and synergistic capabilities [30].

2.3. Integrated Collaborative Environment System

D-VADE (Distributed Virtual Assembly Design Environment) inherited from the initial VADE is an advanced application to design a collaborative virtual environment. The dramatic development of Internet technology in modern society has allowed the connection of computers over a network and allows users to interact with them in real time and share the same virtual world [31]. This addressed a need to have an application run on multiple computers over the network so that users at each computer are able to use the system to perform the assembly in the virtual environment. Also, this allows the distribution of the load of computationally expensive modules that are required for

simulating the virtual world among a set of less powerful computers. This system is based on CORBA (Common Object Request Broker Architecture) which is a transparent, platform-independent specification of an architecture and interface that allows applications to interact with distributed objects each other [32].

Gowda et al. introduce four types of architectures for internet-based collaborative virtual prototyping – Product development approach, CAE tools integration approach, User session approach, and Functional And black-box approach [33]. The Virtual Design and Manufacturing (VDM) architecture is mainly derived from the user session approach and contains significant contributions from the Product Development Approach and the CAE Tools Integrations approach. This architecture made an effort to provide collaborative, distributed environment by integrating a VR system, a Human Modeling system and a Visualization system. VDM took advantage of OOP(Object-Oriented Programming), CORBA, and Java programming [31].

Research related to the integration between the virtual assembly system (VADE) and an ergonomic tool (JACK) to provide better collaboration was carried out at the WSU VRCIM laboratory [29]. This research was aimed at interchanging the data between two applications using the shared memory mapping technique under a single machine.

2.4. Required Technologies

Python® [8] is a dynamic object-oriented programming language that can be used for software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and is relatively easy to learn and implement. Many Python programmers report substantial productivity gains and feel the

language encourages the development of higher quality, more maintainable code**.** Even though Python might not be as fast as compiled languages such as C or C++, it is “an interpreted, object-oriented, high level programming language with dynamic semantics” [7]. JACK software provides a tool, called JackScript, which is an interface to manipulate the behaviors of objects and adjust the relationship among objects. JackScript is equipped with python-based API (Application Programming Interface) libraries and provides a link between the Jack application and C++ libraries which is very useful for accessing well-developed resources such as the shared memory.

CHAPTER THREE

PROBLEM STATEMENT AND PROPOSED SOLUTION

3.1. Problem Statement

Ergonomics evaluation tools (EGTs) provide methods to evaluate products, measure human performance, and analyze workplaces, but these techniques are not always appropriate for a dynamic simulation. For example, consider an assembly process to put two components together. An EGT does not have a way to interpret assembly relationships between parts, nor does it understand what an assembly hierarchy is. There are no modules to check constraints between objects. For instance, suppose a user has two parts and wants to insert the shaft part into a hole in the other part. The radius of the shaft and the radius of the hole are important considerations and so is the axial alignment. Unfortunately ergonomics tools do not have functions and behaviors enough to resolve issues such as these. Therefore, particular algorithms or interfaces will need to be embedded into the ergonomics tool to help specific assembly evaluations. Some of these algorithms that come to mind are those for gripping or releasing a part, constraint-based modeling, physically-based motion, and collision detection.

VR devices involve tracking devices and gloves to locate digital humans or components. Even though EGTs, e.g. JACK^{TM} provide additional modules to directly configure VR devices and use their features, an independent module for VR devices is needed because of data requirements between the EGT and IMT.

The primary objective of the research presented in this thesis was to build an assembly design environment within an ergonomics/human modeling system to assist in

the evaluation of ergonomics issues during assembly planning. Instead of creating all the new functionalities in the EGT, we deemed it advantageous to use existing algorithms and functionality from an IMT. There are distinct savings of time and development effort. However, this demands an efficient and harmonious integration approach and architecture. To integrate these distinct systems, a few of the aspects that need to be considered are as follows.

- How can we apply algorithms needed for assembly evaluation, such as gripping, physically-based modeling, and constraint checking, from an external IMT application in the EGT application?
- What approaches should be chosen to build a common environment and distribute the data related to models, virtual devices, and interactions with the EGT as the front-end application?
- What are the architectures that would provide efficient ways for the information exchange and coordination between the two applications?
- How can we develop an interface to connect the virtual devices such as cyber-glove and Flock-of-birds and share the data obtained from these devices to both applications?
- How can we share the data of the IMT and EGT over the internet?
- How can we synchronize the coordinate systems between the two applications?
- Will this combined system be extensible? Is it easy to plug/add new algorithms into the existing system?
- What advantages and disadvantages do these approaches have?

3.2. Proposed Solution

The overall approach starts with identifying what exactly are the desired assembly simulation capabilities that are currently not available in EGTs. EGTs do not have certain algorithms to support realistic assembly task simulations. For instance, algorithms such as the constraint-based modeling and collision-detection are needed for gripping or releasing a part and to assemble parts. In addition, the functionality that is being targeted is the ability to handle constraints between parts and the ability to move parts realistically using dynamics simulation and gravity effects. These functionalities are available in IMTs.

As described in the problem statement, our previous approach was to have an IMT-oriented solution where algorithms from the EGT were integrated into an IMT. In the research presented in this work, we reversed the situation and pursued an EGT-oriented approach. *The user interacts with the EGT*. However, behind the scene, the IMT will be responsible for solving mathematical or physical algorithms used in the assembly process simulation and sending the resulting information to EGT continuously. The EGT would update the display of the environment and components based on this information.

To accomplish this goal, a message-driven method was chosen. That is, all messages use well-defined information such as events, states, and transformations. The EGT would be used as the interface with the user. The user would interact with the ergonomics evaluation tool to participate in assigned tasks, to visualize the task process graphically, and to evaluate the ergonomics of products or workplaces while the IMT would offer useful algorithms needed for virtual assembly processes simulation and would control and coordinate the entire state of the assembly parts and processes.

Figure 4 displays an EGT that is communicating with an IMT and VR peripheral

devices around the shared immersive environment. From the diagram, we can see that all the systems communicate with each other on the basis of the shared environments.

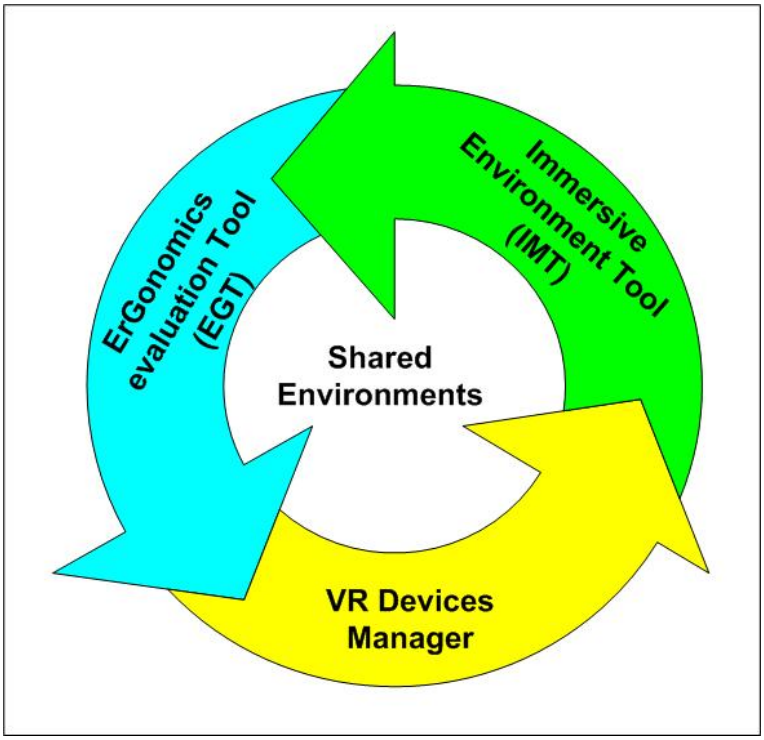

Figure 4 : Harmonious Blending of Different Technologies

In the EGT-oriented approach it is important that the environment in both systems (the EGT and the IMT) be equivalent and coordinated at all times. Each application has assembly parts, tools, and human model located and oriented independently. Techniques need to be devised to share and coordinate this data among the two systems (or among multiple systems in the case of an EGT-distributed system). Usually TCP/IP communication and shared memory mapping are two standard and sufficient methods to achieve this goal. As a network protocol, TCP/IP can be used for communicating between applications either on the same machine or on remote machines. On the other hand, shared memory mapping can connect one application to another only locally and multiple applications can access the memory in the same location of the system at the same time. However, shared memory mapping has the advantage of being much faster than the

network communication.

Thus, the fact that the systems being integrated can be on a single machine or be distributed over the Internet is also an important consideration. Furthermore, the operating system for each application may be different. For example, the immersive simulation system can be running either on a Windows-based machine or on a Unix-family machine. In our work, the EGT runs on a Windows-based machine.

A variation of this approach is to provide a method and architecture for a distributed setting with multiple EGTs collaborating. In this case, there are two EGT applications at different machines and two persons are involved in the same task. For example, two people may be working on an assembly simulation for an assembly task that needs two people. These multiple EGTs need to communicate with the supporting IMT.

Approach	Distinguishing Features
EGT-Oriented (Approach proposed in this thesis)	• Front-end EGT/ Back-end IMT • Bring some of the assembly manipulation and simulation algorithms from IMT in EGT • Have full access to the ergonomics analysis algorithms in the EGT
IMT- Oriented (Previous work)	**• Front-end IMT/ Back-end EGT • Bring some of the ergonomics analysis algorithms from EGT into IMT • Have full access to the assembly manipulation and simulation algorithms in the IMT**
EGT-Distributed (Future work)	**• Extension of simple EGT-Oriented Approach • Multiple EGTs on different machines • Network-based system**

Table 1 : Approaches to EGT/IMT Integration

Table 1 shows some of the distinguishing features of the EGT oriented approach and compares it with our previous IMT oriented approach. An EGT *distributed* approach, which will be completed in the future, is included to make the discussion complete.

CHAPTER FOUR
DESIGN OF SYSTEM ARCHITECTURES

The three core modules to be considered in designing the architecture are: an ergonomics tool, an immersive simulation tool, and a device manager system. Each tool is an independent module, and has a loosely coupled connection with the other modules through well-designed data communication. This section describes some architectures to design an overall system where EGT is integrated with IMT.

4.1. IMT-Oriented Approach

Figure 5 shows high-level architecture for the IMT-oriented approach which was developed previously. A description of this work is included here to allow the reader to compare it with the newer EGT-oriented approach which is the focus of this work. In the IMT-oriented approach the user interacts with the IMT and is assisted by ergonomic algorithms from the EGT. There are two architectures in this IMT-oriented approach. The upper figure (I) shows the situation where the ergonomics algorithms are bundled and embedded in the IMT [30]. The lower figure (II) illustrates the scenario where the IMT shares information with EGT and the latter performs the evaluations [28]. These methods have been described in detail by Shaikh et al [23] where the systems were running under a single Unix computer.

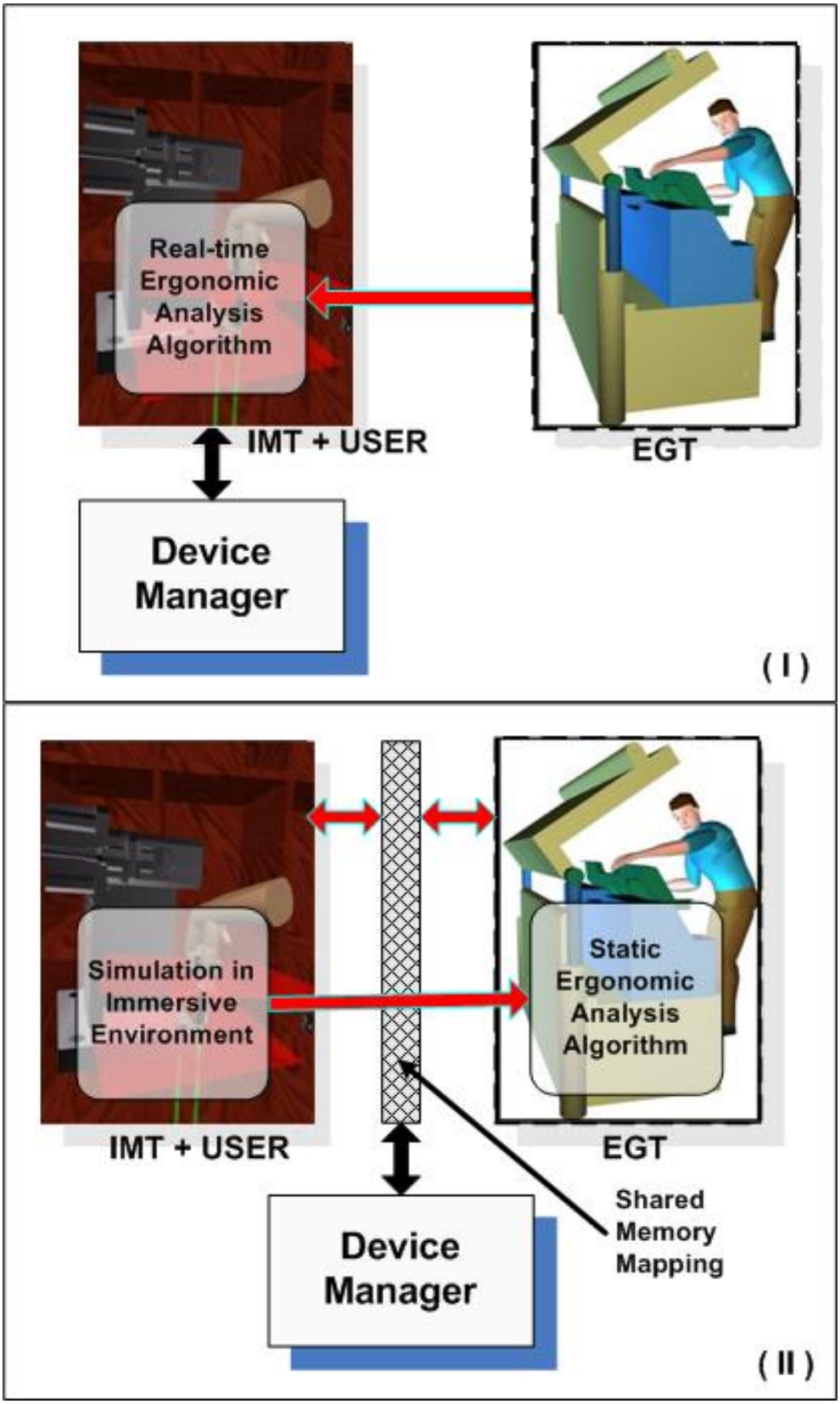

Figure 5 : IMT-Oriented Approach

4.2. EGT-Oriented Approach

In the EGT-oriented approach, which is the focus of this thesis, the ergonomics system is pivotal in the overall architecture [Figure 6]. There are two architectures we propose in this approach. In the first (Figure 6 - III), all the required functions and methods for simulation are rewritten as an independent module which is plugged into EGT. Just as Figure 5-I shows that necessary ergonomics algorithms are bundled into IMT, Figure 6-III shows how simulation algorithms are embedded into EGT. Another possible architecture (Figure 6 – IV) is that the systems are connected to each other through the network. Required data including data from VR devices, model transformations, and environment states are communicated to both IMT and EGT. EGT is for visualizing the simulation and performing ergonomics analysis. IMT is responsible for computing assembly simulation related algorithms such related to constraints, physics-based modeling, and collision detection and sending a notification message to EGT as needed.

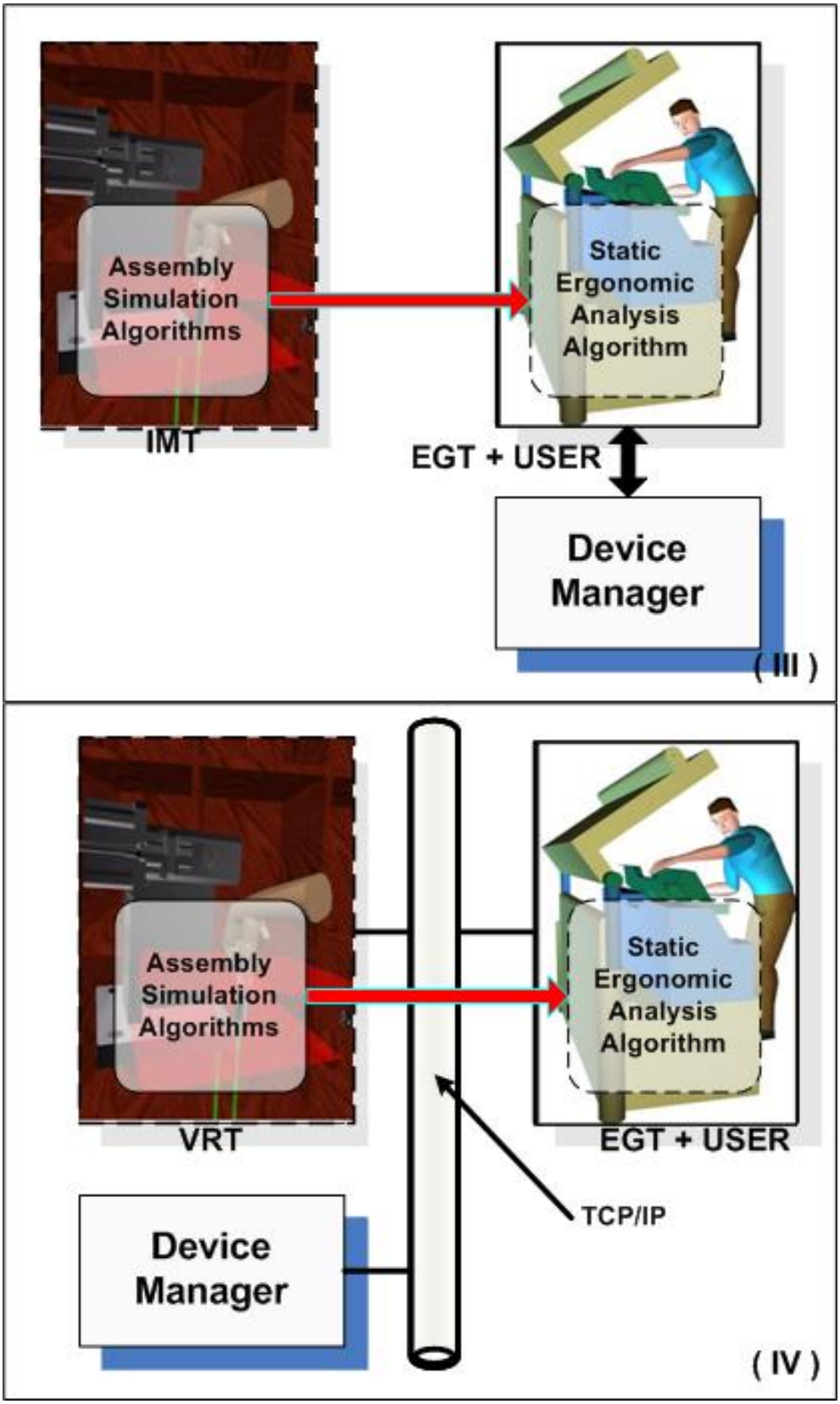

Figure 6 : EGT-Oriented Approach

4.3. EGT-Distributed Approach

The distributed assembly environment, which is an extension, is shown in the Figure 7. The roles of the device manager and IMT are similar to those in Figure 6, EGT-oriented approach, but multiple EGTs can be executed over the network. This architecture is similar to D-VADE (Distributed Virtual Assembly Design Environment) system based on CORBA (Common Object Request Broker Architecture) and Java except for the data communication methods [31, 32].

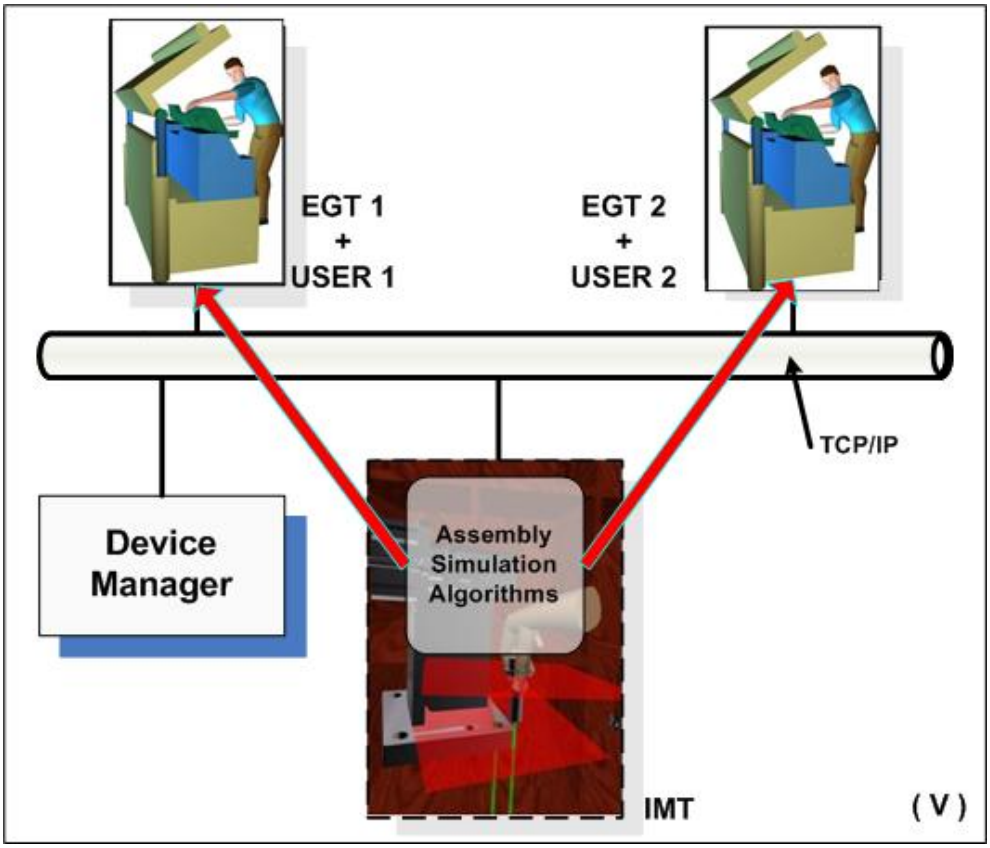

Figure 7 : EGT-Distributed Approach

As seen in the Figure 7, this will involve running two EGTs at different machines and two persons are involved in the same immersive assembly environment. For instance, suppose that a windshield would be attached to a body of an automobile. Since a windshield is too heavy for only a person to lift up and move, he/she have to collaborate with another person. If a person is in Pullman, WA and the other person is In Los

Angeles, CA, the application should be able to supply an interface such that one person can communicate with the other and two persons can work together. Establishing the collaborative environment is based on the network communication and needs some considerations to be taken for sharing resources and environments. Although this was not implemented in this thesis, we tried to design the architecture to be extensible so that the distributed version could be created easily later.

4.4. Hardware and Operating System Considerations

IMT applications demand high-end hardware to visualize 3-D models and to compute graphical algorithms. Most VR applications have been developed using UNIX systems in the past. Today, with complex hardware priced affordably, IMT is available on the personal computer with a user-friendly Windows system. If a well-developed VR application supplies algorithms or functionalities for VR simulations in the form of libraries, other applications can make good use of these libraries. In addition, a high speed network environment allows the local system to connect to a remote system and exchange data more quickly and reliably. At present, clustering technology allows a set of computers to connect to each other through fast network devices and be viewed from outside as if they were a single computer. This ability enhances the computing performance compared to a high-end single computer of the past and at a fraction of the cost.

There are a few options related to the operating system used for running applications. In our research, we will use two different operating system; Windows and

Linux. VADE, one of the IMT applications, can be only run on a Linux machine. In this study, JACK™ was run on the Windows operating system.

4.5. Data Distribution

An important consideration in integrating and synchronizing two different systems is that the environment in each system should be equivalent. For an assembly simulation scenario, each application has a surrounding and components that are positioned at some locations. For example, a virtual environment that provides a visual representation for mechanical assembly operations is composed of components, such as virtual hands, digital humans, and parts. The virtual hands are managed through an instrumented glove device such as a CyberGlove™, and the digital humans are positioned and moved with the help of tracking devices. Another consideration is which application is responsible for collecting data from VR hardware devices. EGT can connect to VR devices and get the data, and so can the IMT. Alternatively an application for managing VR devices can be created independently from EGT or IMT. Data from virtual devices can then be communicated to the applications.

4.6. Status and Events

As mentioned before, the digital hand and the human can be controlled by VR devices. However, some parts do not need VR devices to be repositioned and should be synchronized between applications. For example, the transformation data and status of a part can be shared by sending or receiving the message with status or event information. Status messages include the status of a part such as GRIPPED, RELEASED,

ASSEMBLED and etc as well as a transformation data representing the position and orientation of the part. Event messages inform the other system of the alteration of status in the part.

4.7. Coordinate Systems

Attention to the coordinate systems is an important consideration to make a successful and coordinated connection between various different systems. The different representations of the coordinate system keep applications from exporting or importing the transformation data of components directly. Therefore, if two tools use different coordinate systems, data should be transformed in the proper ways. The diagram below explains graphically the coordinate systems of the EGT and IMT and how to convert the coordinate systems. Transformation matrices can be used to convert data from one coordinate system to another [34].

Each joint in the human model has its own local coordinate system and one joint is connected to the other joint in the child-parent relationship hierarchically. For example, if a right shoulder in the human model is a child of a torso and is represented with respect to it, we can get a global coordinate system of a right shoulder through matrix multiplications. Figure 8 and Figure 9 display the relationship and transition between different coordinate systems.

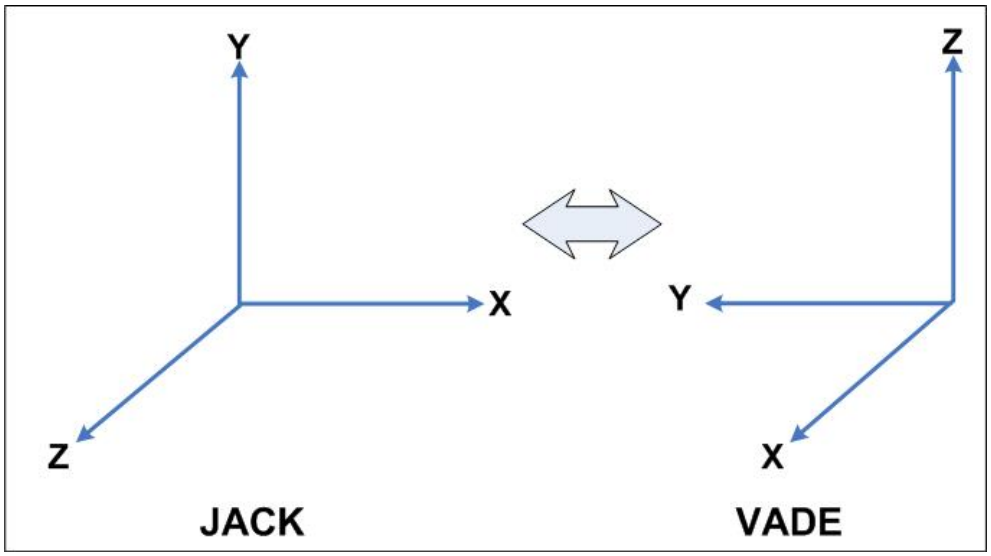

Figure 8 : Coordinate Systems of the Two Different Tools

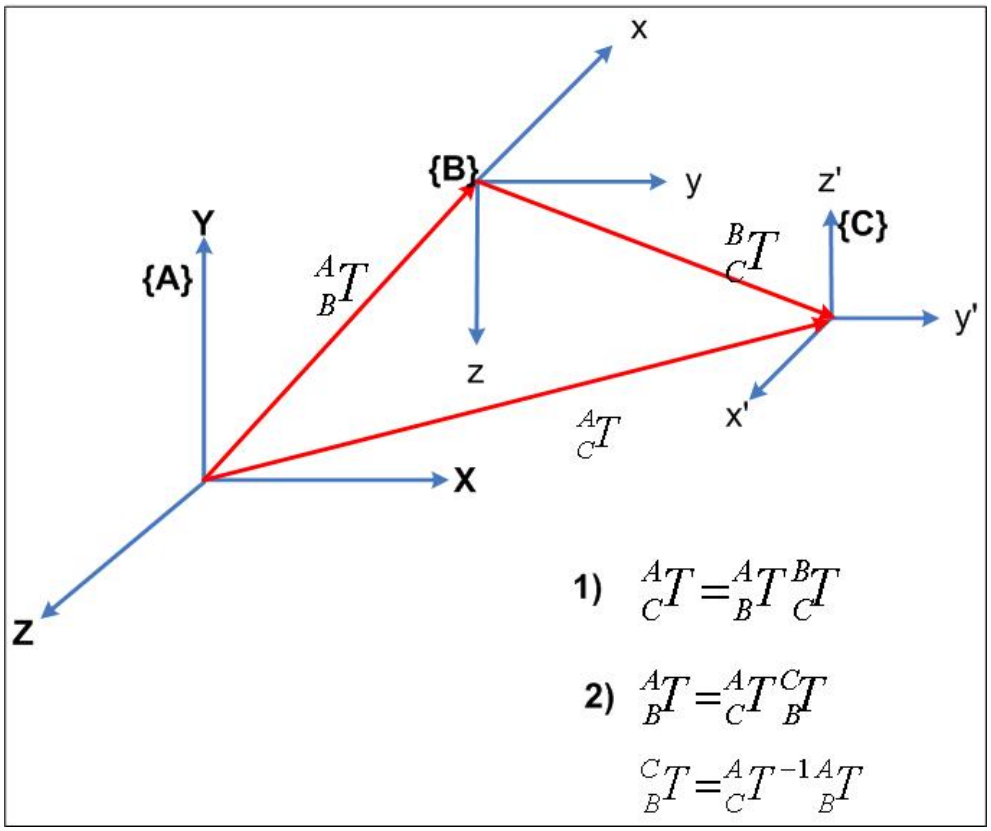

Figure 9 : Relative Coordinate System Computation [34]

CHAPTER FIVE

OBJECT-ORIENTED DESIGN

5.1. Overall System

The overall system is composed of three subsystems/modules; EGT, IMT, and Device Manager. In EGT-Oriented approach, which is the focus of this dissertation, EGT module is the core in which the environment is created and operations are monitored, while the IMT and Device Manager are helper modules to assist virtual assembly tasks within EGT such as connection to VR devices and contribution of simulation algorithms. When EGT is started up, it will connect to the IMT and Device Manager module through the TCP/IP or the shared memory and register them as clients. It acts like a server and requests clients to send some pieces of information or to run essential algorithms for proceeding with the assembly simulation. Based on information and the results of the computation, it will update the simulation within the environment and execute the ergonomic evaluations. Specific functionalities and roles of each subsystem are introduced in detail later.

5.2. Device Manager

5.2.1. Objectives and Responsibilities

The two primary roles of this application are making connections with VR devices and distributing the corresponding data. In addition to supporting a variety of communication interfaces such as socket (TCP/IP), serial (RS232C), and parallel (EPP), it also delivers the data coming from devices to other applications by inter-process

communication or over the internet.

Every VR device exposes different types of communication interfaces so that the manager needs to implement diverse methods to exchange data with the devices. As seen in the Table 2, our devices are configured mostly with RS232C and EPP. Through the data communication, the manager can acquire the hardware status or capabilities and data necessary for positioning models in the virtual environment.

Device	Manufacturer	Features
Flock-Of-Birds	Ascension	- Interface : RS232C - Speed : 9600 bps - Number of Birds : 6 EA
CyberGlove	Immersion	- Interface : RS232C - Speed : 38400 bps - Number of Sensors : 22 EA
Button Box	WSU	- Interface : Parallel EPP - Speed : 500KB/S to 2MB/S - Number of buttons ; 8 EA

Table 2 : Hardware configuration of VR devices

Data distribution is another important responsibility of the device manager. Functional integration between EGT and IMT is achieved by the synchronization of data and environment. The fast and reliable data delivery over the network or between local processes is a critical factor to accomplish the integration task.

5.2.2. Architecture of Device Manager

The application has two main modules as displayed in Figure 10. One is a device controller which plays an important role in communicating with VR devices and the other one is a network controller distributing data over the network. This subsystem is designed

as a multi-thread based application. From the experiments, it is evident that if one thread takes responsibilities of handling all VR devices, it takes much time to update the data of each device in every frame. Because of that reason, we create multiple threads and let one thread take care of one VR device. As soon as a thread is instantiated, it connects to one of VR devices and requests a specific data. In addition, it puts the data obtained from the device into the memory-mapped file which is shared by multiple applications. A network controller module in charge of distributing the data is not operated by a thread. Instead of that, the data delivery is controlled by a system timer. The application issues a timer event every 50 milliseconds and it invokes a particular event handler which fetches the necessary data from the shared memory and sends it to other subsystems or applications Figure 11.

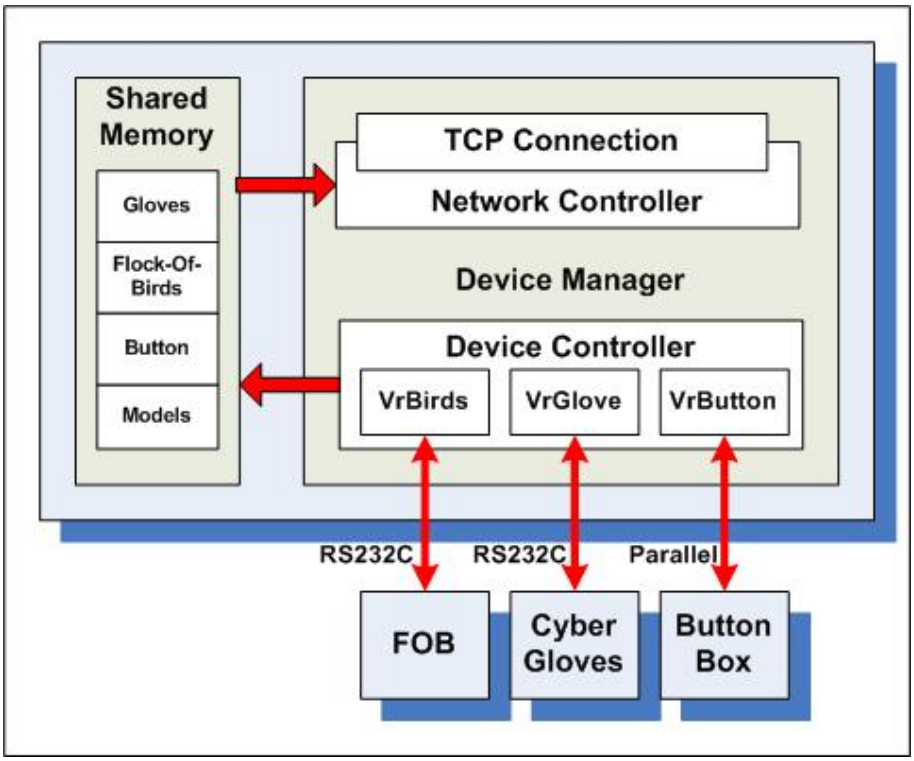

Figure 10 : Architecture of Device Manager

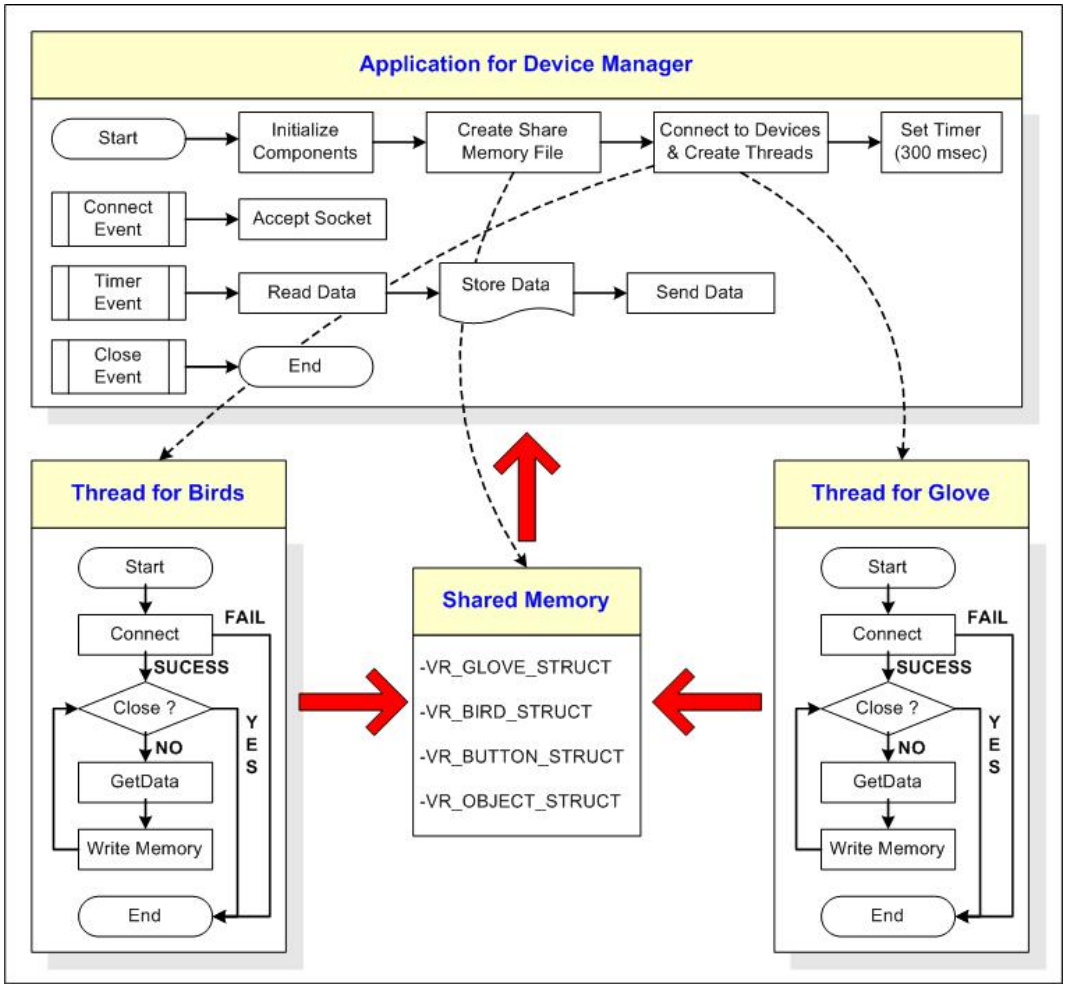

Figure 11 : Some Flow Charts of Device Manager

This module can be operated as either a client or sever. If VR devices can be connected to the device manager directly, the latter will be run as a server and be ready to convey data to other device manager program on other machines. Otherwise, the device manager will work as a client and will need to request the required data from other device managers. The C/S (Client/Server) structure is very efficient for a collaborative environment.

The other applications, such as EGT or IMT, can access the shared memory where the data sent by VR devices are updated in real-time without limitations, if they are running along with the device manager on the same machine. When it comes to the speed

in the data communication, the speed of accessing to the shared memory is much faster than that of exchanging the data through the network.

Overall, the features of the device manager can be summarized as follows.

- Connection to VR devices using the serial or parallel communication.
- Distribution of device data over the network.
- Multi-threaded application
- Client/server architecture

5.2.3. Key Classes and Their Roles

In the component diagram [Figure 12], significant modules and files are concisely displayed, even though the device manager application comprises a number of executable modules and many files. The application is classified into three hierarchies; the uppermost level is a user interface which requests a user's inputs and depicts the outputs, the middle one is user drivers and utilities providing libraries to utilize system's resources and serving as a bridge to interconnect between user-interface and I/O controller, and the lowest is communication modules to access VR devices and files.

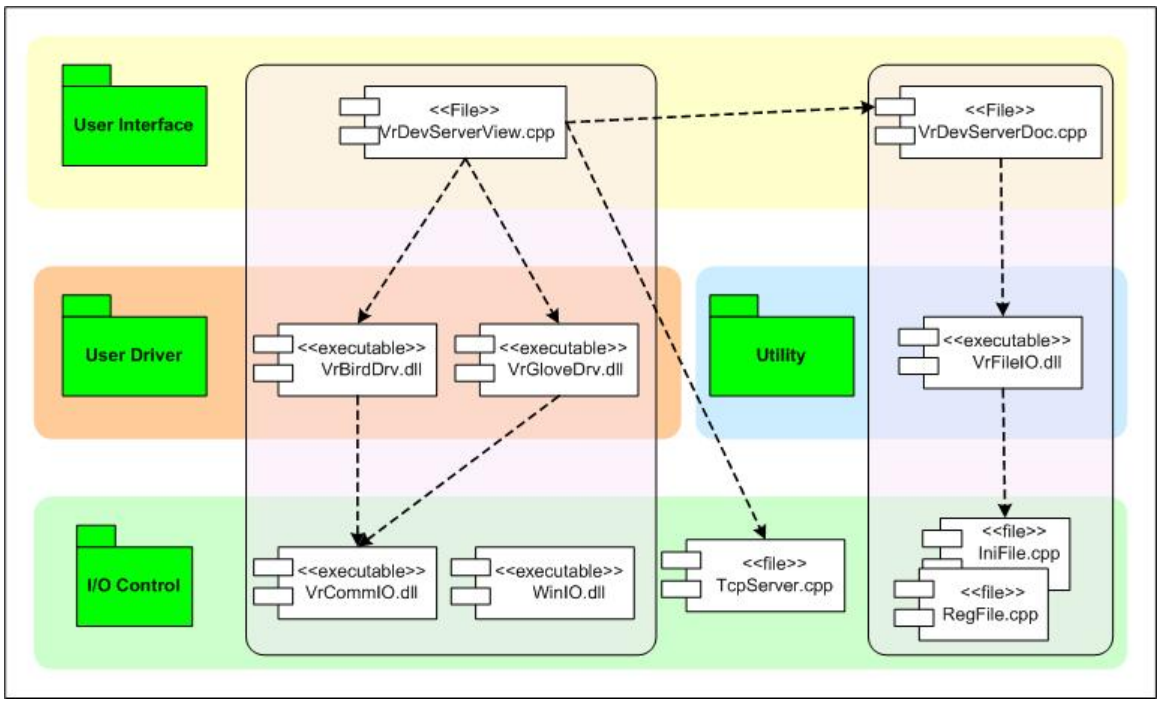

Figure 12 : Component diagram for Device Manager

- *User Interface*: The application is designed as a Single Document Interface (SDI) which means that it has a view class for displaying data on the screen and accepting users' inputs and a document class for managing all the data within the application. In the diagram above [Figure 12], CVrDevServerView class is inherited from CView class which is provided by Microsoft C++ libraries and CVrDevServerDoc is inherited from CDocument class. When a view class, CVrDevServerView, is initialized and a user wants to open the VR devices, it will create a thread which is responsible for connecting to devices, getting data from them, and saving them in the shared memory. Furthermore, it accepts TCP connections from other applications in remote machines and delivers data requested by them. The CVrDevServerDoc class will do work relevant to data management such as accessing INI files or windows registry, approaching the shared

memory, and logging traces. A view class reads data from a document class and displays it on the screen or distributes it over the internet.

- *User Drivers & Utilities*: User drivers including VrBirdDrv.dll and VrGloveDrv.dll undertake tasks in relation to the VR devices. They take charge of connecting or disconnecting to the devices, holding information about status and capabilities of the device, and continuously updating the device data such as sensor data. With the aid of user drivers, instances in the user-interface level can access the VR devices and get necessary data. Utility libraries can help users to access the INI files, registry, and trace files more easily to set or get information necessary for application operations.
- *I/O Controller*: It implements communication interface such as serial (RS232C), parallel (EPP), and TCP/IP. This module provides some APIs (Application Programming Interfaces) to allow users to communicate with real devices or other applications.

5.3. Immersive Tool, VADE

5.3.1. Objectives and Responsibilities

The VR simulation application used in this implementation is VADE which has been introduced in the Chapter Two. VADE supports two handed assembly simulation with realistic gripping, constraint-based motion, physically-based modeling and collision detection. Gravity effects are also simulated in VADE. All of these are important capabilities for assembly/disassembly simulations. These are capabilities that are not typically found in ergonomics tools which focus more on the static posture analysis and

not on the interactions between humans and the parts and between parts and the environment. In this implementation, VADE was used as a service for the EGT. The EGT sends model information to VADE in real time (gripped object transformation, other objects in the environment, etc.) and VADE computes the interactions and the physics of the environment using collisions, constraints and gravity. VADE then notifies the EGT of any changes in state and any updated transformations.

One-handed and two handed operations are supported by VADE and the virtual hands are managed by connecting to a CyberGlove™. Tracking is enabled through a Flock of Birds. The tracking information from the gloves and trackers need to be shared in real time between both the IMT and the EGT for synchronization. Parts that are grasped by the hand and moved are shared by the two applications. However, other parts are not controlled by the tracking devices but are subject to state changes and transformations.

5.3.2. Key Classes and Their Roles

The component diagram is drawn as based on the original VADE executed under UNIX system. The IMT application is composed of multiple managers such as Interaction Manager, Output Manager, Input Manager, Constraint Manager, and Model Manager. Each manager is assigned to different roles and run in the independent domain. In the diagram [Figure 13], the main module just calls one function of the interaction manager module which administers all other manager modules. The specific roles of each manager module will be as follows:

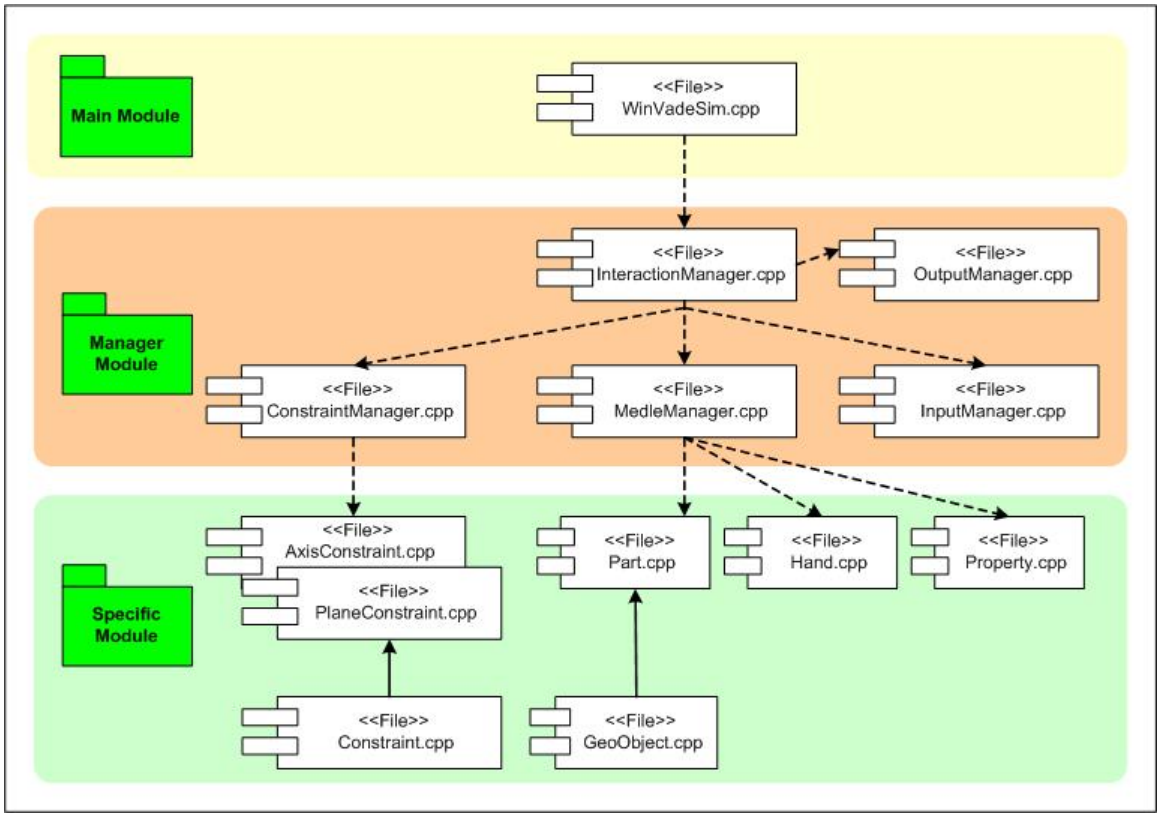

Figure 13 : Component Diagram of VADE

- *InteractionManager class*: As a core component of the application, it administers all subcomponent and calls functions or requests resources. This class is designed as event-driven methods. In simpler words, some event and states are defined in advance, and an event handler function will be called when an event is arrived. Table 3 displays some code in order to show how to design the event-driven method. Another important role of this component is to send a notification to EGT in the integrated system when the state is changed during the assembly operations. The detail contents will be explained in the next chapter.

```
// State and event table
typedef struct {
        int current_state;
```

```
        int event;
        int next_state;
        void (*action) (CInteractionManager *);
} StateEventTable;
// state values
enum {
        ST_REACHFOR,        // Reach for a part to be assembled
        ST_ASSEMBLE,        // assemble process by hand
        ST_TRANSPORT        // assemble process by crane
};
enum {
        EV_RELEASE          = 0,    // release a part from hand
        EV_GRIP             = 1,    // grip a part by hand
        EV_ASSEMBLE         = 2,    // assembling event during the
                                    // assemble process by hand
};
// configuration values of state-event table
StateEventTable g_tblStateEvent[] = {
//current_state          event              default  next_stat    event-
handler
{ST_REACHFOR, EV_GRIP,          ST_ASSEMBLE, actAssemblePart},
{ST_ASSEMBLE, EV_ASSEMBLE, ST_ASSEMBLE, actAssemblePart},
{ST_ASSEMBLE, EV_RELEASE,   ST_REACHFOR, actReleasePart},
{ -1,              -1,              -1,              0}    // end of table
};
```

Table 3 : Event-driven Method in InteractionManager Class

- *OutputManager class*: This component exists to display all the models and operations on the screen using OpenGL Performer libraries.

- *ConstraintManager class*: The algorithms of this component are really needed by EGT. This class represents operations and constraints between CAD models so that it can implement AxisConstraint class and PlaneConstraint class inherited from Constraint class.
- *ModelManager class*: This component manages all the CAD models in the virtual environment. It keeps tracks of properties of models such as material properties and physical properties. It implements a few classes; Part class, Hand class, and Property class.
- *InputManager class*: The class controls tasks relevant to communicating with the device manager application and EGT. It exists to receive data for the tracking system and gloves and send some notification messages about status alterations and events during an assembly simulation of models to EGT.

5.4. Ergonomics Tool, JACK

5.4.1. Objectives and Responsibilities

The ergonomics evaluation application used in this implementation was JACK. The principal objective of an ergonomic application is to create a virtual environment for assembly simulations, to involve a digital human in it, and to execute tasks. While a virtual human is working on a task, the application can perform the ergonomic evaluation. In order to carry out this scenario, the ergonomic application demands some sort of abilities to locate parts, move joints and segments of the digital human, and provide interaction between the human and parts or between the parts.

JACK supports VR peripherals (tracking, glove, HMD, etc.) and has industry accepted methods for determining posture and analyzing ergonomics. However, JACK does not provide sufficient capabilities for assembly evaluation. JACK provides a basic reach and gripping capability which is not realistic enough for certain scenarios and there is no checking for fingers intersecting with parts for gripping algorithms. Jack does not recognize the concept of CAD assembly hierarchy, CAD model constraints, and model properties based on a CAD model. Because of this, the ergonomic application should find ways to supplement insufficient functionalities. Integrating the ergonomic tool with selected functionality from an immersive simulation tool (IMT) can supply numerous simulation algorithms.

JACK also provides various methods for customization and integration with other applications including JackScript and a Python/C interface. Since the JackScript was not able to access to the Device Manager directly, we used the Python/C interface and shared memory.

5.4.2. Architecture of the Ergonomic Application

The diagram [Figure 14] describes the way to integrate Jack with other subsystems in detail and system architecture.

First, the data from tracking devices and gloves are transported into the device manager on the JACK-side and are saved in the shared memory-mapped file which is shared by the other application. The JACK application can access the shared memory using a dynamic linked library (DLL) written by Python/C APIs. If the devices are directly connected to the machine running the JACK application, the device manager will

work as a server and the device data are addressed into the shared memory without TCP/IP connections.

Second, simulation algorithms such as gripping a part and putting two parts together are provided by an immersive application, VADE. In order to utilize those algorithms, the ergonomic application should synchronize the environmental resources with the immersive application. The data coming from VR devices can be shared with ease since every application is connected over the network. However, some parts in virtual environment can not be handled by tracking devices so that transformation data of the parts should be shared. In this implementation, the parts information passes through the shared memory and is conveyed to an immersive application with the help of the device manager.

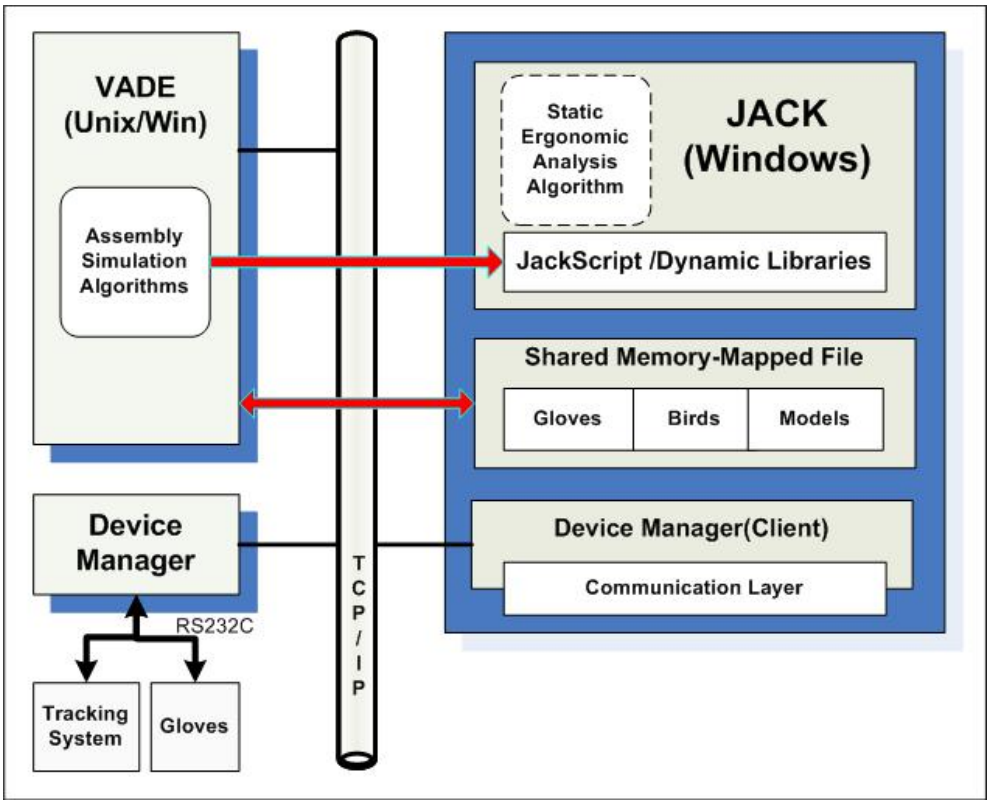

Figure 14 : System Architecture of EGT System

5.4.3. Key Classes and Their Roles

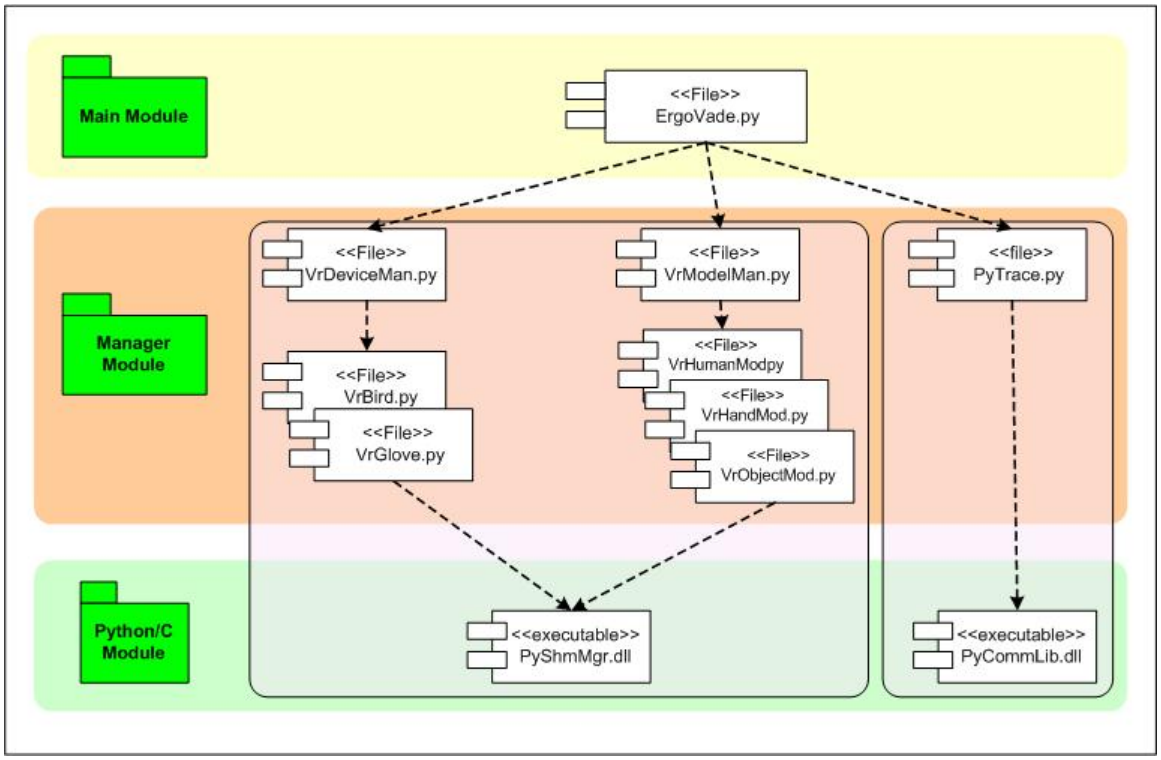

Figure 15 : Component Diagram of the Jack Application

The class hierarchy of the Jack application is similar in some ways to those for the immersive tool described before this section and there are some additional classes. It is made up of multiple manager modules which can handle VR devices and models. A component diagram is displayed above [Figure 15] and the detailed explanation of each module and files will be as follows.

- *The Structure of Shared Memory-Mapped File*: The table below shows a data structure used in the Shared Memory. It contains a lot of information about devices and parts in immersive environments. Three structures of each device are provided:
  - Status Structure : Keeps track of the status of each devices
  - Capability Structure : Holds capabilities of each devices

- Data Structure : Updates information of each devices including sensor data

This structure has all the information necessary for operating assembly simulation.

```
typedef struct _VrSharedMem {
        // Gloves
        VR_GLOVE_STRUCT         vrGloveStruct {
        VR_GLOVE_STATUS         vrRightGloveStatus;
        VR_GLOVE_CAPS           vrRightGloveCaps;
        VR_GLOVE_DATA           vrRightGloveData;
        }
        // Flock-of-birds
        VR_BIRDS_STRUCT   vrBirdsStruct {
                VR_BIRDS_STATUS   vrBirdsStatus;
                VR_BIRDS_CAPS     vrBirdsCaps;
                VR_BIRDS_DATA     vrBirdsData[16];
        }

        // Button box
        VR_BUTTON_STRUCT        vrButtonStruct {
                VR_BUTTON_STATUS        vrButtonStatus;
                VR_BUTTON_CAPS          vrButtonCaps;
                VR_BUTTON_DATA  vrButtonData;
        }

        // Objects information
        VR_OBJECT_STRUCT vrObjectStruct[12] {
                VR_OBJECT_STATUS        vrObjectStatus;
                VR_OBJECT_CAPS          vrObjectCaps;
```

```
                VR_OBJECT_DATA          vrObjectData;
        }
} VR_SHM_DEVICES, *PVR_SHM_DEVICES;
```

Table 4 : Data Structure for Shared Memory-Mapped File

- *Python/C Module*: Since the Python language is designed independent of the operating system, the application can not access system resources of Windows. Hence, this Python/C module exists to export interfaces to access the system resources. A PyShmManager module provides an ability to read data from the shared memory or write data to it. Another advantage of the Python/C module is increasing the speed of computation, because it is based on the C program language. In simpler words, it makes up for a weak point in the slow performance for mathematical calculations emerging in all script languages. A PyCommLib module in the diagram above [Figure 15] equips some libraries related to computing and debugging.
- *Manager Modules*: There are two classes: VrDeviceMan and VrModelman in the component diagram [Figure 15]. The VrDeviceMan class handles the VR devices and exports some methods to get status, capability and data. Actually, it does not access to the VR devices, but refers to the data written in the shared memory. Its subclasses, such as VrBird class and VrGlove class, have another important role which is to read necessary data from the shared memory and calibrate it. For example, the bird data should be calibrated by converting to the coordinate system for the ergonomic application. The model manager deals with attributes and properties of models including parts, hands, and humans. Operations like adjusting joints

of a human and its hand with the help of VR devices and constraining parts with other models are executed through the model manager.

- *Main Modules*: ErgoVade class is regarded as the root module as embracing all the modules and takes many responsibilities. It can not only manipulate movements and operations of components through JackScript APIs in the VR simulation environment, but also manage procedures for assembly simulation. Furthermore, it can perform ergonomic evaluations.

CHAPTER SIX

IMPLEMENTATION

This section provides some of the details of implementation of each subsystem and some integration details. The diagram below [Figure 16] depicts the overall system architecture in more detail where each application runs on independent machines and connects to each other over the network. The VR devices are connected to the Device Manager and common environmental data is delivered to both IMT and EGT in this architecture. A few variations can exist. For example, VADE application can be designed again into a module for the Windows system and be plugged into IMT. Another variation is that the device manager can be run on the same machine with IMT. Another diagram [Figure 17] shows a variation where the device manager application is replaced with the InputManger class, similar to what was explained in the previous chapter. That is, the InputManager class connects to VR devices, communicates with them, and distributes data to EGT through TCP/IP.

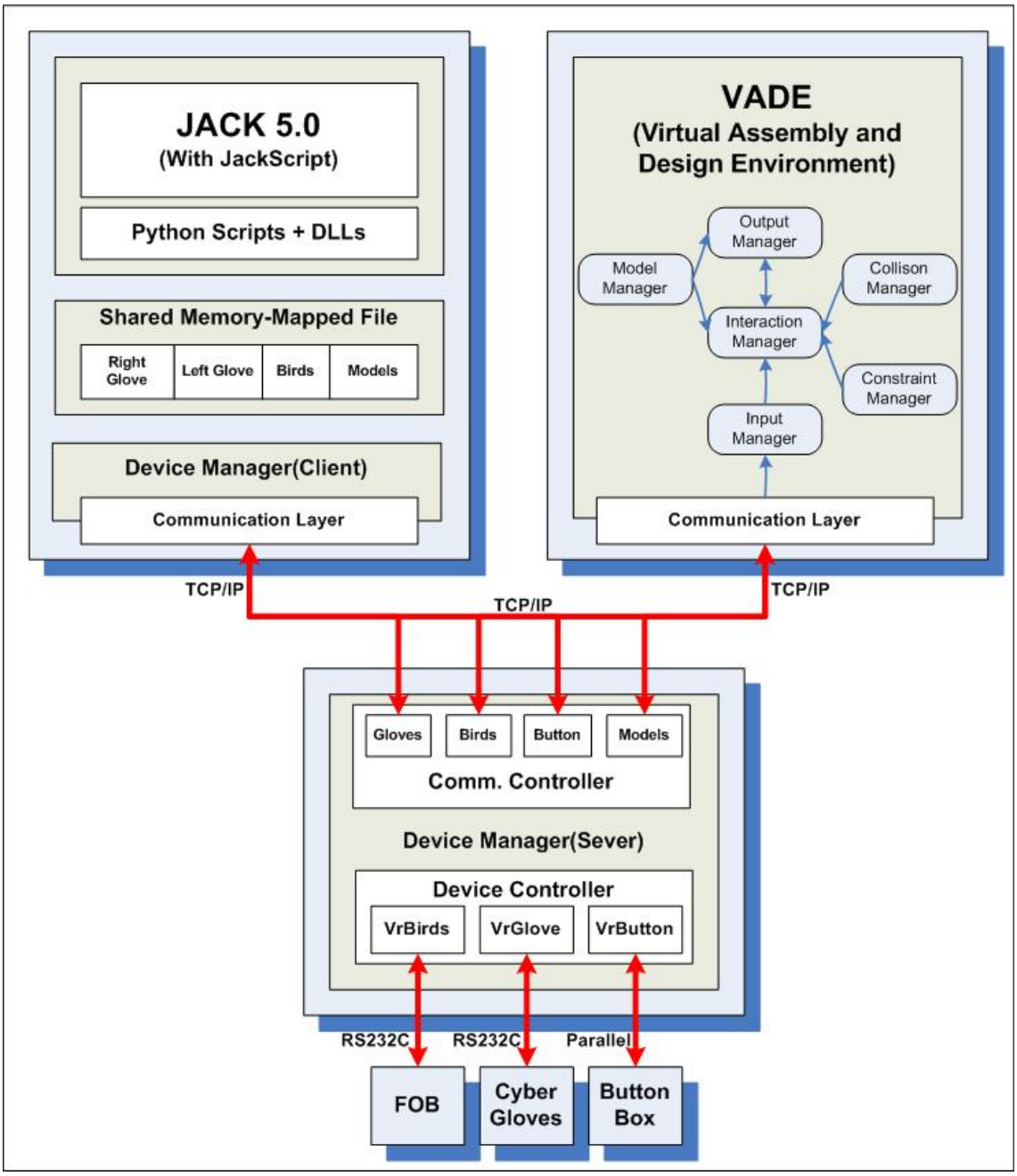

Figure 16 : System Architecture for EGT-Oriented System (I)

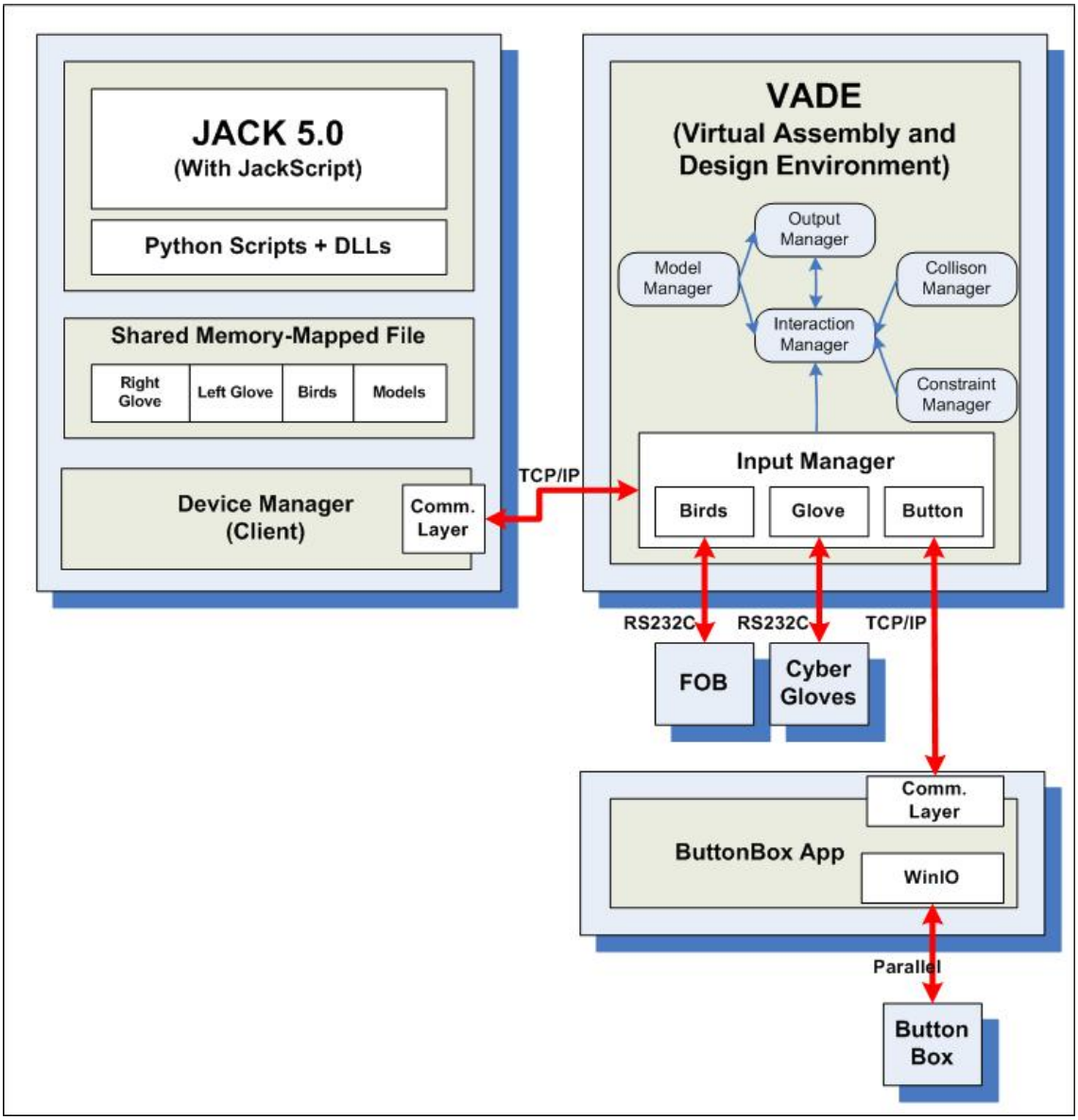

Figure 17 : System Architecture for EGT-Oriented System (II)

6.1. Device Manager

This application is developed with the Visual C++ language and MFC (Microsoft Foundation Classes) libraries provided by Microsoft and under Visual Studio .NET 2003. It is extensible to support a variety of VR devices in the future and some libraries are reusable when other applications are developed.

The device manager is designed for providing users with a friendly user interface to control VR devices and accept connections from other applications for data distribution. The objectives and functions were already discussed in the previous chapter.

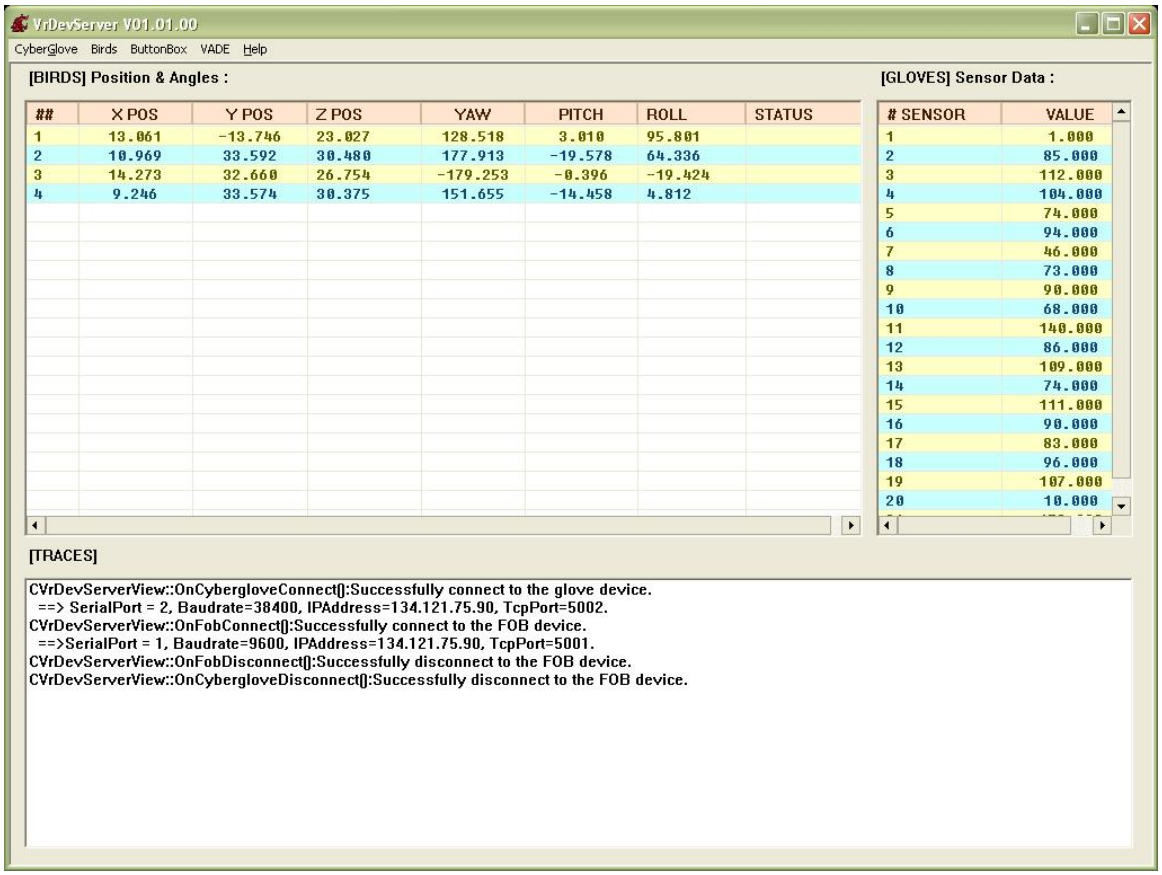

Figure 18 : Implementation of Device Manager

As seen in a snapshot [Figure 18], the main screen displays actual data obtained from the flock of birds and the glove device and logs current processes within the application. In other words, users can investigate the status of devices and flow of data at

a glance using the application. It is communicates with other applications in the background by sending the data through the TCP/IP. From the menus on the top, we can set up configurations for VR devices and communication with other applications. For example, the picture below [Figure 19] shows how to adjust configurations of a glove device such as communication settings and glove types.

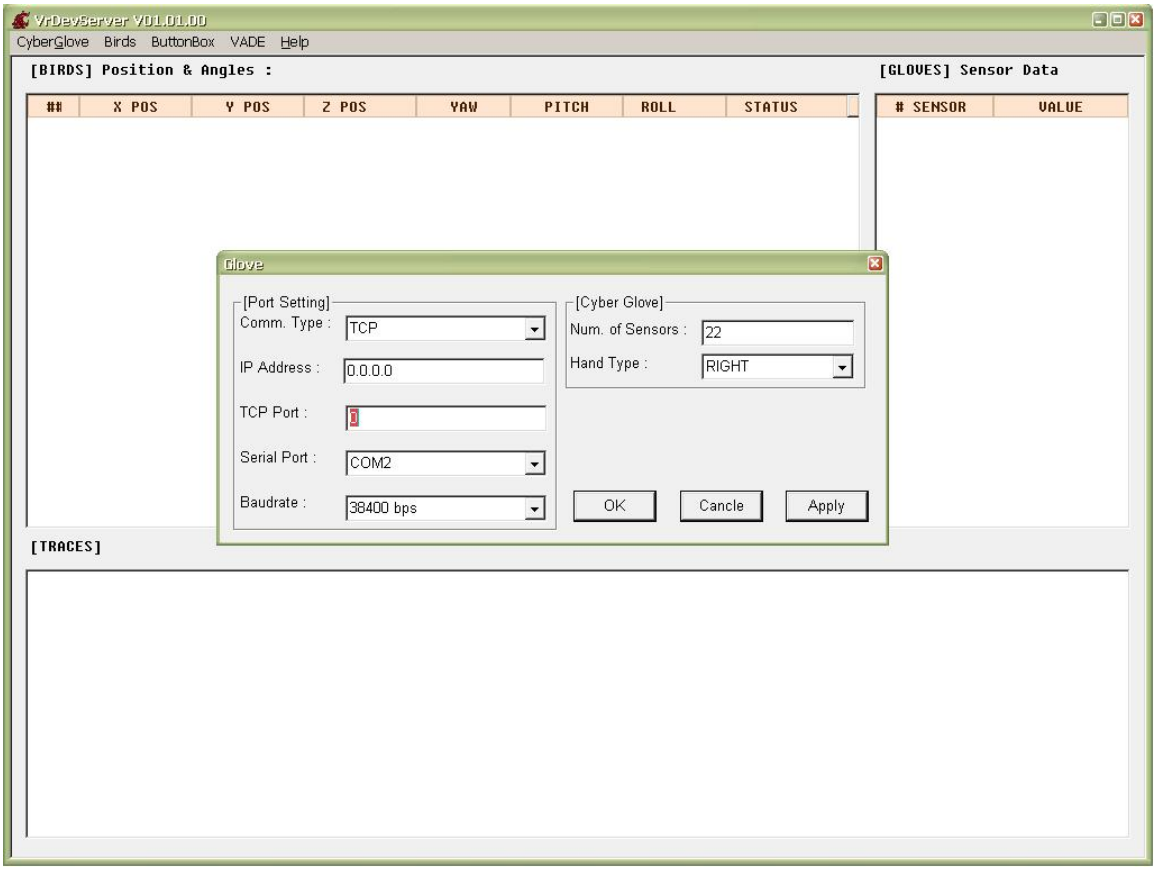

Figure 19 : How to Set Configuration of VR Device in the Device Manager

After connection to VR devices, the data would be collected by the device manager and should be calibrated before they are adopted by other applications. The Figure 20 shows how to calibrate the glove data and bird data for the EGT system.

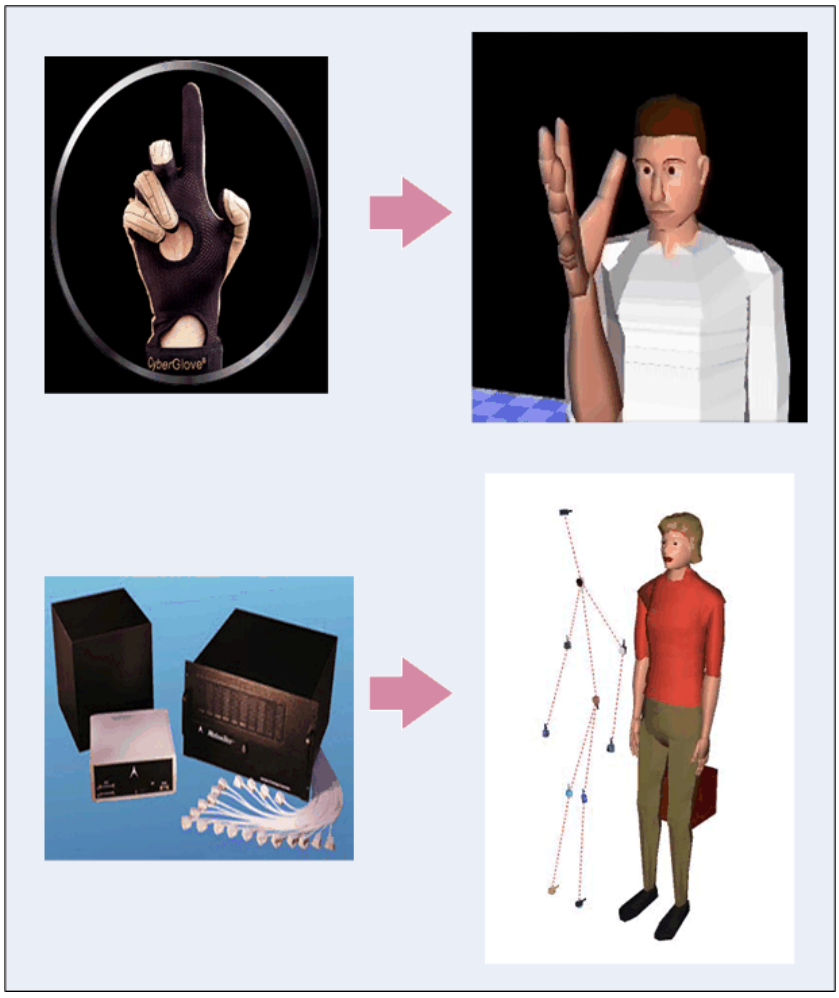

Figure 20 : Calibrations of VR Devices' data

6.2. Integration Details

This integration focused on using JACK for calculating the posture of the human model based on the tracking devices, locating the wrist and sharing that information with VADE. VADE was used to capture the glove information for finger movements, perform gripping calculations, inter-part calculations (constrained motion, kinematics, collisions, etc.), calculate physics of motion (gravity), and supply JACK with the changes in state and the updated transformations. The overall data flow between the systems is shown in Figure 21 and can be described as follows:

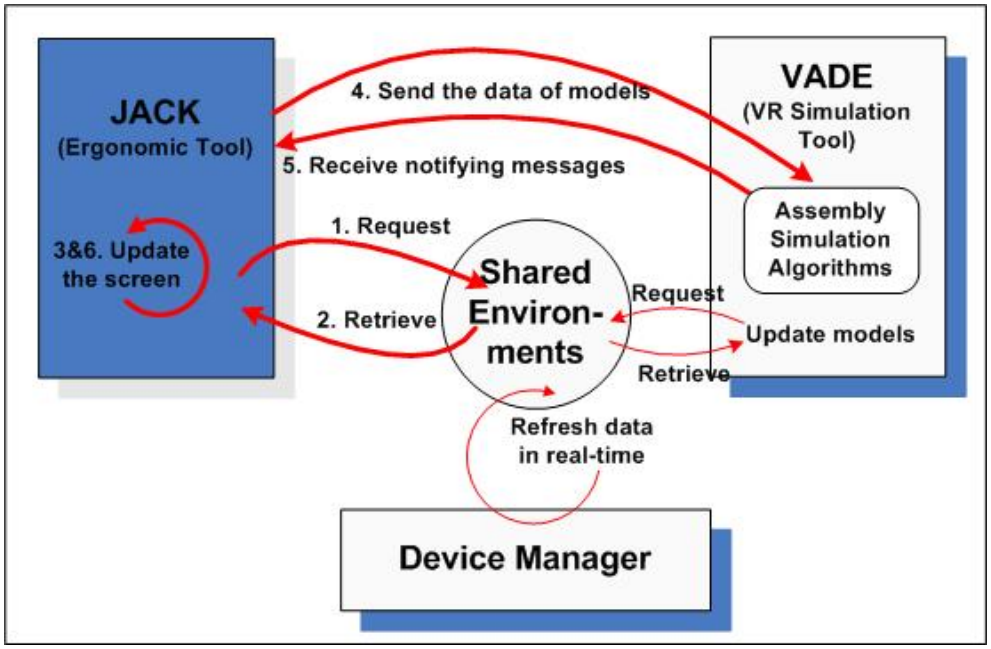

Figure 21 : Procedure to Share Environments between Systems

1. JACK draws the models such as humans and parts within the application, whose properties are already known to the device manager and the VR simulation tool.
2. JACK connects to the devices manager system and retrieves the VR devices data
3. JACK updates the location and orientation of the human models, digital hands, and parts.
4. JACK sends transformation information of each model in real-time during the assembly simulation to VADE.
5. VADE sends a notification message or event to JACK when a user grips, releases, or assembles parts. The message contains the relationships between components. For example, attach a part to a digital hand, drop a part in the space, or put two parts together.
6. Based on the messages, JACK updates the status of each component and an assembly hierarchy.

6.3. State and Event Handling Between Systems

In JACK, the user is embedded in the immersive environment. Initially, all the parts are located in bins (or some pre-defined initial locations). The base part (to which other parts will be attached) is controlled by the tracking device on the left hand. The right hand is manipulated through a CyberGlove^{TM} . When the user grabs a part from the bin, the part is attached to the right hand and the previously defined axial or plane constraints in the base part and the gripped part are activated. As the part is moved by the user, VADE checks for constraints to be applied. If all the constraints of the part are aligned and applied with those of the gripped part, two parts are assembled fully. The user can also disassemble any part that has previously been assembled. During the manipulation of the part, gravity, constraint-based motion, and physically based motion are all calculated and applied by VADE.

In this scenario, there are three states: static, in-hand, and assembled, and there are three operations: grip, release, and assemble.

- State 1 (Static): Part is released, and not constrained.
- State 2 (In hand): Part is gripped, but not fully constrained.
- State 3 (Assembled): Part is released, and fully constrained.

Table 5 and Figure 22 show the states and the transformations between the states. Pseudo-code descriptions for JACK and VADE are displayed in Table 6.

Operation / State	Grip	Release	Assemble
Q1 (Static)	Q2	---	---
Q2 (In hand)	Q2	Q1	Q3
Q3 (Assembly)	Q2	---	Q3

Table 5 : State Table for State and Event Handling

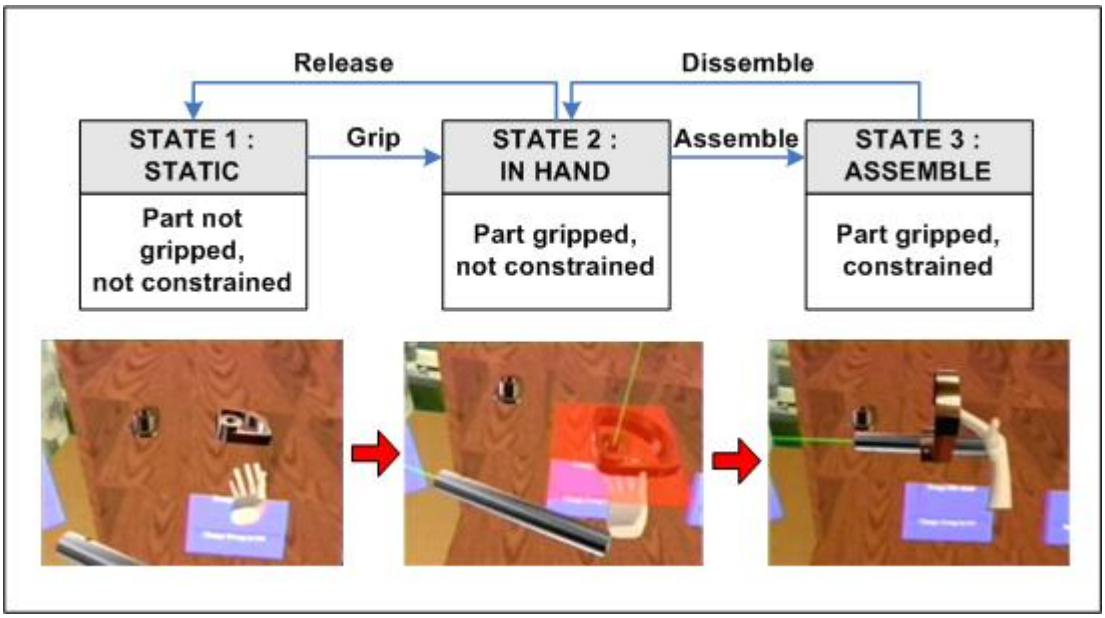

Figure 22 : State Diagram for State and Event Handling

The ergonomics application can visualize the assembly simulation and evaluate the total process, but should depend on the VR simulation tool like VADE for the alteration of states, the validation of constraints, and the detection of collisions. Sharing all pieces of information about objects enables VADE to identify the current states in the virtual assembly environment. After monitoring the user's operations and perceiving the transformation of objects, VADE is responsible for notifying JACK application of the updated state in a message or event format.

JACK_MAIN_LOOP:

WHILE :

Fetch VR device data from the device manager

Update position and angles of joints in the hand and body

Update location of parts in the virtual environments

Determine if there is new events from IMT(VADE)

IF: New event exist {

IF: A part is released {

Remove the constraints between parts

Update the location of parts

}

ELSE IF: A part is grabbed {

Attach a part to a hand

Add and display the constraints in each part

Update the location of parts

}

ELSE IF: A part is assembled {

Detach a part from a hand

Attach a part to a base part and assemble them

Update the location of parts

}

ELSE {

Unknown event & Error handling

}

Update the screen

VADE_MAIN_LOOP:

WHILE :

Fetch the data of tracking devices and a glove from the device manager

Determine the current state of assembly procedure

IF: All parts are released(STATE 1) {

Calculate intersections between parts

Determine whether a hand grabs a part

IF: A hand grabs a part {

Move to next STATE 2

Send a notification with a grabbed event

}

}

```
ELSE IF: A parts is in a hand(STATE 2) {
        Determine whether a hand releases a part
        IF: A hand releases a part {
                Move to previous STATE 1
                Send a notification with a released event to JACK
        }

        Calculate the constraint relations between parts
        Determine if constraints are fully met
        IF: Two parts are fully constrained {
                Move to newt STATE 2
                Send a notification with a assembled event to JACK
        }
}
ELES IF: two parts are assembled(STATE 3) {
        IF: A hand grabs a part {
                Move to previous STATE 2
                Send a notification with a grabbed event to JACK
        }
}

Compute the physical properties of each part
Send the part transformation matrix to EGT(JACK)
```

Table 6 : Pseudo-Code Description for State and Event Handling

CHAPTER SEVEN

TEST CASES AND RESULTS

7.1. Hardware Configuration

The original VADE was run under SGI Onyx2 Workstation with six processors, but all the source code was recompiled for Redhat Linux system and it is running under 2.8GHz Xeon dual Processors with 6GB DDR2 SDRM. The ergonomic tool, JACK, is for Windows system and it is running under a 3.0GHz single processor with 4GB DDR2 SDRAM. The hardware configuration is listed in the table below [Table 7].

System	Hardware Configuration
IMT : VADE (old) **OS : Unix**	**- Unix for VADE** **- SGI Onyx2 Workstation** **- Six Processors**
IMT : VADE (new) **OS : Redhat Linux Enterprise WS3**	- Dell Precision 470 - 2.8GHz Xeon 2 Processors - 6GB DDR2 SDRAM - nVidia Quadro FX3400 - Broadcom Gigabit Lan
EGT : JACK **OS : Windows XP**	- Dell Precision 380 - 3.0GHz 1 Processor - 4 GB DDR2 SDRAM - nVidia Quadro FX1400 - Broadcom Gigabit Lan

Table 7 : System Hardware Configuration

7.2. Scenario I: Simple IMT-Oriented Integration (Previous Work)

As VR technologies were deeply involved in virtually simulating assembly process, researchers were interested in ergonomic issues such as immersive human performances and the safety of workplaces. VADE (Virtual Assembly Design Environment) was considered as a good VR-based engineering application and in order to extend functionalities, there were a great number of activities, one of which is including the ergonomic analysis capabilities from ergonomic tools. The figure below [Figure 23] is a picture captured from an experiment where RULA was used for analyzing a digital human's posture in IMT-Oriented system. It shows that RULA fired a warning when a human picked up the piston from the top. This research was performed by WSU VRLAB [30] to integrate ergonomic analysis capabilities into an IMT. There were two approaches and these are listed below [Table 8].

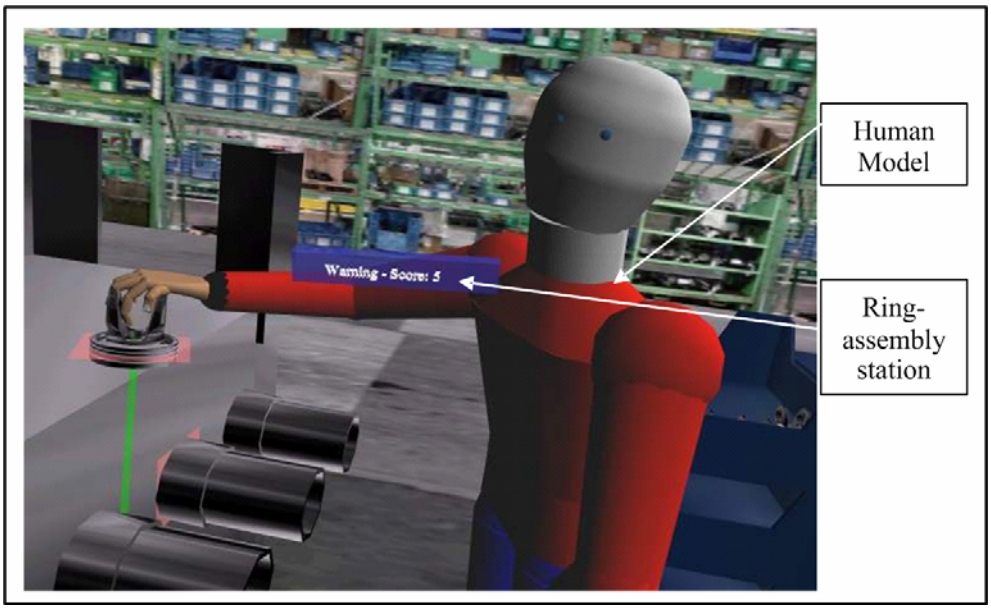

Figure 23 : IMT-Oriented Approach –RULA Warning While Picking Up the Piston Form Top of Right Assembly Station [28]

System	Strategy	Features
JACK (Unix) + VADE (Unix)	- Integration of VADE application with ergonomic software -Shared memory between applications - Independently VADE and JACK running on Unix	- Loosely coupled agents - Suitability for distributive and integrative applications - High ergonomic evaluation - Dependence on inter-process communication skills and difficulty with synchronization between two systems
	- Customizing VADE application with ergonomic analysis capability - VADE running on Unix	- Tightly coupled agents with high fidelity - Suitability for stand-alone simple application - High-end visualization and immersion application - Limitation in implementing ergonomic capabilities

Table 8 : Comparison of the Two Approaches for IMT-Oriented System [28]

7.3. Scenario II: Simple EGT-Oriented Integration

We had JACK™ running on a Windows computer and VADE on a Linux computer. Figure 24 shows an example of the integration between JACK and VADE through the TCP/IP in EGT-oriented system. VADE incessantly executes enormous functionalities, which results in consuming much time on running unnecessary tasks, for examples, loading and displaying models. The time for exchanging many pieces of information over the network should not be overlooked. In addition, the data communication over the network will result in a certain amount of latency in the delivery of update information.

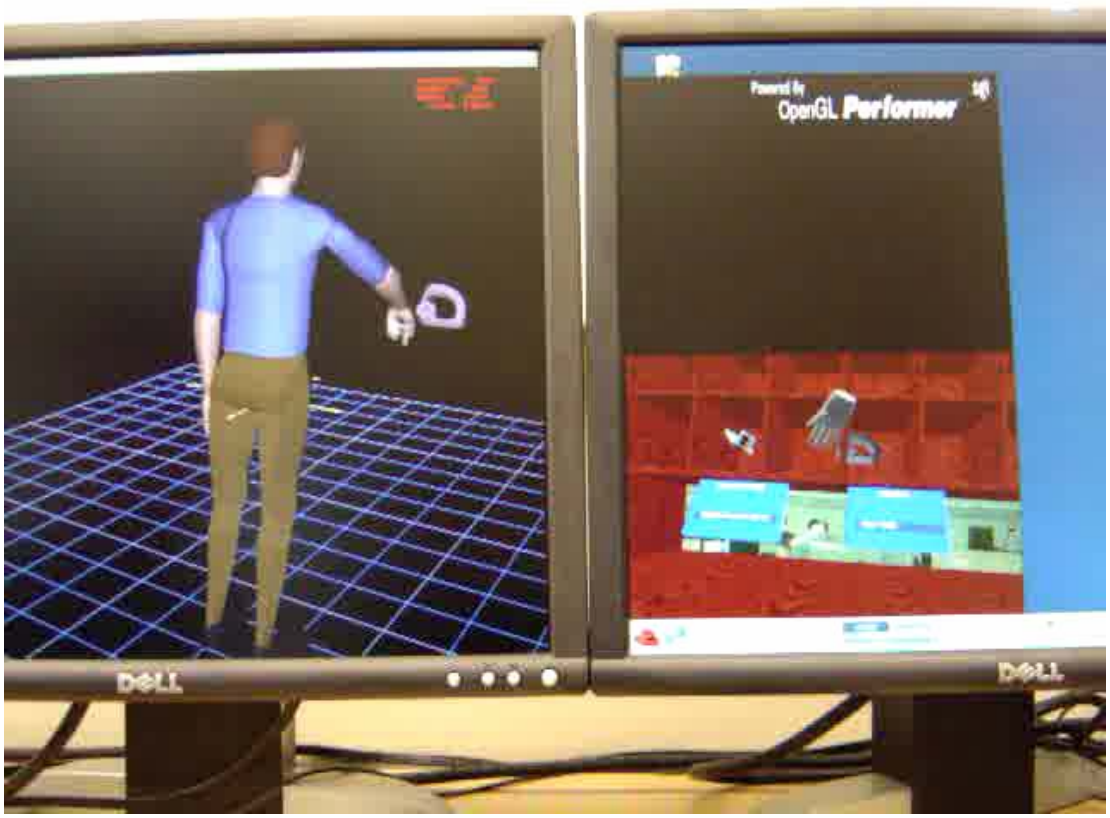

Figure 24 : Test Case I for EGT-Oriented System – Two Applications on Different Computers

The alternative case is that the principal assembly simulation algorithms are captured into an independent module. Both JACK and VADE are running on a single Windows computer and the shared-memory mapped file is adapted as the method of data communication. This technique enables multiple applications to access the memory in the same location of the system at the same time, and the communication time to be much faster than the well-known network communication. As seen in Figure 25, the upper side is the newly customized VADE and the lower one is ergonomic application. This experiment let us know that customized VADE is very light-weight, serves fully required functionalities and makes the communication much faster, but this architecture requires high-performance hardware.

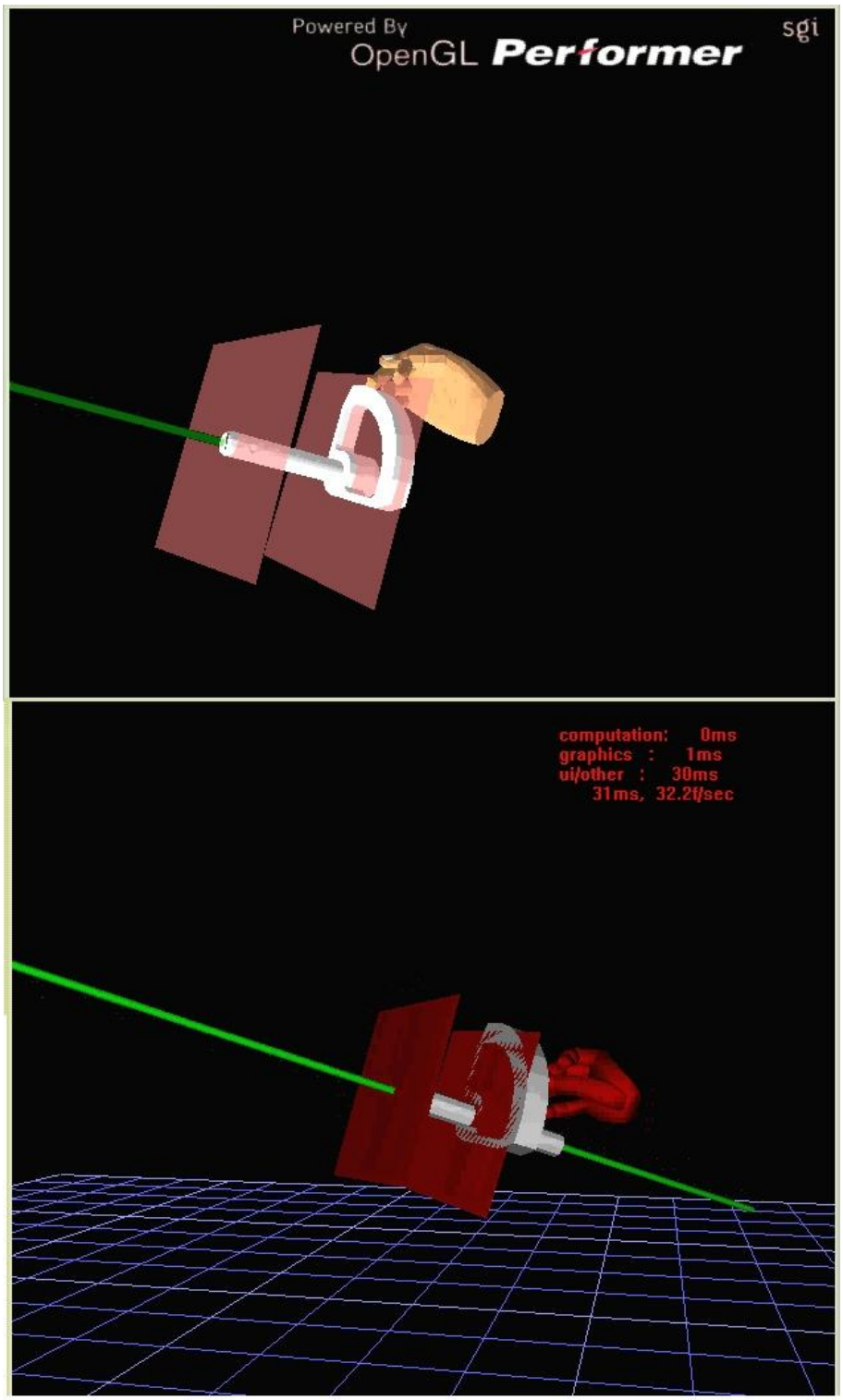

Figure 25 : Test Case II for EGT-Oriented System (1) – Two Applications On A single Computer

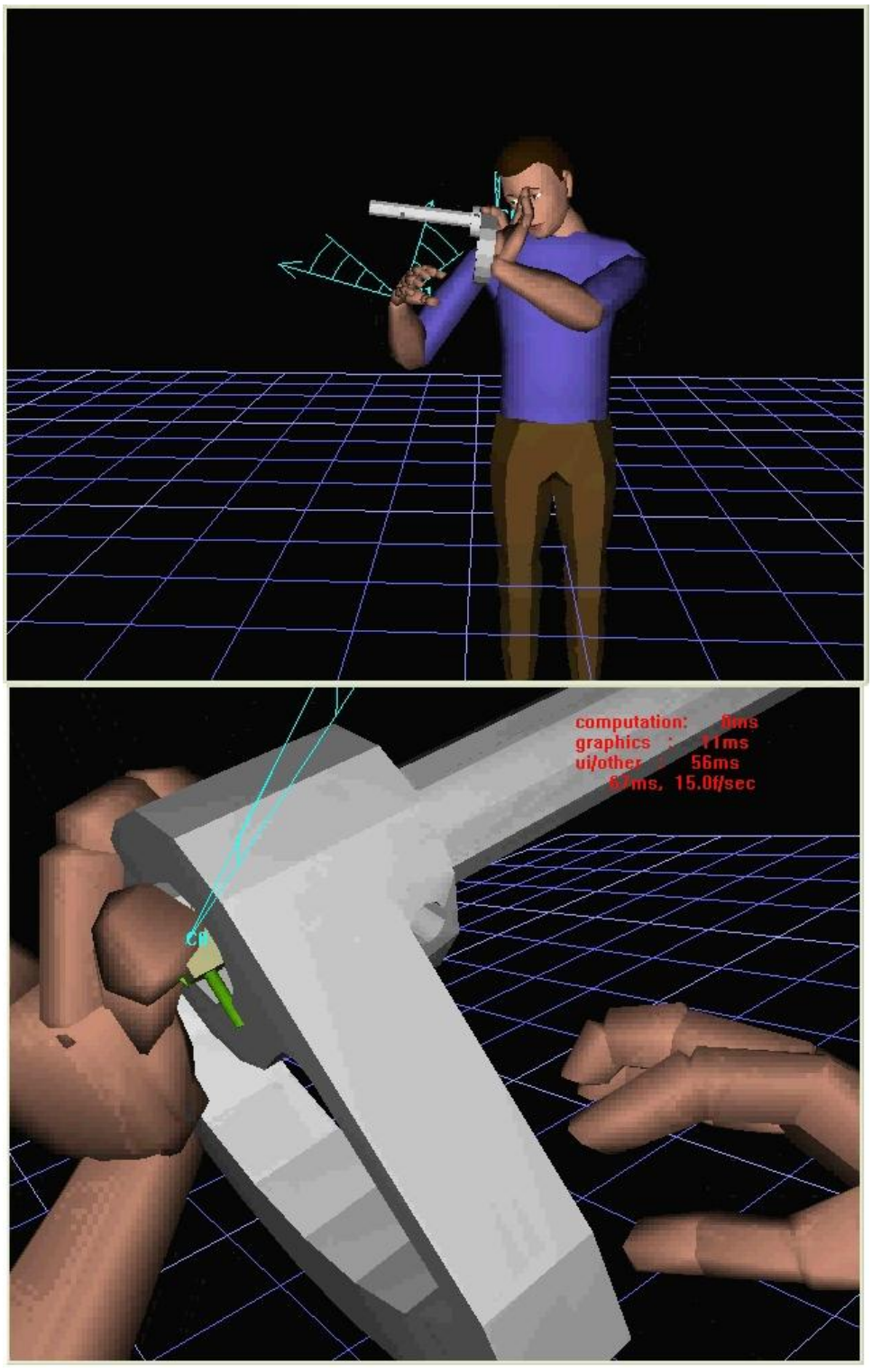

Figure 26 : Test Case II for EGT-Oriented System (2) – Two Applications On A single Computer

In the Figure 26, a human modeling working on an assembly task is involved and the figure is respectively captured from different camera position- outside and eyes.

System	Strategy	Features
JACK (Win) + VADE (Linux)	- Integration of ergonomics software with VADE application - Shared data over the network - Optionally VADE loading on the clustering system	- Loosely coupled agents - Suitability for distributive and integrative applications - High ergonomics evaluation, fully immersive simulation and high-quality visualization - Dependence on inter-computer communication skills and difficulty with synchronization between two systems
JACK (Win) + VADE (Win)	- Customized and light-weight VADE - Customizing Ergonomics application with assembly simulation capability - Shared memory between applications - Independent module with virtual assembly simulation capabilities	- Tightly coupled agents with high fidelity - Suitability for autonomous aggregate applications - High ergonomics evaluation and immersive simulation - Difficulty with importing virtual assembly simulation capabilities

Table 9 : Comparison of the two Approaches for EGT (JACK) Oriented System

All in all, two different architectures are prepared in simple EGT-Oriented systems. The conspicuous difference between them is where IMT is located. One is that an executable VADE application is an independent module and runs on Linux system. The other is that a VADE-similar module is recreated and runs on the same machine with JACK application. The two architectures are compared and some features from experiments are tabulate on the Table 9. EGT-Oriented systems are naturally designed to provide high ergonomic evaluations in an immersive environment. In case of the upper architecture in Table 9, since two applications are running remotely on different machines,

the systems are loosely coupled, but immersive operations are fully simulated and visualizations with high performance are exposed. Most of all, the communication proficiency should be focused to transfer necessary data with rapidity and without flawless. In the second architecture in Table 9, two applications are tightly coupled, because ergonomic algorithms are reorganized into an independent module and plugged into the ergonomic system. The speed for transferring data is increased but overall system performance resulting from the two big applications is a little diminished.

7.4. Statistical Analysis

From the previous section, features of each integration methodology in both IMT-oriented system and EGT-oriented system are illustrated. However, the stability of applications and speed to update screen were not introduced. The latency resulting from the speed to exchange information of components including a digital human, hands, and parts have an effect on moving or locating them in the virtual environments. Also, the frame rate is another issue to be taken into account, because it is useful to measure how quickly the application updates and refreshes all the graphical components on the screen. With the help of statistical analysis, this chapter discusses which factors can significantly affect latency and frame-rate in the graphical display.

The latency is observed by the interval time which is an elapsed time between when an immersed user tries to grab a part in the virtual environment as quickly as possible and when the application attaches the part into a hand and displays constraints. To make latency checking more feasible, the device manager sends the data of VR devices at some intervals to produce latencies between applications. The four time intervals are chosen: 1000, 2000, 3000, and 4000 milliseconds. The frame rate is

measured in the main loop of the ergonomic application by taking the time necessary to refresh all the models one time. In the experiments for the frame rate, there are three treatment factors in the tables: operating systems (Windows, Linux), communication type with VR devices (RS232C, TCP), and the number of models. The observed results are shown in the Table 10 and Table 11, respectively. After observations, the data will be analyzed using the SAS (Statistical Analysis System) software. The output from the analysis is also seen in Table 12 and Table 13, respectively.

In term of latency in the ANOVA (ANalysis Of Variance) table [Table 12], there are significant differences in time due to the time intervals. That is, the time interval gives significant effects on the latency. The response from experiments is presented in the Figure 27 which shows that the time is greater as the time intervals are larger. The experiments show that a time-lag to exchange data between applications results in producing latency to display the updated information at some intervals. This problem will make a user to feel uncomfortable to proceed his/her task in the virtual environment.

Observation	1000 msec	2000 msec	3000 msec	50000 msec
1	1.17	1.62	2.65	2.38
2	1.03	1.84	2.38	3.06
3	0.94	2.65	2.70	3.73
4	0.81	1.26	2.70	3.19
5	0.81	2.02	2.92	5.04
6	0.76	1.57	2.79	2.92
7	0.81	1.35	3.06	3.46
8	1.12	1.48	2.92	4.90
9	0.90	1.84	3.01	4.27
10	0.00	1.84	2.92	4.77

Table 10 : Time for Grabbing a Part (time in seconds)

Number of Components	Windows (JACK) + Windows(VADE)		Windows(JACK) + Linux(VADE)	
	RS232C	TCP/IP	RS232C	TCP/IP
2	31	32	30	31
	30	32	30	32
	31	31	15	32
	30	16	31	30
	31	32	31	31
5	31	32	31	30
	31	32	30	31
	30	31	30	31
	32	30	30	30
	30	30	31	30
10	30	47	31	31
	47	31	30	46
	47	46	31	31
	47	31	30	30
	46	46	31	31
20	46	46	46	46
	46	46	46	46
	46	46	46	46
	47	47	46	31
	62	46	47	46

Table 11 : Frame-rate in JACK to Refresh Models (time in milliseconds)

*p-value** : indicates the corresponding test is statistically high significant*

Source	Degree of freedom	Sum of Square	Mean square	F-Value	Pr>F
Time interval	3	45.927	15.309	56.72	<0.0001**
Error	36	9.716	0.270		
Total	39	55.643			

Table 12 : ANOVA Table for Grabbing Time

*p-value** : indicates the corresponding test is statistically high significant*

Source	Degree of freedom	Sum of Square	Mean square	F-Value	Pr>F
OS	1	608.40	608.40	19.13	<.0001**
Comm.	1	372.10	372.10	11.70	0.0008**
OS*Comm.	1	225.63	226.63	7.09	0.0086**
Models	3	9328.43	3109.48	97.78	<.0001**
OS*Models	3	635.95	211.98	6.67	0.0003**
Comm.*Models	3	242.85	80.95	2.55	0.0584
OS*Comm.*Models	3	103.225	34.41	1.08	0.3588

Table 13 : ANOVA Table for Frame-rate in JACK

Another statistical analysis shows significant difference in frame-rate to refresh the screen due to the following effects: operating system, communication type, the number of models, interaction between operating system and communication type, and interaction between operating system and the number of models. There are no significant differences due to interaction between the number of models and communication type and interaction among all effects [Table 13]. From the statistical analysis, it can be concluded that most of the considered factors have an effect on the frame-rate in the ergonomic tool. In the Figure 28, it is evident that more time is taken as the number of models is increased. The detail SAS code will be displayed in Appendix E.

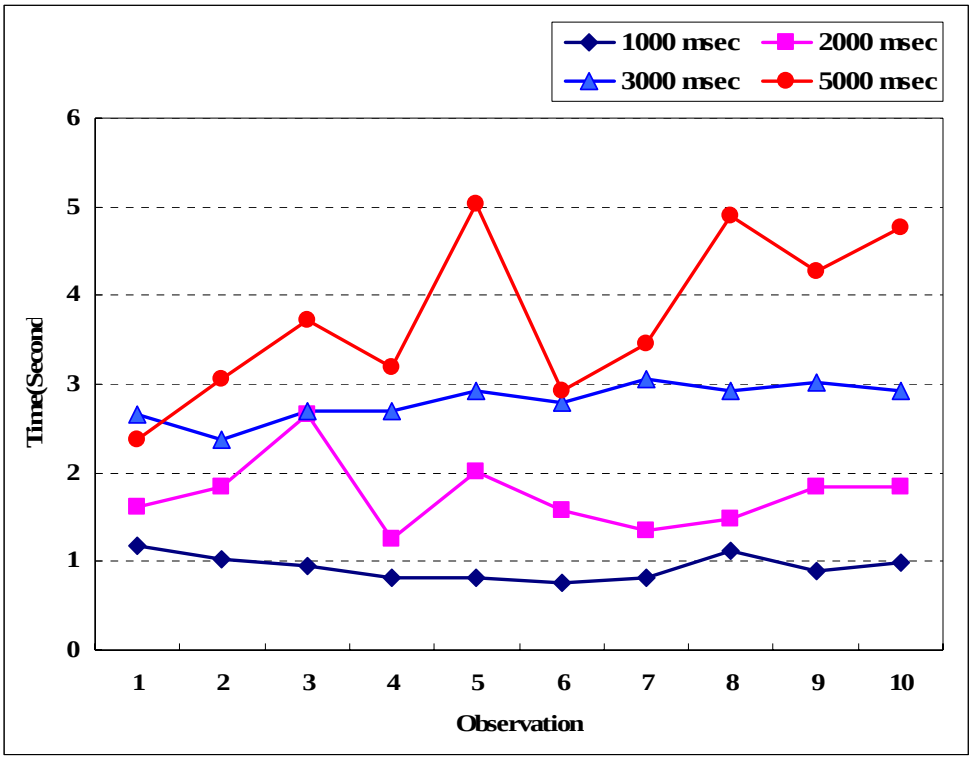

Figure 27 : The Response Plot for Grabbing Time

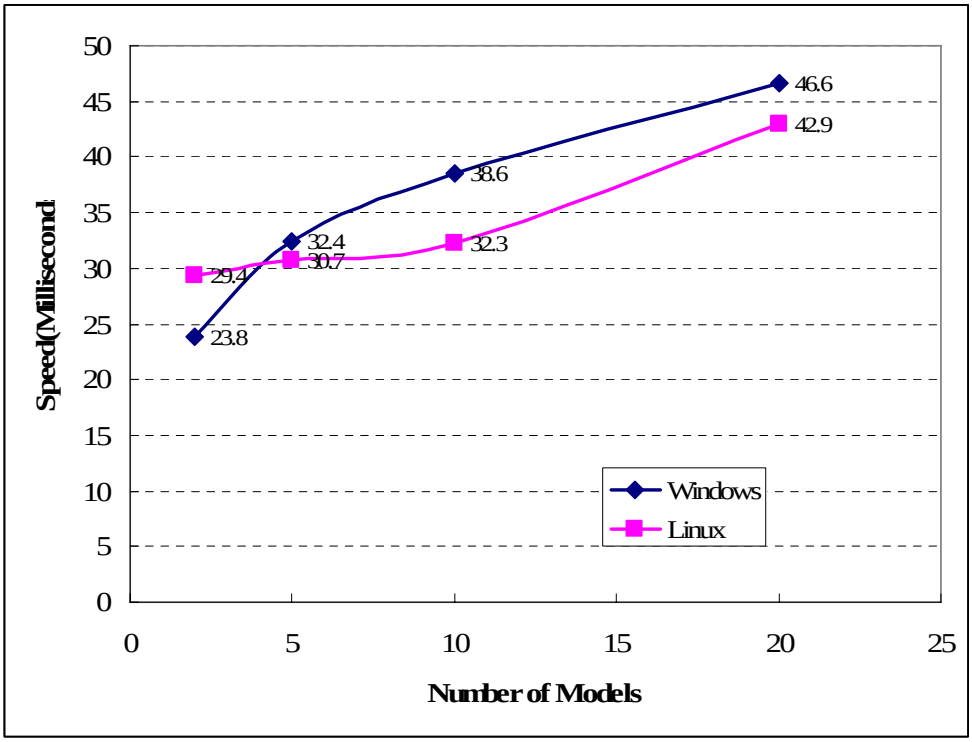

Figure 28 : The Mean Response Plot for Frame-rate in JACK

CHAPTER EIGHT

SUMMARY AND FUTURE WORK

8.1. Summary

This work presents an approach to use a traditional ergonomics evaluation tool as a front-end and provide access to certain key functionalities present in immersive evaluation tools used for assembly simulations. This provides important functionality that is not present in either system by itself. The details of this approach are presented and contrasted with an earlier approach where the front end was the immersive evaluation tool and not the ergonomics evaluation tool. Implementations with a specific EGT and IMT are also presented and discussed.

A few methodologies are introduced for this EGT-oriented approach and all of them have advantages and disadvantages. In particular, one of the architectures is shared-memory based and the other is based on a network protocol. Our studies found that the shared-memory based architecture which has an embedded simulation module and the inter-process communication technology without a connection to the network, is faster than the other architecture for data communication.

This work shows that as an EGT was integrated with an existing IMT, some valuable synergistic effects were acquired, such as ergonomic evaluations for assembly scenarios, fully immersive simulation, and high quality visualization. In the previous IMT-oriented architecture, the ergonomic evaluation was partially supported or the data of human postures in the IMT was conveyed to the EGT and ergonomic evaluation was executed. However, the EGT-oriented system allows users to run ergonomic analysis

algorithms as well as process assembly simulations. In other words, capabilities and algorithms from the IMT were fully embedded into the EGT-based system.

From experiments and statistical analyses, some conclusions were drawn that the amount of latency in the delivery of updated information and the speed of frame-rate depends on the operating system (Windows or Linux), communication type (RS232C or TCP), and the number of models in the virtual environment. In our study, the number of models was limited to 20 in the experiment, but a lot of models in the virtual environments might increase the latency and frame-rate more strongly.

There are still some unsettled restrictions. First of all, the collision-detection functionality is required to depict the virtual environment more realistically, but unfortunately those libraries are not provided for windows system. Second, inverse-kinematics in JACK keeps a human model from moving more flexibly, so that we are able to make more natural human model by removing or adjusting the inverse-kinematics. The last limitation lies in the performance of data exchange, especially the CAD models. The case study in this work assumes that each application has the same CAD models in advance. Even though it will be very burdensome to deliver the CAD model data over the network or among applications, an efficient method to exchange data in the models should be considered.

8.2. Future Work

In the future, a distributed EGT system can be inherited from the EGT-oriented architecture and additional functionality to run multiple EGTs over the internet can be added. The procedures to share the environments and handle events and status messages

between systems can still be used identically. Since each EGT in this system connects to different VR devices and perform the different operations, we should carefully design the methods to share the VR devices data as well as environmental data. Since the current EGT-oriented architecture has been designed to be extensible, it can form the basis of the distributed EGT system.

In addition, we are using concepts related to distributed ontology-based knowledge system [35] in an integration framework. This new research will be very helpful to exchange domain knowledge or information related to the multiple applications and the related integration issues.

REFERENCES

1. U.S. Department of Labor Workplace Injuries and Illnesses in 2001, http://www.bls.gov/iif/home.htm
2. **Chaffin, D. B., Faraway, J. J., Zhang, X., and Woolley, C.,** 2000, "Stature, Age, and Gender Effects on Reach Motion Postures", Human Factors, Vol.42, No.3, pp.408-420.
3. **Mavrikios, D., Karabatsou, V., Alexopoulos, K., Pappas, M., Gogos, P., and Chrysolouirs, G.,** 2006, "An approach to human motion analysis and modeling", International Journal of Industrial Ergonomics, Vol.36, pp.978-989.
4. **Das, B., and Shikdar, A.,** 1999, "Participative versus assigned production standard setting in a repetitive industrial task: a strategy for improving worker productivity", International Journal of Occupational Safety and Ergonomics, Vol.5, No.3, pp.417-430.
5. UGS - Human Performance: Jack, http://www.ugs.com/products/tecnomatix/human_performance/jack/
6. HUMAN SOLUTIONS: RAMSIS, http://www.human-solutions.com/automotive_industry/ramsis_en.php
7. **Hetland, M. L.** 2005. Beginning Python, vol. Apress.
8. Python Programming Language, http://wwww.python.org
9. **Jayaram, S., Vance, J., Gadh, R., Jayaram, U., and Srinivasan, H.,** 2001, "Assessment of VR Technology and its Applications to Engineering Problems", ASME Transactions Journal Of Computing and Information Sciences in Engineering, Vol.1, pp.77-83.
10. **Jayaram, U., Jayaram, S., DeChenne, C., Kim, Y. J., Palmer, C., and Mitsui, T.,** 2004, "Case Studies Using Immersive Virtual Assembly in Industry" Proceedings of DETC '04 Computers and Information in Engineering Conference, Salt Lake City, Utah, Oct 2, 2004.
11. **Jayaram, S., Angster, S., Gowda, S., and Kreitzer, R. R.,** 1998, "An Architecture for VR-Based Virtual Prototyping of Human Operated Systems" Proceedings of ASME 1998 Design Engineering Technical Conferences, Atlanta, GA, September 1998.
12. **Whitman, L. E., Jorgensen, M., Hathiyari, K., and Malzahn, D.,** 2004, "Virtual Reality: Its Usefulness For Ergonomic Analysis" Proceedings of the 2004 Winter Simulation Conference.
13. **Deisinger, J., Breining, R., and Robler, A.** ERGONAUT: A tool for ergonomic analyses in virtual environments.
14. **Lee, N. S., Park, J. h., and Park, K. S.,** 1996, "Reality and human performance in a virtual world", International Journal of Industrial Ergonomics, Vol.18, pp.187-191.
15. **Fernando, T., Murray, N., Tan, K., and Wimalaratne, P.,** 1999, "Software Architecture for a Constraint-based Virtual Environment" Proceedings of ACM Symposium on Virtual Reality Software and Technology.
16. **Jones, R. E., and Wilson, R. H.,** 1996, "A Survey of Constraints in Automated Assembly Planning" The 1996 IEEE International Conference on Robotics and

Automation, Location.
17. **Barzel, R.** Physically-Based Modeling for Computer Graphics, vol. 1. Academic Press, INC.
18. **Jayaram, S., Jayaram, U., Wang, Y., Lyons, K., and Hart, P. F.,** 1996, "VADE: A Virtual Assembly Design Environment", IEEE Computer Graphics and Applications, Vol.19, No.6, pp.44-50.
19. **Wang, Y., Jayaram, U., Jayaram, S., and Shaikh, I.,** 2003, "Methods and Algorithms For Constraint Based Virtual Assembly", Virtual Reality, Vol.6, pp.229-243.
20. **Jayaram, S., Connacher, H., and Lyons, K.,** 1995, "Virtual Assembly Design Environment" Proceedings of ASME 1995 Design Technical Conferences/International Computers in Engineering Conference, Boston, September.
21. **Jayaram, S., Angster, S., and Hutton, D.,** 1997, "Case Studies on the Use of Virtual Reality for an Integrated Design and Manufacturing System" Proceedings of ASME Design Engineering Technical Conference, Sacramento, CA, September.
22. **Cramer, D., Jayaram, S., and Jayaram, U.,** 2002, "A Collaborative Architecture For Multiple Computer Aided Engineering Applications" Proceedings of 2002 ASME Computers in Engineering Conference.
23. **Jayaram, S., Kreitzer, R., and Jayaram, U.,** 1998, "Preserving Design Intent in Data Integration Between Virtual Prototyping and CAD Systems" Proceedings of ASME DETC98, Atlanta, Georgia, September 13-16, 1998.
24. **Jayaram, U., Kim, Y. J., Jayaram, S., Jandhyala, K., and Mitsui, T.,** 2004, "Reorganizing CAD Assembly models (As-Designed) for Manufacturing Simulations and Planning(as-build)", ASME Transactions Journal Of Computing and Information Sciences in Engineering, No.Special Issue on Virtual Reality Application in Product Development, pp.98-108.
25. **Jayaram, S., Connacher, H., and Lyons, K.,** 1996, "Integration of Virtual Assembly with CAD" Proceedings of Symposium on Virtual Reality in Manufacturing Research and Education, Chicago, October.
26. **Wang, Y.** 1998. Physically Based Modeling In Virtual Assembly. Washington State University.
27. **Wang, Y., Jayaram, U., and Jayaram, S.,** 2001, "Physically Based Modeling In Virtual Assembly" Proceedings of DETC 2001: DETC-CIE, Pittsburgh, Pennsylvania, Sep. 9-12.
28. **Shaikh, I., Jayaram, U., Jayaram, S., and Palmer, C.,** 2004, "Participatory Ergonomics Using VR Integrated With Analysis Tools" Proceedings of the 2004 Winter Simulation Conference.
29. **Shaikh, I., Kim, Y., Jayaram, S., Jayaram, U., and Choi, H.,** 2003, "Integration Of Immersive Environment And RULA For Real-time study of Workplace Related Musculoskeletal Disorders In The Upper Lime" Proceedings of DETC'03 ASME 2003 DETC-CIE, Chicago, Illinois, Sep. 2-6.
30. **Jayaram, U., Jayaram, S., Shaikh, I., Kim, Y., and Palmer, C.,** 2006, "Introducing quantitative analysis methods into virtual environments for real-time and continuous ergonomic evaluations", Computers in Industry, Vol.57, pp.283-296.
31. **Jayaram, S., Jayaram, U., and Yang, Y.,** 2004, "A Distributed Virtual Assembly

Environment Using CORBA", IJAM International Journal of Agile Manufacturing, Vol.7, No.2.
32. The OMG's CORBA Website, www.corba.org
33. **Gowda, S., Jayaram, S., and Jayaram, U.,** 1999, "Architectures For Internet-Based Collaborative Virtual Prototyping" Proceedings of the 1999 ASME DETC, Las Vegas, Nevada, September 12-15.
34. **Craig, J. J.** Introduction to Robotics, Second ed, vol. Addison-Wesley.
35. **Noy, N. F., and McGuinness, D. L.,** 2001. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory.

APPENDIX A

DATA STRUCTURES

```
// A structure for a bird device's data
typedef struct _VrBirdsData {
        float fPosX;            // X position
        float fPosY;            // Y position
        float fPosZ;            // Z position
        float fPitch;           // Pitch
        float fRoll;            // Roll
        float fYaw;             // Yaw
} VR_BIRDS_DATA, *PVR_BIRDS_DATA;

// A structure for a bird device's status and configuration
typedef struct _VrBirdsStatus {
        byte    bStatus;        // Status
        byte    bExpandedMode;
        byte    bHostSyncMode;
        byte    bCrtSyncMode;
        byte    bXonOrXoff;
        byte    bRunMode;
        byte    bSteamMode;
        byte    bGroupMode;
        byte    bFlockStatus[32];
        char    szVersion[16];
        char    szErrorCode[16];
} VR_BIRDS_STATUS, *PVR_BIRDS_STATUS;

// A structure for a bird device's capabilities
typedef struct _VrBirdsCaps {
        byte    bDeviceNum;
        byte    bXmtrAddr;
        byte    bDataFormat;
        byte    bHemisphere;
```

```
        byte    bXmtrType;
        byte    bReportRate;
        int     nXtalSpeed;
        int     nScaling;
        double  dMeasurementRate;
} VR_BIRDS_CAPS, *PVR_BIRDS_CAPS;

// A structure to hold all the information about a bird device
typedef struct _VrBirdsStruct {
        VR_BIRDS_STATUS   brdStatus;
        VR_BIRDS_CAPS             brdCaps;
        VR_BIRDS_DATA             brdData;
} VR_BIRDS_STRUCT, *PVR_BIRDS_STRUCT;

// A structure for a glove's data
typedef struct _VrGloveData {
        float   fRawAngle[23];// Original data got from the cyber-glove
        float   fCalibAngle[23];        // Calibrated data using the interpolation
method
} VR_GLOVE_DATA, *PVR_GLOVE_DATA;

// A structure for the status of a glove device
typedef struct _VrGloveStatus   {
        byte    bDevStatus;             // Device Status
        int     nSwitchStatus;  // Status of a switch button
        int     nNumOfSensors;          // Number of sensors
        char    szVersion[16];          // Version
        char    szErrorCode[16];        // The last error code
} VR_GLOVE_STATUS, *PVR_GLOVE_STATUS;
```

```
// A structure for the capability of a glove device
typedef struct _VrGloveCaps {
        boolean bIsSwitch;      // Has a switch buuton
} VR_GLOVE_CAPS, *PVR_GLOVE_CAPS;

// A structure to hold all the information about a glove device
typedef struct _VrButtonStruct {
        VR_GLOVE_STATUS  glvStatus;
        VR_GLOVE_CAPS           glvCaps;
        VR_GLOVE_DATA           glvData;
} VR_GLOVE_STRUCT, *PVR_GLOVE_STRUCT;

// Astructure for a button device's data which is similar to the circular queue
typedef struct _VrGloveData {
        int             nHeadPtr;               // Head position
        int             nTailPtr;               // Tail position
        char            bBtnKey[256];   // Key buffer
} VR_BUTTON_DATA, *PVR_BUTTON_DATA;

// A structure for a button device's staus
typedef struct _VrButtonStatus {
        int             nStatus;        // Status
        char            szName[128];            // Device name
} VR_BUTTON_STATUS, *PVR_BUTTON_STATUS;

// A structure for a button device's capability
typedef struct _VrButtonCaps {
        unsigned int    nBtnNumber;             // The number of button
} VR_BUTTON_CAPS, *PVR_BUTTON_CPAS;
```

```
// A structure to hold all the information about a button device
typedef struct _VrButtonStruct {
        VR_BUTTON_STATUS        btnStatus;
        VR_BUTTON_CAPS          btnCaps;
        VR_BUTTON_DATA  btnData;
} VR_BUTTON_STRUCT, *PVR_BUTTON_STRUCT;

// A structure for an object's status
typedef struct _VrObjectStatus {
        unsigned long           lMask;          // Mask filter
        int                     nStatus;// Status
        char                    szName[128];    // Object name
} VR_OBJECT_STATUS, *PVR_OBJECT_STATUS;

// A structure for an object's capability and properties
typedef struct _VrObjectCaps {
        unsigned int            nColor; // Color
        float                   fWeight;        // Weight
} VR_OBJECT_CAPS, *PVR_OBJECT_CPAS;

// A structure for an object's data
typedef struct _VrObjectData {
        int             nLastEvent;             // The last notification event
        float           fXform[16];             // Transformation matrix
} VR_OBJECT_DATA, *PVR_OBJECT_DATA;

// A structure to hold all the information about an object
typedef struct _VrObjectStruct {
        VR_OBJECT_STATUS objStatus;
        VR_OBJECT_CAPS          objCaps;
        VR_OBJECT_DATA          objData;
```

```
} VR_OBJECT_STRUCT, *PVR_OBJECT_STRUCT;

// The maximum number of glove devices
#define VR_GLOVE_MAX                2

// The maximum number of bird devices
#define VR_BIRDS_MAX                2

// The maximum number of button devices
#define VR_BIRDS_MAX                1

// The maximum number of objects
#define VR_OBJECT_MAX               12

// A structure representing a shared memory
typedef struct _VrSharedMem {
        // A structure for a glove's information
        VR_GLOVE_STRUCT  vrGloveStruct[VR_GLOVE_MAX];

        // A structure for a bird device's information
        VR_BIRDS_STRUCT   vrBirdsStruct[VR_BIRDS_MAX]

        // A structure for a button device's information
        VR_BUTTON_STRUCT          vrButtonStruct[VR_BUTTON_MAX];

        // A structure for an object information
        VR_OBJECT_STRUCT vrObjectStruct[VR_OBJECT_MAX];
} VR_SHM_DEVICES, *PVR_SHM_DEVICES;
```

```
// The status of a communication device
// IDLE : The communication device is waiting for new command.
// RUN    : The communication device should execute a command.
// BUSY : The communication device is busy during operating.
// DOWN : The communication device has a hardware problem.
#define VR_COMM_IDLE                0x0000
#define VR_COMM_RUN                 0x0001
#define VR_COMM_BUSY                0x0002
#define VR_COMM_DOWN                0x0004

// The status of a return value
// SUCCESS : The last operation was completed successfully
// FAIL : The last operation was failed with some errors
// WAIT : The device is busy during operation
// NOTSUPPORT : Unknown problem.
#define VR_RET_SUCCESS              0x0000
#define VR_RET_FAIL         0x0001
#define VR_RET_WAIT                 0x0002
#define VR_RET_UNKNOWN  0xFFFF

// The structure for communicating between an application and a communication driver. //
If you would like to send some data into the communication device,
// first you should put VR_COMM_RUN value into wSendFlag and check if wRecvFlag
// is changed into VR_RET_SUCCESS or VR_RET_FAIL. While the device is
// executing the command which you send, the wSendFlag will be changed into
// VR_COMM_BUSY. Whenever the wSendFlag is changed into VR_COMM_RUN,
// the driver will try to execute the command. After completing the mission, the driver
returns value(VR_RET_SUCCESS or VR_RET_FAIL) to wRecvFalg.
typedef struct    _tagCommMem {
        unsigned char    szSocketAddr[24];
        unsigned int     nSocketPort;
```

```
        unsigned int      nSerialPort;
        unsigned int      nSerialBaud;

        unsigned short    wSendFlag;
        unsigned short    wSendSize;
        unsigned char     bSendBuff[COMM_BUFF_SIZE];

        unsigned short    wRecvFlag;
        unsigned short    wRecvSize;
        unsigned char     bRecvBuff[COMM_BUFF_SIZE];
} VR_COM_MEM, *PVR_COM_MEM;
```

APPENDIX B

Core Code for the VADE-Side

```
// data struct definition
typedef struct {
        int current_state;
        int event;
        int next_state;
        void (*action) (CInteractionManager *);
} StateEventTable;

// state values
enum {
        ST_REACHFOR,       // Reach for a part to be assembled
        ST_ASSEMBLE,       // assemble process by hand
};

// event values
enum {
        EV_RELEASE  = 0,      // release a part from hand
        EV_GRIP               = 1,      // grip a part by hand
};
// Define action functions here
void actAssemblePart(CInteractionManager* i_pInteractManager);   // by hand
void actReleasePart(CInteractionManager* i_pInteractManager);    // by hand

// configuration values of state-event table
StateEventTable g_tblStateEvent[] = {
        //current_state          event                default next_stat    function
        {ST_REACHFOR,   EV_GRIP,          ST_ASSEMBLE,   actAssemblePart },
        {ST_ASSEMBLE,   EV_ASSEMBLE,   ST_ASSEMBLE,   actAssemblePart  },
        {ST_ASSEMBLE,   EV_RELEASE,     ST_REACHFOR,    actReleasePart},
        { -1,           -1,             -1,             0     }   // end of table
};
```

```
int CInteractionManager::interact(void)
{
	// Declare local variables
	//----------------------------
	float fMatrix[16];
	int nEvent = -1;
	int i;

	if ((NULL == d_pMyInputManager) || (NULL == d_pMyOutputManager)) {
		return -1;
	}

	// Get bird and glove data from InputManager
	//-------------------------------------------------------
	d_pMyInputManager->getFlockData(d_pFlockData);
	d_pMyInputManager->getGloveData(d_pGloveData);

	// Update the location and orientation of the hand
	//-------------------------------------------------------------
	// d_pMyModelManager->updateHand(d_pFlockData, d_pGloveData);
	d_pMyInputManager->getLocalBirdData(1, fMatrix);
	d_pMyOutputManager->updateDisplay(d_pFlockData, d_pGloveData);

	// Get a new event
	//--------------------
	nEvent = getEvent();
	if (nEvent >= 0) {
		// look for table to decide which action should be done according to the
		// event and current state
		//-----------------------------------------------------------------------------
		i = 0;
```

```
                while (g_tblStateEvent[i].current_state != -1) {
                        if ((nEvent == g_tblStateEvent[i].event) &&
                                (d_nCurState == g_tblStateEvent[i].current_state)) {
                                // Get the default next state
                                //-----------------------------
                                d_nCurState = g_tblStateEvent[i].next_state;

                                // Do action defined in the table
                                //-----------------------------------
                                (*g_tblStateEvent[i].action) (this);
                                break;
                        }
                        i++;
                }
        }

        // Update parts on the screen based on data from VR devices
        //-------------------------------------------------------------------
        d_pMyModelManager->updatePart
                        (d_pMyConstraintManager->getCurrentPart());

        return 0;
}

// Check and get an event
int CInteractionManager::getEvent(void)
{
        // Declare local variables
        //----------------------------
        static int nFrameDelay = 10;
```

```
 int        i;
 int        nIntersectHand = 0;
 int        nEvent = -1;
 // The distance between a thumb and an index finger
 float      fDistThumbIndex = 0;
// The distance between a thumb and an middle finger
float       fDistThumbMiddle = 0;
 char       szMoterCheck[5];
 int        nObjId;
 pfMatrix           matHand, matPalm, matPart, matBasePart;
 static pfMatrix matTot;

 CHand* pHand = d_pMyModelManager->getHand();
 List*     pListParts = d_pMyModelManager->getPartsList();
 CPart* pPart = (CPart *)pListParts->getHead();

 // Check the event queue whether there are events available
 //-------------------------------------------------------------------------
 if ((nEvent = popEventFromQueue()) >= 0)   return nEvent;

 //----------------------------
 // Check the current status
 //----------------------------

 // Check whether the hand approach one of parts
 //---------------------------------------------
 if (d_nCurState == ST_REACHFOR) {
          nIntersectHand = pHand->checkIntersections (&fDistThumbIndex,
                &fDistThumbMiddle,    d_pMyOutputManager->getGlobalDCS(),
                szMoterCheck, d_nPrevGripStatus);
```

```
d_nPrevGripStatus = d_nCurGripStatus;
d_nCurGripStatus=d_pMyConstraintManager->
    checkGrip(nIntersectHand, d_pMyOutputManager, d_nGripFlag);

if (d_nCurGripStatus == 1) {
        nEvent = EV_GRIP;

        // Send the part transformation matrix to JACK
        //-----------------------------------------------------
        pPart = (CPart *)pPart->getNext();          // gear
        pPart->getCurrentXform(&matPart);
        pHand->getHandXform(&matHand);
        pHand->getPalmXform(&matPalm);
        nObjId = pPart->getModel_Id();
        matTot.mult(matPart, matPalm);
        matTot.postMult(matHand);
        d_pMyInputManager->
            setObjectData(nObjId,EV_GRIP, matTot);
}
else {
        pPart->getDCS()->getMat(matBasePart);
        pPart = (CPart *)pPart->getNext();          // gear
        if((pPart->getState() == INSPACE) ||
          (pPart->getState() == STATIC)) {
                if (pPart->isPartPlaced() == 1) {
                        pPart->getDCS()->getMat(matPart);
                        matTot.mult(matPart, matBasePart);
                        nObjId = pPart->getModel_Id();
                        d_pMyInputManager->setObjectData
                                (nObjId,EV_ASSEMBLE,matTot);
                        }
```

```
                        else {
                                pPart->getCurrentXform(&matTot);
                                nObjId = pPart->getModel_Id();
                                d_pMyInputManager->setObjectData
                                    (nObjId, EV_RELEASE, matTot);
                        }
                }
        }
}
else if (d_nCurState == ST_ASSEMBLE) {
        if (checkRelease(d_pGloveData[0]) == 1) {
                d_nCurGripStatus = 0;
                nEvent = EV_RELEASE;

                //   Send the part transformation matrix to JACK
                //-------------------------------------------------------
                pPart = (CPart *)pPart->getNext();          // gear
                pPart->getCurrentXform(&matPart);
                pHand->getHandXform(&matHand);
                pHand->getPalmXform(&matPalm);
                matTot.mult(matPart, matPalm);
                matTot.postMult(matHand);
                nObjId = pPart->getModel_Id();
                d_pMyInputManager->setObjectData
                                (nObjId, EV_RELEASE, matTot);
        }
        else {
                // Send the part transformation matrix to JACK
                //-------------------------------------------------------
                pPart = (CPart *)pPart->getNext();          // gear
                nEvent = EV_ASSEMBLE;
```

```
                pPart->getCurrentXform(&matPart);
                pHand->getHandXform(&matHand);
                pHand->getPalmXform(&matPalm);
                matTot.mult(matPart, matPalm);
                matTot.postMult(matHand);
                nObjId = pPart->getModel_Id();
                d_pMyInputManager->
                        setObjectData(nObjId, EV_GRIP, matTot);
            }
        }

        return nEvent;
}

// Assemble a part to base part     Wu added 20-June
void actAssemblePart(CInteractionManager* i_pInteractMan)
{
        COutputManager* pOutputMan = i_pInteractMan->getOutputManager();
        CConstraintManager * pConstraintMan
                                = i_pInteractMan->getConstraintManager();

        if (pConstraintMan->assemblePart(pOutputMan)) {
            // if finish assembling current handle part, set state to ST_REATCHFOR
            // for reaching for another part
            //-------------------------------------------------------------------------------
            i_pInteractMan->setState(ST_REACHFOR);

            // Send the part transformation matrix to JACK
            //-------------------------------------------------------
            int                 nObjId;
            int row, column;
```

```
                pfMatrix        matHand, matPalm, matPart, matTot, matBasePart,
                                matTemp1;
                CHand* pHand = i_pInteractMan->getModelManager()->getHand();
                List* pListParts=i_pInteractMan->getModelManager()->getPartsList();
                // Base Part
                CPart* pPart = (CPart *)pListParts->getHead();
                //pPart->getCurrentXform(&matBasePart);
                pPart->getDCS()->getMat(matBasePart);
                pPart = (CPart *)pPart->getNext();      // gear
                pPart->getDCS()->getMat(matPart);
                //pPart->getCurrentXform(&matPart);
                matTot.mult(matPart, matBasePart);

                nObjId = pPart->getModel_Id();
                i_pInteractMan->getInputManager()->
                                setObjectData(nObjId, EV_ASSEMBLE, matTot);
        }
}

// Release a part.
void actReleasePart(CInteractionManager* i_pInteractMan)
{
        COutputManager* pOutputMan = i_pInteractMan->getOutputManager();
        CConstraintManager* pConstraintMan
                                = i_pInteractMan->getConstraintManager();
        pConstraintMan->releasePart(pOutputMan);
}
```

APPENDIX C

Core Code in JackScript

```
# VRCIM Includes
#----------------
import VrTimer
import VrDeviceManager
import VrHumanModel
import VrObjectModel

g_bVadeAlive = 1     # A boolean value to decide whether a test driver is running or stopped

class ErgoVade:
        """
        An instance of the ErgoVade class decides the life of this application.
       In simpler words, when you create an instace, call initVade() function,
       and call runMain() function, the application will begin. On the other hand,
       when you call destroyVade() function, this program will be terminated.
        """

        def initVade(self):
                """
                FUNCTION       : initVade
                PURPOSE        : Initialize all the resources.
                                    Create a bird object and a human object.
                """

                # Local Variables
                #-----------------
                ptrHuman = None
                ptrObject = None
                self.d_nGripStatus = -1
```

```
# Connect to all the cyber devices
#---------------------------------
self.d_myDeviceMan.connectDevices()

# Initialize a human model
#-------------------------
self.d_objHuman.initHuman()
self.d_objTimer.Sleep(0.5)
ptrHuman = self.d_objHuman.getHumanPtr()
hss = ReadHandShapeFile('handshapes.data')
hs = hss['pinch']
hs.Apply(ptrHuman, 'left')

# Later, I will create a ModelManager Class
objPart = VrObjectModel.PyVrObject()
objPart.initObject("shaft", "shaft.fig")
objPart.setScale(100, 100, 100)
self.d_lstParts.append(objPart)

objPart = VrObjectModel.PyVrObject()
objPart.initObject("gear", "gear1.fig")
objPart.setScale(100, 100, 100)
self.d_lstParts.append(objPart)

objPart = VrObjectModel.PyVrObject()
objPart.initObject("bird_lefthand", "bird.fig", 0)
objPart.setScale(0.5, 0.5, 0.5)
self.d_lstParts.append(objPart)
cons1 = CreateConstraint(effector=ptrHuman.left_palm.base, \
                    goal=self.d_lstParts[2].d_figObject.bird11.base, \
                joint=ptrHuman.left_shoulder, poweight = 0.3)
```

```
        joint=ptrHuman.left_shoulder, poweight = 0.3)

        objPart = VrObjectModel.PyVrObject()
        objPart.initObject("bird_righthand", "bird.fig", 0)
        objPart.setScale(0.5, 0.5, 0.5)
        self.d_lstParts.append(objPart)
        cons2 = CreateConstraint(effector=ptrHuman.right_palm.base, \
                        goal=self.d_lstParts[3].d_figObject.bird11.base, \
                        joint=ptrHuman.right_shoulder, poweight=0.3)

def destroyVade(self):
        """
        FUNCTION       : destroyVade
        PURPOSE        : Uninitialize all the resources
        """
        # Uninitialize a bird object
        #---------------------------
        self.d_myDeviceMan.disconnectDevices()
        self.d_objHuman.destroyHuman()

def runMain(self):
        """
        FUNCTION       : runMain
        PURPOSE        : The main loop.
                         The program will be terminated when this function returns
        """
        # Instead of that, the test manager below call this fuction infinitely
        #-----------------------------------------------------------------------
        lstActions = []
        matGlobalBird = Matrix4()
        matGlobalModel = Matrix4()
```

```
matOffset = Matrix4()
matTot = Matrix4()

#-----------------------------------------------------------------------------
# Move and update all the joints in the human model.
# Actually, the function like updateRightShoulder will return an acation
# object. We can execute actions sequecially or together using
# DoInOrder() or DoTogether()    functions respectively.
#-----------------------------------------------------------------------------

# Update the location and joint angles of the right hand
#---------------------------------------------------------------
tupGloveData = self.d_myDeviceMan.getRightGloveData()
actHand = self.d_objHuman.updateRightHand(tupGloveData)
lstActions.extend(actHand)

# The angles of the left wrist will be used the values of the third bird
#-----------------------------------------------------------------------------
matGlobalBird = self.d_myDeviceMan.getBirdData(1)
matOffset.SetAxes([0, 1, 0], [0, 0, 1], [1, 0, 0])
matTot = matOffset * matGlobalBird
actBird1 = self.d_lstParts[2].update(matTot)
lstActions.append(actBird1)

# The angles of the right wrist will be used the values of the third bird
#-----------------------------------------------------------------------
matGlobalBird = self.d_myDeviceMan.getBirdData(2)
matOffset.SetAxes([0, -1, 0], [0, 0, -1], [1, 0, 0])
matTot = matOffset * matGlobalBird
actBird2 = self.d_lstParts[3].update(matTot)
lstActions.append(actBird2)
```

```
# Send the location of the right wrist
# The angles of the wrist will be used the values of the third bird
#-----------------------------------------------------------------
pHuman = self.d_objHuman.getHumanPtr()
matRWrist = pHuman.joint.right_wrist.GetLocation()
tupRawBirdData = self.d_myDeviceMan.getRawBirdData(2)
pos = matRWrist.GetTranslation()
ang = tupRawBirdData[4:7]
self.d_myDeviceMan.setBirdData(4, pos[2] / 2.54,
                        -pos[0]/2.54, -pos[1]/2.54, ang[0], ang[1], ang[2])

# Update the location of the human model.
#------------------------------------------------
matGlobalBird = self.d_myDeviceMan.getBirdData(3)
actBody = self.d_objHuman.updateBody(matGlobalBird)
lstActions.append(actBody)

# Update the parts
#-----------------
actParts = self.getEventAndUpdateParts()
lstActions.extend(actParts)

# Execute all the actions using DoInOrder() or DoTogether() function.
#-----------------------------------------------------------------
tupActions = tuple(lstActions)
DoInOrder(tupActions)
#DoTogether(tupActions)
#apply(DoInOrder, tupActions)

# Update the screen
```

```
        Flush()

def getEventAndUpdateParts(self):
        """
        FUNCTION       : getEventAndUpdateParts
        PURPOSE        : Check the events receiving from the VADE
                                    0 : Released
                                    1 : Gripped
                                    2 : Assembled
        """
        lstActions = []
        matGlobalBird = Matrix4()
        matGlobalModel = Matrix4()

        # Update the base part
        #----------------------
        matGlobalBird = self.d_myDeviceMan.getBirdData(1)
        actPart0 = self.d_lstParts[0].update(matGlobalBird)
        lstActions.append(actPart0)

        # Update the part lists
        #-----------------------
        tupObjData = self.d_myDeviceMan.getObjectXform(1)
        matGlobalModel.SetAxis(0, tupObjData[1:4])
        matGlobalModel.SetAxis(1, tupObjData[5:8])
        matGlobalModel.SetAxis(2, tupObjData[9:12])
        matGlobalModel.SetTranslation(tupObjData[13]*2.54,
                                    tupObjData[14]*2.54, tupObjData[15]*2.54)

        # Check the event getting from the VADE
        #-----------------------------------------------
```

```
nEvent = self.d_myDeviceMan.getObjectEvent(1)
if nEvent == 0:
        print '<EVENT> Part is released.'
        self.d_lstParts[1].d_figObject.AttachTo(None)

        actPart1 = self.d_lstParts[1].update(matGlobalModel)
        lstActions.append(actPart1)

        # Turn off the constraints
        #-----------------------------
        self.d_lstParts[0].displayConstraints(on=False)
        self.d_lstParts[1].displayConstraints(on=False)
elif nEvent == 1:
        print '<EVENT> Part is grabbed.'
        actPart1 = self.d_lstParts[1].update(matGlobalModel)
        lstActions.append(actPart1)

        #Turn on the constraints
        #---------------------------
        self.d_lstParts[0].displayConstraints(on=True)
        self.d_lstParts[1].displayConstraints(on=True)
elif nEvent == 2:
        print '<EVENT> Part is assembled.'
        self.d_lstParts[1].d_figObject.AttachTo(None)

        actPart1 = self.d_lstParts[1].update(matGlobalModel)
        lstActions.append(actPart1)

        # Turn off the constraints
        #-------------------------
        self.d_lstParts[0].displayConstraints(on=False)
```

```
            self.d_lstParts[1].displayConstraints(on=False)
    else:
            print '<EVENT> Unknown event.'

    return lstActions
```

APPENDIX D

Core Code in Device Manager

```
void CVrDevServerView::OnInitialUpdate()
{
        m_pDoc = (CVrDevServerDoc*)(((CMainFrame*)AfxGetMainWnd())->
                                        GetActiveDocument());

        m_wEnabledDevices = 0;
        m_wEnabledNetClients = 0;
       initializeAllComponents();

        //Create shared memory for shared struct
        m_hShmDevices = (HANDLE)::CreateFileMapping((HANDLE)0xFFFFFFFF,
                                                NULL,
                                                PAGE_READWRITE,
                                                0,
                                                sizeof(VR_SHM_DEVICES),
                                                VR_SHM_DEVICES_NAME);
        if(NULL == m_hShmDevices) {
                _writeTrace(1, _T("CVrDevServerView::OnInitialUpdate():
                        Fail to create the memory-mapped file. ErrorCode=%08X"),
                           ::GetLastError());
                return;
        }
        m_pShmDevices = (PVR_SHM_DEVICES)::MapViewOfFile(m_hShmDevices,
                                        FILE_MAP_ALL_ACCESS, 0, 0, 0 );
        if(NULL == m_pShmDevices) {
                _writeTrace(1, _T("CVrDevServerView::OnInitialUpdate():
                        Fail to view the memory-mapped file. ErrorCode=%08X"),
                           ::GetLastError());

                ::CloseHandle(m_hShmDevices);
                m_hShmDevices = NULL;
```

```
                return;
        }
        memset(m_pShmDevices, 0x00, sizeof(VR_SHM_DEVICES));

        return;
}

int CVrDevServerView::_sendDataToNetClients(void)
{
        int i;
        PBYTE pbBuff = new BYTE[1024];
        WORD wLen = 0;

        memset(pbBuff, 0x00, 1024);
        if ((m_wEnabledNetClients & VR_DEV_RGLOVE) &&
                (m_wEnabledDevices & VR_DEV_RGLOVE)) {
                for (i=0; i<23; i++) {
                        wLen += sprintf((char*)&pbBuff[wLen], "%f ",
                                m_pShmDevices->vrRightGloveData.fCalibAngle[i]);
                }
                wLen = 255; // Set the size as 255 forcefully
                // Send data over the network
                //-----------------------------------
                m_pNetGlove->Send(pbBuff, (int)wLen);
        }

        memset(pbBuff, 0x00, 1024);
        if ((m_wEnabledNetClients & VR_DEV_BIRDS) &&
                (m_wEnabledDevices & VR_DEV_BIRDS)) {
                for (i=0; i<m_pShmDevices->vrBirdsCaps.bDeviceNum-1; i++) {
                        sprintf((char*)pbBuff, "%d %f %f %f %f %f %f", i,
```

```
                    m_pShmDevices->vrBirdsData[i].fPosX * (-1.0f) + 10.0f,
                    m_pShmDevices->vrBirdsData[i].fPosY * (-1.0f) ,
                    m_pShmDevices->vrBirdsData[i].fPosZ * (1.0f) - 28.0f,
                    m_pShmDevices->vrBirdsData[i].fYaw + 180.f,
                    m_pShmDevices->vrBirdsData[i].fPitch,
                    m_pShmDevices->vrBirdsData[i].fRoll);

                    wLen = 255; // Set the size as 255 forcefully
                    // Send data over the network
                    //---------------------------------
                    m_pNetBirds->Send(pbBuff, (int)wLen);
          }
}

memset(pbBuff, 0x00, 1024);
if (m_wEnabledNetClients & VR_DEV_VADE) {

memset(pbBuff, 0x00, 1024);
if ((m_wEnabledNetClients & VR_DEV_BUTTON) &&
          (m_wEnabledDevices & VR_DEV_BUTTON)) {
          PVR_BUTTON_STRUCT pStruct =
                    &(m_pShmDevices->vrButtonStruct);
          if (pStruct->btnData.nHeadPtr != pStruct->btnData.nTailPtr) {
               pStruct->btnData.nTailPtr=(++pStruct->btnData.nTailPtr)% 256;
               bData1 = pStruct->btnData.bBtnKey[pStruct->btnData.nTailPtr];
               pStruct->btnData.nTailPtr = (++pStruct->btnData.nTailPtr) % 256;
               bData2 = pStruct->btnData.bBtnKey[pStruct->btnData.nTailPtr];
               wLen = wsprintf((LPSTR)pbBuff, "%d %d", bData1, bData2);

               // Send data over the network
               //---------------------------------
```

```
                        m_pNetButton->Send(pbBuff, (int)wLen);
                }
        }

        delete pbBuff;
        return 0;
}

// A thread for updating the birds' data
DWORD WINAPI UpdateBirdsThreadProc(LPVOID lpParam)
{
        static BOOL bFirstTime = TRUE;
        CVrDevServerView *pView = (CVrDevServerView *)lpParam;

        while (pView->m_bBirdsConnected) {
                if (TRUE == bFirstTime) {
                        bFirstTime = FALSE;
                        VrBirds_GetDevStatus(&pView->m_pShmDevices
                                                ->vrBirdsStatus);
                        VrBirds_GetDevCaps(&pView->m_pShmDevices->vrBirdsCaps);
                }

                VrBirds_GetDevData(&pView->m_pShmDevices->vrBirdsData[0]);
                ::Sleep(10);
        }

        return 0;
}
```

```
// A thread for updating the glove's data
DWORD WINAPI UpdateGloveThreadProc(LPVOID lpParam)
{
	static BOOL bFirstTime = TRUE;
	CVrDevServerView *pView = (CVrDevServerView *)lpParam;
	static int nCount = 0;

	while (pView->m_bGloveConnected) {
	   if (++nCount >= 100) {
	      nCount = 0;
	      VrGlove_GetDevStatus(&pView->m_pShmDevices->vrRightGloveStatus);
	      VrGlove_GetDevCaps(&pView->m_pShmDevices->vrRightGloveCaps);
	   }

	   VrGlove_GetDevData(&pView->m_pShmDevices->vrRightGloveData);

	   ::Sleep(10);
	}

	return 0;
}
```

APPENDIX E

Statistics Source Code and Description

```
DATA Speed;
INPUT OS Comm Models Second;
CARDS;
/* First Column : 1= Windows, 2=Linux */
/* Second Column : 1=RS232C, 2=TCP */
/* Third Column : The Number of Models */
1       1       2       31
1       1       2       30
1       1       5       31
1       1       5       31
1       1       10      30
1       1       10      47
1       1       20      46
1       1       20      46
1       2       2       32
1       2       2       32
1       2       5       32
1       2       5       32
1       2       10      47
1       2       10      31
1       2       20      46
1       2       20      46
2       1       2       30
2       1       2       30
2       1       5       31
2       1       5       30
2       1       10      31
2       1       20      46
2       1       20      46
2       2       2       31
2       2       2       32
```

2 2 5 30

2 2 5 31

2 2 10 31

2 2 10 46

2 2 20 46

2 2 20 46

2 2 20 46

```
PROC glm;
class OS Comm Models;
model Second=OS|Comm|Models;
means OS Comm Models/tukey;
run;
```

Step 1. Model Identification

Three fixed factor design

A : Operating System (i = 1, 2)

B : Communication Type (j = 1, 2)

C : The number of models (k = 1, 2, 3, 4)

Response variable: frame-rate time (milliseconds)

Step 2. Statistical linear model

$X_{ijkl} = u + \alpha_i + \beta_j + \delta_k + \gamma^{AB}_{ij} + \gamma^{AC}_{ik} + \gamma^{BC}_{jk} + \gamma^{ABC}_{ijk} + \varepsilon_{ijkl}$

X_{ijkl} : Total time of i^{th} level of Operating System, j^{th} level of a communication type, k^{th} level of model numbers , and l^{th} level of replications.

Step 3. Asumptions

$\Sigma\alpha_i = 0, \Sigma \beta_j = 0, \Sigma \delta_k = 0, \quad \Sigma\Sigma \gamma^{AB}_{ij} = 0, \Sigma\Sigma \gamma^{AC}_{ik} = 0, \Sigma\Sigma \gamma^{BC}_{jk} = 0, \Sigma\Sigma\Sigma \gamma^{ABC}_{ijk} = 0.$

Step 4. Hypothesis

Null hypothesis: $\alpha_i = 0, \quad \beta_j = 0, \quad \delta_k = 0, \quad \gamma^{AB}_{ij} = 0, \quad \gamma^{AC}_{ik} = 0, \gamma^{BC}_{jk} = 0, \gamma^{ABC}_{ijk} = 0.$

There is no significant effect due to the main effect of A (Operating System), B (a communication type), C (the number of models), and the first interaction effect between A and B, A and C, B and C, and second interaction effect between A, B, C.

Alternate hypothesis: $\alpha_i \neq 0, \quad \beta_j \neq 0, \quad \delta_k \neq 0, \gamma^{AB}_{ij} \neq 0, \quad \gamma^{AC}_{ik} \neq 0, \gamma^{BC}_{jk} \neq 0, \gamma^{ABC}_{ijk} \neq$ **0. (at least one i, j, k)**

There is significant effect due to the main effect of A (Operating System), B (a communication type), C (the number of models), and the first interaction effect between A and B, A and C, B and C, and second interaction effect between A, B, C.

Step 5. Statistical analysis

ANOVA Table

Source	Degree of freedom	Sum of Square	Mean square	F-Value	Pr>F
OS	1	608.40	608.40	19.13	<.0001**
Comm.	1	372.10	372.10	11.70	0.0008**
OS*Comm.	1	225.63	226.63	7.09	0.0086**
Models	3	9328.43	3109.48	97.78	<.0001**
OS*Models	3	635.95	211.98	6.67	0.0003**
Comm.*Models	3	242.85	80.95	2.55	0.0584
OS*Comm.*Models	3	103.225	34.41	1.08	0.3588

Step 6. Conclusion

- Accept null hypotheses: The 1st interaction between communication type and the number of models and the 2nd interactions between all factors
- Reject null hypotheses: The main effects due to operating system, communication type and the number of models and the 1st interaction between operating system and communication type and between operating system and the number of models

We can conclude that there are significant differences among frame-rate time at various levels of operating system, communication type, and the number of models and the 1st interactions between operating system and communication type and between operating system and the number of models