# OPEN ARCHITECTURE FOR EMBEDDING VR BASED MECHANICAL TOOLS IN CAD

By

HRISHIKESH S. JOSHI

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

December 2006

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of HRISHIKESH S. JOSHI find it satisfactory and recommend that it be accepted.

_____

Chair

_____

_____

# ACKNOWLEDGMENTS

I would like to thank The Almighty, my parents and my beloved wife Rupali for their help and support. I would also like to thank my advisor Dr. Sankar Jayaram and Dr. Uma Jayaram for their guidance and support during my thesis work. Also, a special thanks to Dr. Hakan Gurocak for being on my committee. I will also like to thank Mr. Larry Varoz from Sandia National Laboratories for sponsoring the research.

I want to thank all my friends at WSU, and IES who have been very helpful throughout my stay in Pullman.

**OPEN ARCHITECTURE FOR EMBEDDING VR BASED**

**MECHANICAL TOOLS IN CAD**

# Abstract

By Hrishikesh S Joshi, M.S.
Washington State University
December 2006

Chair: Sankar Jayaram

Virtual assembly technology is revolutionizing the way we design mechanical assemblies. This technology enables the users to interact with virtual models of mechanical assemblies in an immersive environment. The user interacts with the virtual environment via various hardware devices, which can track motion and other user inputs. These devices Haptic devices are also capable of providing touch and force feedback to the user. The existing virtual assembly systems however are not tightly integrated with the CAD systems that are primarily used for modeling the assemblies. This disconnect between the CAD systems and virtual reality (VR) environments is one of the most important limitation in the existing virtual assembly systems. In this thesis we propose an open architecture for embedding VR based virtual mechanical tools in the CAD systems. The goal of this research is to enhance the CAD experience by allowing the user to interact with the CAD assemblies inside a virtual environment in a more realistic manner.

The open architecture is designed to leverage off of the inherent visualization capabilities provided by any commercial CAD system. The architecture consists of a CAD plugin module, which is primarily responsible for interacting with the CAD system and handling the visualization through the application programming interface (API) exposed by

the CAD system. The other significant module of the proposed architecture is the Graphics-Haptics bridge (GHBridge), which is built on top of the haptics API. The GHBridge enables the integration of haptic devices with diverse visualization systems. Thus, the CAD plugin and the GHBridge together facilitate tight integration between the CAD system and the haptic devices. We also propose a CAD model library structure for managing virtual tools that can be used for assembly operations in the virtual environment.

The proposed architecture is implemented to provide virtual assembly functionality for CATIA V5 CAD system and VirtualHand™ toolkit haptics API. This implementation (VR Tools) supports assembly and disassembly operations on CATIA assemblies that use threaded fasteners. The user can choose from available virtual tools to manipulate compatible fasteners in the virtual environment.

TABLE OF CONTENTS

# LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

Virtual Reality (VR) has excited researchers for over several decades. The technology though has fallen short on delivering on the promises and fulfilling the visions due to limitations imposed by the available computing power at disposal of the researchers. In recent times VR systems have been getting better with improvements in the available computing capabilities and due to research in the field. VR systems are finding a wide range of applications in fields of engineering, medicine, defense and gaming. A significant amount of research is being conducted in all those fields for improving the VR technology and making the VR experience more realistic. The automotive industry has been one of the first to use VR systems and environments in the design-concept phases for new models. VR environments have also been effectively used in ergonomics studies. Mechanical engineering companies have supported research activities in the field of virtual assembly- disassembly. These tools have enabled engineers to study the assembly- disassembly processes intuitively.

A very basic assembly- disassembly operation that is typically simulated in a VR system involves real-time human interaction using hands with the assembly components in a VR environment. The next level of complexity comes when assembly- disassembly operations are carried out using virtual tools. In this kind of a system, the human hand has to be able to manipulate a tool and the tool in turn has to manipulate an assembly component. The system can be developed further to make it more realistic by supporting usage of haptic devices for providing touch and force feedback while working in the VR environments. These kinds of VR environments that provide engineers with the ability to study assembly

operations offer lot more insight into potential design flaws from the viewpoint of design for assembly. The engineers can get feedback on basic unknowns such as: is the part accessible to the tool? Is there enough space for the tool motion during the assembly process? Is there enough clearance for the hand to operate the tool? Thus VR systems significantly reduce the need for expensive and time-consuming physical prototyping.

Several virtual assembly environments have been developed to simulate the assembly process. In these environments the user can simulate industrial assembly scenarios. The quality of immersive experience in VR environments depends heavily on visual and haptic feedback. Most of the developed systems use haptic devices that enable manual interaction with the virtual environments. These systems implement proprietary 3D viewers capable of rendering the virtual environment in stereo for visual feedback. One of the drawbacks of these systems though is their disconnect with the CAD systems that are used for modeling the assembly components. The geometry of virtual objects in these systems is typically imported from the CAD systems in the form of polygonal data. Some of the existing systems such as the Virtual Assembly Design Environment (VADE) developed at the Virtual Reality Computer Integrated Manufacturing (VRCIM) Lab at Washington State University do provide limited CAD integration. While performing the assembly operation in VADE the user can make design modifications to the parts by accessing pre-selected CAD parameters in the VR environment.

In this research, we have attempted to overcome the following limitations of existing virtual reality (VR) systems for assembly operations:

1. User has to leave the familiar CAD environment that he is used to and work in a new UR environment.

2. Most of the Virtual assembly systems do not provide any functionality for 2-way integration with CAD.
3. Some Virtual assembly systems, which provide loose integration with CAD systems, require model preprocessing to tag the parameters that are made available in the VR environment.

This thesis focuses on developing an open architecture for embedding VR based mechanical tools in CAD system. Some of the primary goals set forth for this research were as follows:

1. Allow the implementation of virtual tools capability in the CAD system environment.
2. Propose an architecture for interfacing haptic devices directly with the CAD system.
3. Support functionality for building a library of mechanical tools, fasteners etc.
4. Implement the proposed architecture using CATIA V5 as a CAD system and VirtualHand Toolkit as the haptics API.
5. Support screw joint based mechanical fasteners such as bolts, screws, allen, bolts etc. and compatible tools.

In the next chapter we take a detailed look at the previous research done in the field of virtual assembly environments and clearly define the goals for this research.

# CHAPTER 2

# LITERATURE SURVEY AND PROBLEM STATEMENT

Several assembly design and planning applications have been developed by various researchers to address a variety of design and manufacturing problems. Researchers have identified assembly representation, design for assembly, assembly/disassembly operation sequencing, motion planning, and tool accessibility as major areas in assembly modeling and simulation. Many researchers have also developed useful haptic and force feedback devices and applications [1]. In this section, we present a brief summary of work done in these areas, in particular as it relates to virtual assembly.

Zachman et al. [2] describe a VR-based method of simulating complex assemblies for the automotive environment. They utilize menu and voice driven commands to constrain interactive object motion inside the VR system, Virtual Design II. For a screw movement, the tool axis is aligned to the screw axis only when collision occurs between the parts. A thresholding mechanism was created to switch off the snapping constraint when the user moves the tool a specified distance from the fastener. Sets of associations were used to match different tools with their respective screws. Less rigid mechanisms were also created to allow coincident sliding motion between parts during insertion movements in confined spaces (such as a car door) if no force feedback is available. This mechanism uses a copy of the object, or "ghost", which allows it to penetrate other parts while still being rigidly constrained to the hand.

Constantinescu et al. [3] have developed a local model of rigid body interaction that provides haptic feedback while manipulating a virtual tool within a virtual environment.

Their controller allows a user to feel physical phenomenon with the rigid environment such as collision and friction. This method allows the user to perceive tightly confined spaces and manipulate stiffer objects. They state one of the limitations as being that the environment does not allow dynamic manipulation of other parts other than the single tool.

Regenbrecht et al. [4] describe a method of utilizing tactile force feedback (TFB), in place of more costly force feedback systems, to assist the user in assembly processes. TFB associates the sensation of touch as perceived by stimulation on the surface of the skin. The stimulation is provided using separate vibro-actuators for three degree-of-freedom response. Their device, TACTOOL, was tested using tasks such as placing a generator in an engine as well as placing a battery in an engine cavity and proved beneficial in aiding collision feedback.

Coute et al. [5] and McDermott et al. [6] present Haptic Integrated Dis/Re-assembly Analysis (HIDRA), a haptically enabled assembly/disassembly simulation environment. This work is primarily aimed at providing haptic interface into a disassembly simulation environment. The haptic, graphic, and collision detection representations for the objects are created from the geometry obtained from the CAD system. They describe a haptic feedback and a graphic feedback loop that lets real-time, interactive haptic simulation of complex mechanical operations. The aim is to ultimately incorporate a set of virtual tools (screwdrivers, wrenches, pliers, etc.) for the user. This work has been extended [7] to characterize perception of weight in the virtual environment. The experiments are designed in such a way that the user can judge the weight of the model in virtual reality with the help of haptic feedback devices and provide an experience very similar to the weights in real environment.

Zhu et al. propose an approach for grasp identification and multi-finger haptic feedback to provide a realistic force sensation while performing the virtual assembly tasks [8]. A Voxmap-PointShell (VPS) algorithm is used to detect collisions and the result of this collision detection is used to guide the motion of the virtual hand.

Liu et al. discuss how haptic devices can be used in the design and deformation of complex 3-D models [9]. The haptic devices help in touching a native B-rep CAD model and using the tactile senses to manipulate it. Wan et al. present MVIAS, a Multi-modal Immersive Virtual Assembly System, which provides the user with an intuitive and natural way of assembly evaluation and planning [10]. The user interaction includes realistic virtual hand interaction, force feedback and real time display of complex assemblies.

Several key algorithms have been proposed to handle various aspects of geometry computations required in haptics. For example, Johnson et al. propose a generalized method to compute the minimum distance between two models in a virtual scene, which is a fundamental operation in simulation, haptics, and path planning [11]. Research has been done in Iowa state university to couple a CAD model to the analysis model in a 3-D environment to study the stress distribution within the product because of the shape changes [12]. Haptic devices have been used in this research to provide additional information related to feasibility of design and the impact of shape changes on the actual assembly. VR system and integrated haptic devices have also been used for the testing of aircraft engines [13]. This helps to reduce the cost and avoid the necessity of physical mock-ups for maintainability. Research for creating tools for cable harness design in virtual environment has been done [14, 15]. An industrial case study has also been put forth consisting of Assembly Process Planning using Immersive Virtual Reality [16].

Gupta et al. [17] have developed high fidelity assembly simulations and visualization tools that can detect assembly related problems without going through physical mock-ups. These tools can also be used to create easy to visualize instructions for performing assembly and service operations.

Gomes et al. [18] in Germany have developed a VR based application used for verification of assembly and maintenance processes. They investigate steps needed to apply virtual reality for virtual prototyping to verify assembly and maintenance processes. The authors have developed a three-layer framework to meet strategic and operative objectives. The three layers are the scene graph layer, the script layer, and the application layer. The scene graph layer is the CAD interface to the virtual environment, while the scripting layer drives the events. The application layer provides specific user interfaces depending on the user intended application domain. Assembly tools have been incorporated in the environment. Tools snap onto screws and are constrained, based on the snapping paradigms. The snapped screw is made to follow a 1-DOF rotational constraint that is triggered by events.

McDermott et al. [19] present Haptic Integrated Dis/Re-assembly Analysis (HIDRA), a haptically enabled assembly/disassembly simulation environment. This work is primarily aimed at providing haptic interface into a disassembly simulation environment. The haptic, graphic, and collision detection representations for the objects are created from the geometry obtained from the CAD system. They describe a haptic feedback and a graphic feedback loop that lets real-time, interactive haptic simulation of complex mechanical operations. Currently the users can only manipulate and feel the parts with virtual fingers. The authors describe that the aim is to ultimately incorporate a set of virtual tools (screwdrivers, wrenches, pliers, etc.) for the user.

Significant work has been done at Washington state university for the last 12 years in Virtual Assembly design environment "VADE", including [20-22]. Interested readers will find an introduction to the basic concepts of virtual assembly and the approaches to constraint management, kinematics, etc. in these papers. VADE has been tested and used for various industrial case studies [23]. Ergonomic software has also been integrated with VADE for evaluating the ergonomic issues of the assembly process [24, 25]. One of the ergonomic features called RULA has been integrated with VADE for the assessment of right upper limb [26]. A tool-hand interaction model for assembly in virtual environments has been created [27]. Another project that was completed created a haptic device for weight sensation [28].

## *Problem Statement*

As seen in the previous section, several current VR systems provide environments for simulating the mechanical assembly process. Most of these systems do not support integration with the CAD systems that are used to model the assemblies. They use neutral data formats to extract the geometry information out of the CAD systems as tessellated models.

Virtual Assembly Design Environment (VADE) developed at the Virtual Reality Computer Integrated Manufacturing (VRCIM) Lab at Washington State University goes a step further and provides two-way integration with CAD. The user, though outside the CAD environment, can access certain CAD parameters and modify them in the VR environment. VADE is capable of communicating these changes back to the CAD system. The CAD system then updates the models based on the user input and writes out updated polygonal data for the assembly components.

The limitations of existing virtual assembly systems listed in the previous chapter provide the motivation for this work. The basic goal of this research is to propose and implement an architecture to support virtual assembly tools for simulating assembly/ disassembly operations in the CAD systems. Some of the key challenging aspects of this work will involve the following tasks:

1. Using the CAD visualization system for simulating the immersive virtual environment for assembly operations.
2. Extracting the assembly hierarchy and the geometry information from the CAD models.
3. Building and maintaining the haptic scene graph using the API provided by haptics device manufacturer.
4. Synchronizing the visual and the haptic scene graphs during simulation.
5. Managing the tool-fastener interaction through constraints.
6. Managing the states of various simulation components.
7. Providing functionality for building and maintaining tools and fasteners library.
8. Designing and implementing an open architecture that will allow these capabilities to be quickly adapted for other tools, CAD systems, and haptic devices.

In the next chapter we discuss in details a prototype implementation of this functionality. The prototype was implemented as a proof of concept to study the feasibility of the proposed research.

# CHAPTER 3

# PROTOTYPE IMPLEMENTATION

This chapter discusses a prototype implementation of the virtual assembly functionality using virtual tools (VR Tools) for CATIA V5 CAD system and VirtualHand™ toolkit haptics API. It was decided to do a preliminary feasibility study before developing a complete production version of the VR Tools software. The following section lists the goals for this phase.

## *Goals of Prototype*

- To demonstrate the virtual tools capability for assembly/ disassembly operations on a sample CATIA V5 assembly model consisting of mechanical fasteners.
- To support limited variety of mechanical fasteners such as bolts and allen screws and compatible tools.
- To use existing commercial software as much as possible and keep custom software development to a minimum.

## *Software and Hardware Systems*

Based on the requirements laid down by Sandia and available software and hardware modules, the following systems were chosen for performing the feasibility analysis.

### CATIA V5-R15 (CAD System)

Sandia chose CATIA V5 as the CAD system for developing the VR Tools technology. A sample model consisting of a multi-part platform assembly that used allen screws and hex bolts as fasteners was provided. Sample models for tools such as Box wrenches, Allen wrenches and Flathead screwdrivers were created in CATIA. The CATIA Kinematics workbench was used extensively for creating screw joints between the fasteners and other assembly components.

### VirtualHand® for V5 (VH4V5 -CATIA Workbench)

VH4V5 is a CATIA V5 workbench developed by Immersion Corporation (Manufacturer of the Cyber Glove, Cyber Grasp and Cyber Force haptic systems). This workbench enables the haptic devices to interface with CATIA and use it as a visualization system for the haptic scene graph. It enables the user to control a virtual hand inside CATIA thereby facilitating assembly manipulations. It is capable of extracting the graphical representation from the CATIA models for building the haptic scene graph. The user can also manipulate the objects in the scene graph by using the Virtual Hand through the Cyber Glove. It synchronizes the graphical representation in CATIA with the haptic scene graph for real time scene update during the simulation.

### CyberGlove®, CyberGrasp™ & CyberForce™(Haptic devices)

Various haptic devices were used for enabling the interaction between the user and the virtual assembly environment. The CyberGlove was used as a data acquisition device to capture the motion of a human hand using an array of strain gauges mounted on a glove. The CyberGrasp provides touch and grasping feedback thereby making the simulation more

realistic. The CyberForce provides force feedback enabling the user to experience the forces that they would normally experience from working with the real objects. The haptic devices used are all made by Immersion Corporation. Figure 3.1 shows the CyberGlove and the CyberGrasp.

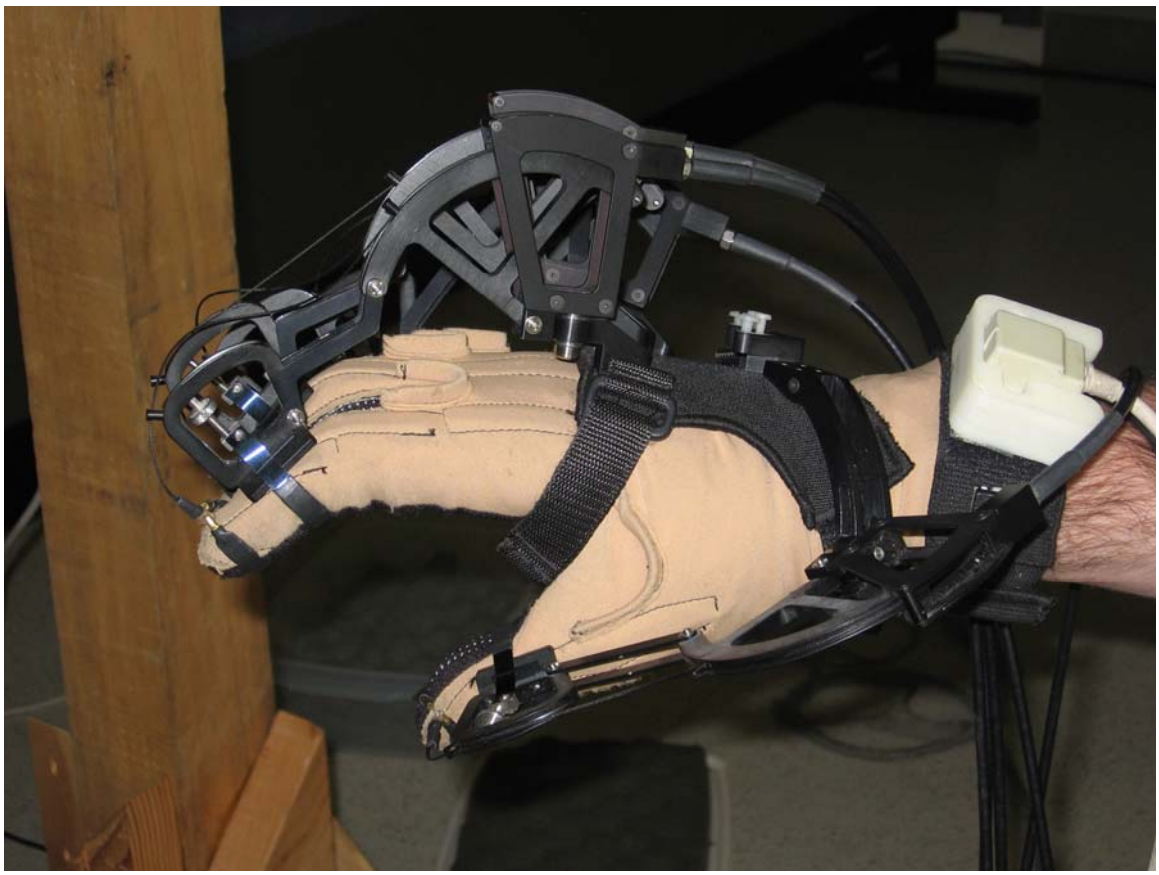

**Figure 3.1: CyberGlove and CyberGrasp**

## *VirtualHand for V5 evaluation*

VirtualHand for V5 supported some functionality that was required to demonstrate the VR Tools capability. However, as VirtualHand for V5 was not an open source system it was not possible to develop it further to achieve all the goals set forth for this research. As a result we decided to use VirtualHand for V5 and add external capabilities in order to provide a workable demonstration for implementing the prototype. Significant amount of investigation was done in order to understand the existing capabilities of VirtualHand for V5 and how they can be used in this project. Following are some of the key strengths and limitations that were discovered during this exercise.

### Strengths of VirtualHand for V5:

1. Capability to parse the CATIA assembly tree and build a haptic scene graph.
2. Capability to perform collision detection between the hand and other objects, thereby enabling grasping.
3. Capability to enforce assembly and kinematics constraints during the simulation.
4. Capability to provide grasp & force feedback during the assembly process.

### Limitations of VirtualHand for V5:

1. Collision detection is performed only between the hand and the components. There are no component-to-component collision checks.
2. Assembly or Kinematic constraints required to implement the tool-fastener interaction cannot be applied or modified dynamically during the simulation.

3. Gripping algorithm is not very realistic particularly for small components. It is very difficult to maintain a grip on small parts for manipulation. The fingers tend to stick to the parts and significant practice is required to become familiar with the process. Due to this problem, it was decided to scale the sample assembly models up to make them larger and easier to manipulate. Also, the tools were modeled with extended and thickened handles to allow easier gripping without obstruction from other parts.

The next section discusses various methodologies adopted to overcome the above-mentioned limitations while implementing the VR Tools prototype. The implementation went through quite a few changes based on the results of various approaches.

## *Implementation*

---

### Approach 1: Reorganize assembly hierarchy

It was observed that VirtualHand for V5 does its own constraint management and does not enforce any dynamically applied constraints during the manipulation process. As the user could take the wrench and align it with the bolt in 6 different possible locations, it was necessary that the constraints be applied dynamically. It was also observed that VirtualHand for V5 treated a sub-assembly as a single body and did not allow any relative motion between its components. Hence one of the very first approaches involved development of an algorithm for dynamically reorganizing the assembly hierarchy to simulate coordinated motion between the fastener and engaged tool. Therefore, it was required that the wrench be in the top-level assembly when it is not engaged with a bolt. In this state the user can move it freely and align it with the bolt. Once aligned it was important

that the wrench be constrained to the bolt and cause the bolt to rotate along with it. Hence it was necessary to dynamically change the assembly hierarchy in the CATIA model.

It was decided to use existing VirtualHand for V5 capabilities for interacting with CATIA to allow manipulation of assemblies, while a separate program would reorganize the assembly hierarchy tree inside CATIA. This reorganization method allowed us to manipulate a tool in engaged and disengaged state using VirtualHand for V5. In the engaged state, the assembly hierarchy would be changed to make the tool a child of the bolt assembly. This would facilitate concurrent rotation of the tool (wrench) and the bolt.

Figure 3.2 depicts two different assembly hierarchies. In state I the wrench is in the top-level assembly and can be grabbed and moved around. In state II the wrench is part of the bolt sub-assembly, effectively forming a rigid joint with the bolt.

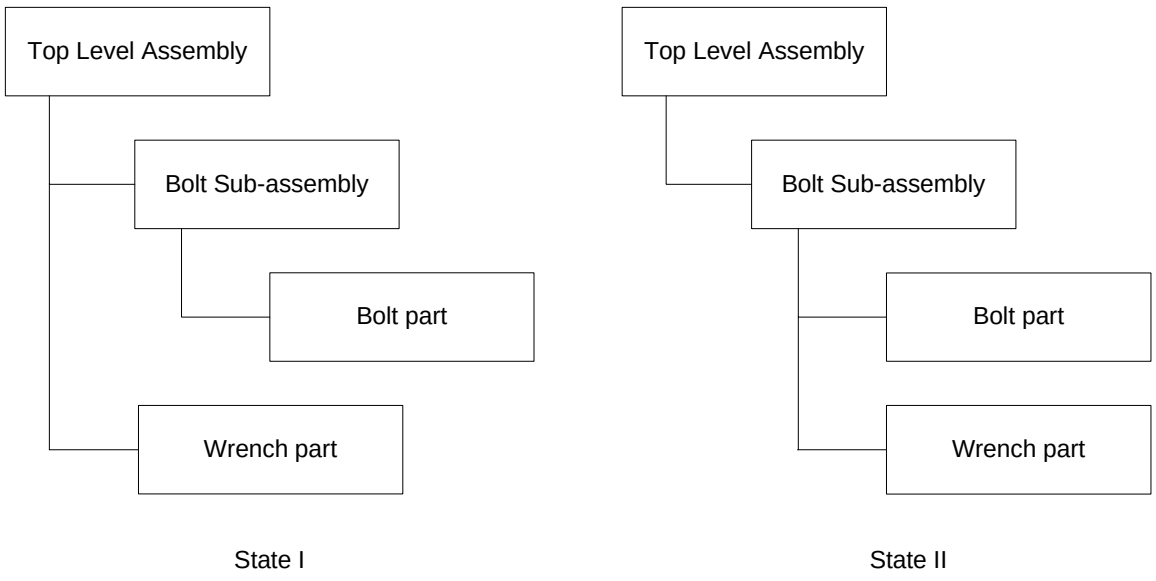


**Figure 3.2: Reorganize assembly hierarchy**

Implementation of the above discussed solution involved development of a CATIA Workbench "addin" for manipulating the assembly hierarchy. This addin was activated using

a simple toolbar button created inside the CATIA assembly workbench. The basic function of the toolbar button was to toggle the assembly structure between the two hierarchies depicted in Figure 3.2. The user could pick up a tool and bring it close to any bolt for simulation. At this point the user would have to interrupt the VirtualHand for V5 simulation loop by stopping the manipulation. The user would then initiate the assembly restructuring code by pressing the addin toolbar button.

***Constraints creation for tool placement***

The assembly-restructuring algorithm was designed to find the closest edge on the closet bolt and add the tool to the bolt sub-assembly. Assembly constraints were created so that the tool is correctly located with respect to the bolt. The user could then re-start the VirtualHand for V5 manipulation process. At this point VirtualHand for V5 reloaded the assembly and its scene graph with the updated assembly hierarchy and constraints. This enabled the user to rotate the tool for tightening or loosening the bolt. Pressing the addin button after stopping the simulation could then restructure the assembly hierarchy. The user could then disengage the tool by restarting the simulation. This sequence could be repeated as many times as necessary to fully tighten or loosen all of the active bolts in the assembly.

To achieve the creation of dynamic constraints a set of datum planes and datum points was needed for reference. A hex bolt has six faces, so opposite faces were named FACE A1 and A2, B1 and B2, and C1 and C2. Planes and a datum point were also created tangent to each bolt edge. Finally a datum plane was created on the top surface of the bolt for height alignment with the wrench. These planes were used as references for the constraints, enabling a rigid connection between the bolt and wrench.

Due to the similarities between a hex and allen screws it was possible to use the same algorithm for both types of fasteners. Extending the functionality to allen screws just

involved creation of datum planes similar to the ones used on the hex screws. An allen wrench was modeled in CATIA based on the allen screw dimensions. Datum planes were created and named according to the convention used for the hex wrench. This expanded the capabilities of the software to allow the use of two separate tools with the limitation that only one tool can be used at a time.

### *Limitations*

Though the approach discussed above provided a workable solution, it failed to provide a smooth demonstration in a manner envisioned by the researchers. The assembly restructuring was necessitated due to the fact that VirtualHand for V5 does its own constraint management and does not solve dynamically created constraints. The approach involved manual steps (button clicks) for triggering the assembly hierarchy restructuring. The user intervention required in this approach was undesirable but also unavoidable. Due to these limitations this approach was found unsuitable for the final demonstration unless some of the manual steps were automated.

## Approach 2: State management using Callbacks

Approach 1 sufficiently demonstrated the VR Tools concept and capability but the simulation was not smooth and involved lot of user intervention. Improvements in the quality of the simulation required new functionality to be implemented in VirtualHand for V5. Immersion enhanced VirtualHand for V5 to provide a callback mechanism that would enable an external application to effect desired changes during the simulation. The callback function was designed such that it would get called on each frame of the simulation and would provide the grasped object information. The callback returned the name and current transformation of the grasped object. Knowing the name and position of the grasped object enabled the program to recognize the grasped tool. This made it possible to use both the allen wrench and the hex wrench together. To address the issue of user intervention and manual button clicks, another software module was implemented using Spy++[29]. Spy++ is a tool provided with Visual Studio® that can be used to understand the window layout of a running application. It helps in finding specific controls on a window and activate them programmatically. This module made it possible to click the Start and Stop manipulation buttons programmatically thereby reducing user intervention during the simulation.

### *Stage-State management for simulation*

The callback support provided during the simulation was limited and would cause the application to crash if not used carefully. It was not possible to perform more than one action on any given frame. This made it necessary to design the tasks in such a fashion that they would be carried out over a span of multiple frames. It was therefore necessary to have an elaborate state management scheme to be able to run complex tasks during the simulation.

To begin with, the following three independent stages of the simulation were identified.

1) EmptyHand stage.

2) ToolInHand stage.

3) RotatingBolt stage.

During the simulation it was possible to go from one stage to any of the other 2 stages. Each stage had multiple states associated with it. The combination of the current stage and its state completely described the state of the simulation thereby enabling the state handlers to take correct steps. Each stage was assigned a state handler for doing its state management. Table 3.1 shows different stages and their state handlers.

| Stage Number | Stage Name | State Handler |
|---|---|---|
| 0 | EmptyHand | WSUCallback() |
| 1 | ToolInHand | ToolInHandHandler() |
| 2 | RotatingBolt | RotatingBoltHandler() |

**Table 3.1: Simulation Stages**

Since it was only possible to perform one operation on every callback, a stage could only make 1 state transition each frame. Following is a brief description of each stage. Tables 3.2 and 3.3 list all the possible states for stages 2 and 3 along with the actions performed by the state handlers during each state.

**Stage 1: EmptyHand Stage**

During this stage the state handler would keep checking if the tool is grasped. Once the tool is grasped the stage is set to 1 and the state is initialized to 0.

**Stage 2: ToolInHand Stage**

During this stage the state handler would keep checking if the tool is close enough to a compatible fastener to be engaged. Once the tool is close the states will increment from 1 through 4 and then the stage will be set to 2.

| State No. | State Name | Action |
|---|---|---|
| 0 | Start State | If the wrench is close to the bolt, go to State 1. |
| 1 | StopManipulation | Stop simulation and set the next state to 2. |
| 2 | SaveFile1 | Save the model and set the next state to 3. |
| 3 | ApplyConstraints | Apply constraints and set next state to 4. |
| 4 | StartManipulation | Start simulation and set the stage to 3 (RotatingBolt) and the state to 0 (StartState). |

**Table 3.2: ToolInHand States**

**Stage 3: RotatingBolt Stage**

This stage is initialized to state 0. The user can loosen of tighten the fastener in this state. This stage can go to various different states from state 0 based on the event caught by

the state handler. For example, when the user rotates the bolt using wrench and the bolt comes out of the hole the state handler will return the simulation to stage 1.

| State No. | State Name | Action |
|---|---|---|
| 0 | StartState | Check if the distance between the bolt and the tool is larger than a preset tolerance, if it is, set next state to 1 followed by state 2,<br>Check if the bolt is out of the hole, if it is, set next state to 1 followed by state 4, If the tool is released set stage to EmptyHand |
| 1 | StopManipulation | Stop simulation and set the next state to 2. |
| 2 | SaveFile1 | Save the model and set the next state to 3. |
| 3 | RemoveConstraintsForTool | Remove constraints on tool and set the next state to 6. |
| 4 | SaveFile2 | Save the model and set the next state to 5. |
| 5 | RemoveConstraintsForBoth | Remove constraints on tool and the fastener, then set the next state to 6. |
| 6 | StartManipulation | Start simulation and set stage to EmptyHand and state to 0. |

**Table 3.3: RotatingBolt States**

The following figures (3.3-3.5) depict the flow charts for various stages. They show logic used for making transitions between stages and between states of the stages. They show how the state handlers use the current stages and states to decide upon the next stage and/or state.

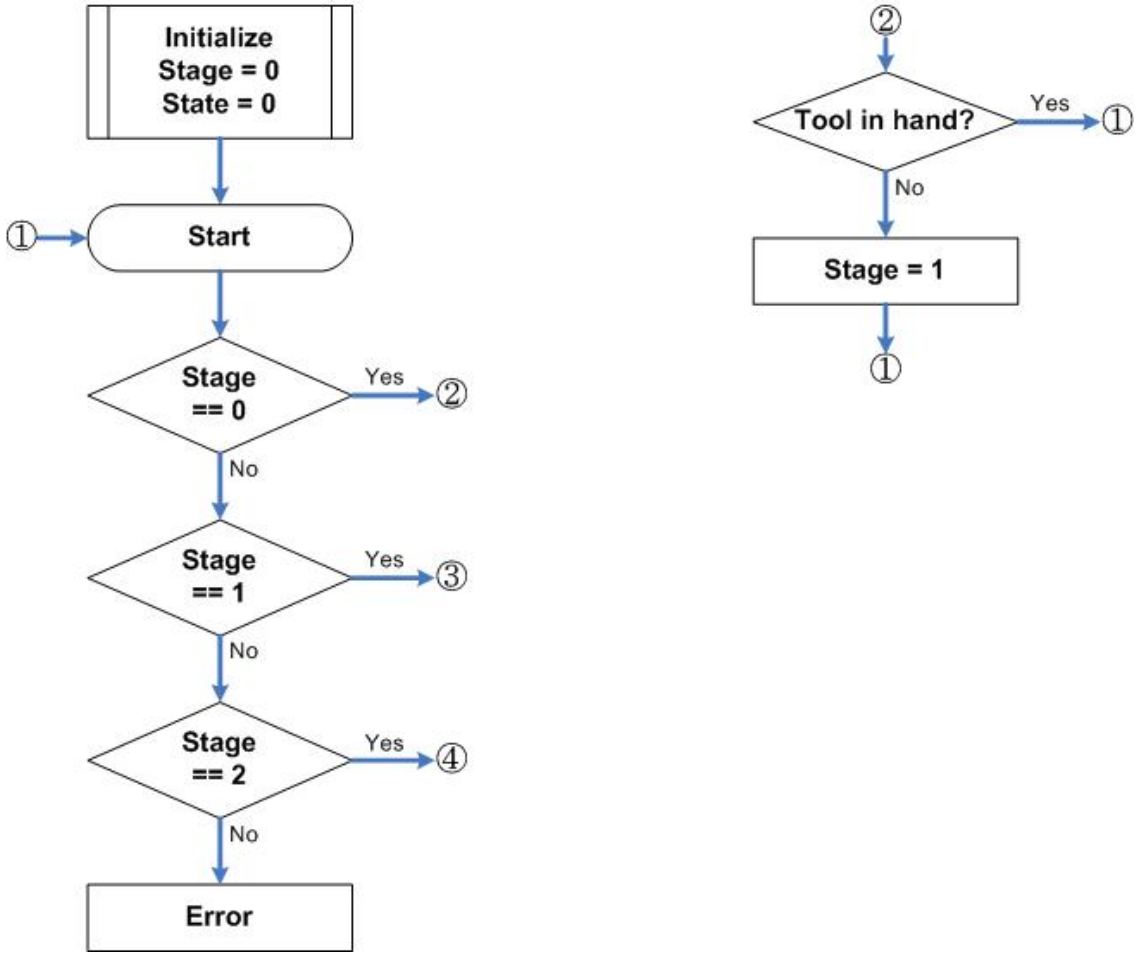


**Figure 3.3: High level simulation stage management flowchart**

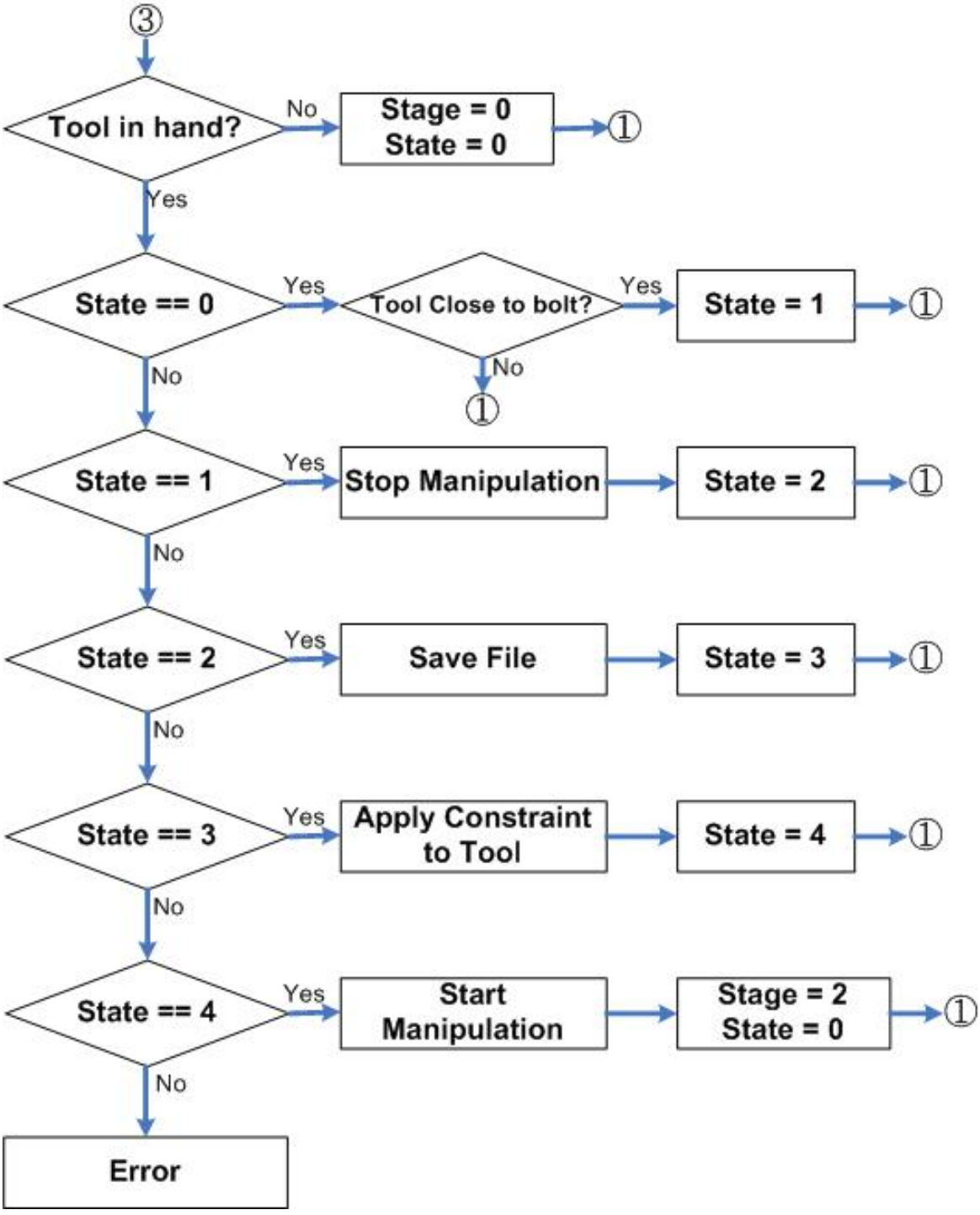


**Figure 3.4: ToolInHand stage state management flowchart**

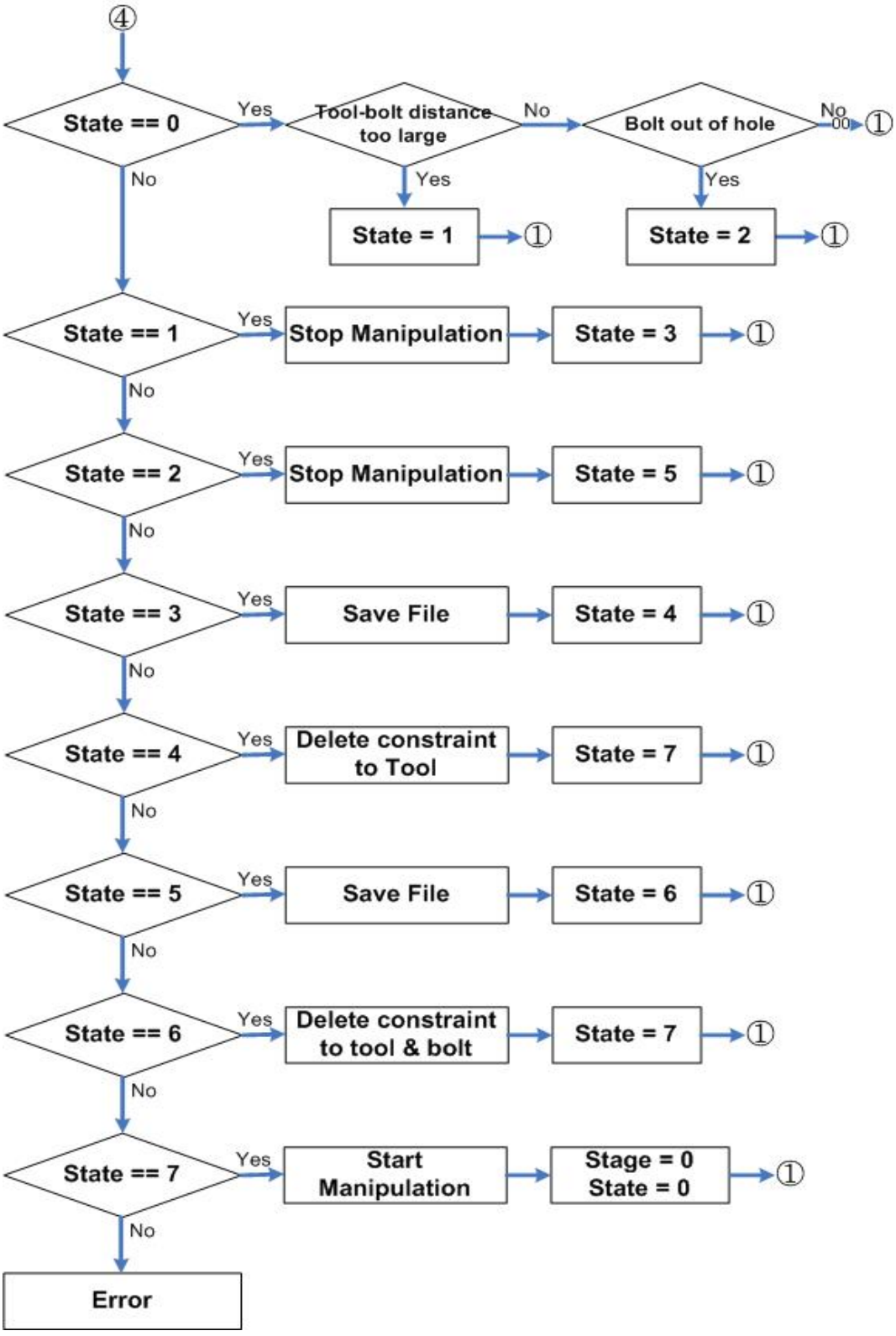


**Figure 3.5: RotatingBolt stage state management flowchart**

### Approach 3: Constraint management using Callbacks

Though Approach 2 significantly reduced user intervention and automated the simulation, speed of the simulation was not up to the mark. The process of stopping and restarting the simulation for changing the assembly hierarchy was quite time consuming. The whole assembly restructuring was being done because of the fact that VirtualHand for V5 was not capable of creating constraints on the fly. After considering several possibilities for overcoming this limitation, Immersion Corporation provided a newer version of VirtualHand for V5. This version was capable of activating and deactivating existing constraints during the simulation. So the algorithm was changed to pass in a list of active constraints on every frame based on the location of the tool with respect to the fastener.

A model was created with all possible constraints (108 assembly constraints for the sample model) for all possible positions of an engaged tool. When the user gripped the tool and tried to engage it with the bolt, based on the relative positioning of the tool and the bolt, a set of applicable constraints was identified and passed to VirtualHand for V5. This allowed the user to dynamically snap the tool on to the fastener in any desired position. This eliminated the need for stopping and restarting the simulation and made the overall experience much smoother.

## *Demonstration*

Sandia identified a sample assembly for demonstrating the VR Tools proof of concept. A suite of CATIA models, which included box wrenches, allen wrenches and compatible fasteners was created. The fastener sizes were picked to be compatible with the sample assembly. When all the components were ready they were all assembled together.

CATIA Kinematics module was used to create screw joint constraints between the screws and their respective holes. It was observed that the constraint solver in VirtualHand for V5 would enforce the screw joint but the pitch was inversely proportional to the value set while defying the constraint. So smaller the pitch value, faster the screw would travel in or out of the hole.

It was also noticed during the preliminary stages of testing that the Sandia model was too small and it was hard to grab the assembly components using the VirtualHand™. It was therefore decided to scale up all the assembly components and the tools and fasteners by a factor of 4. This gave the user a better control over grabbing the components in the virtual environment.

Datum planes and points were created on all the tools and fasteners and were used for creating the constraints to be used during the simulation. The constraints followed a specific naming convention thereby enabling programmatic manipulation of constraint states. It was also observed that due to the constraint solver limitations it was necessary to over constrain a part to be able to position it in a correct manner consistently. Figures 3.6 through 3.11 show various stages during the simulation.

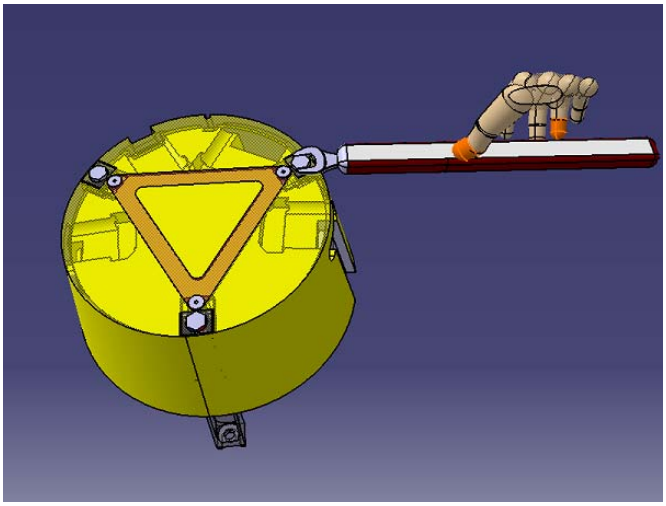

**Figure 3.6: Stage 2 ToolInHand**

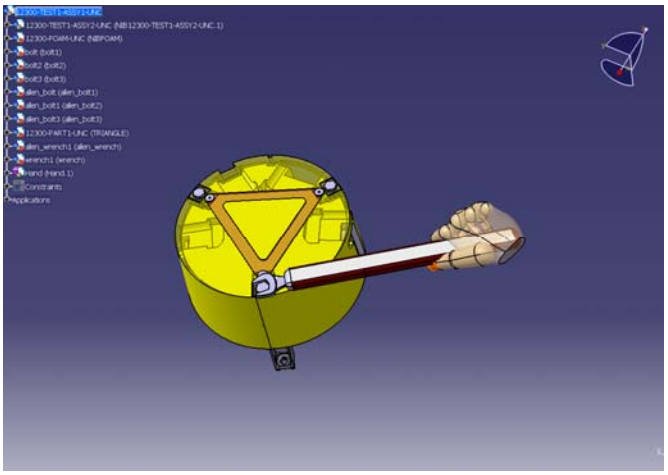

**Figure 3.7: Stage 3 RotatingBolt**

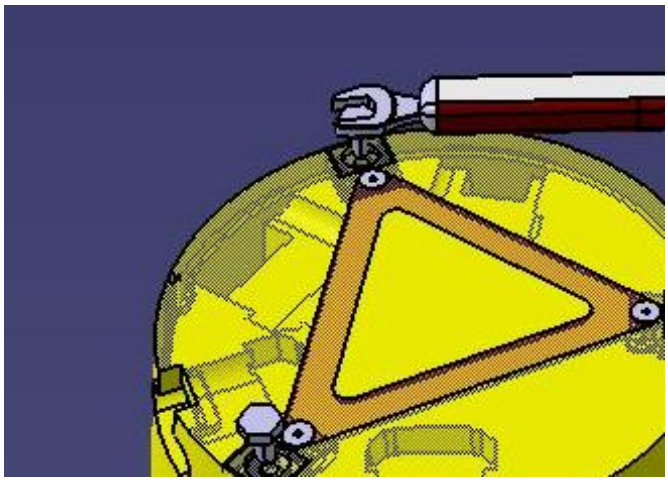

**Figure 3.8: Stage 3 – state 4 Bolt out of hole**

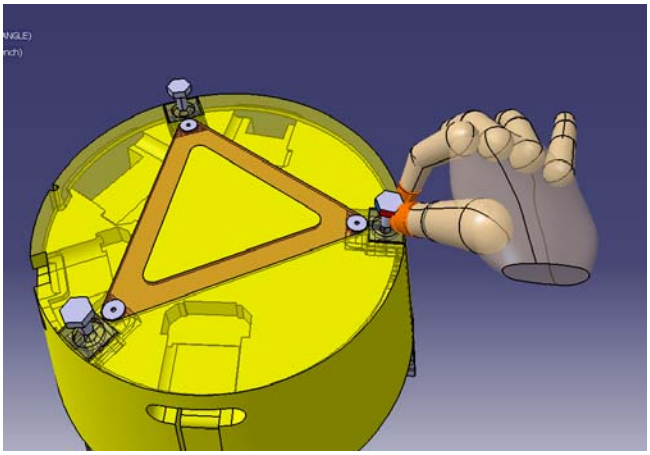

**Figure 3.9: Unscrewed bolt being removed from the assembly**

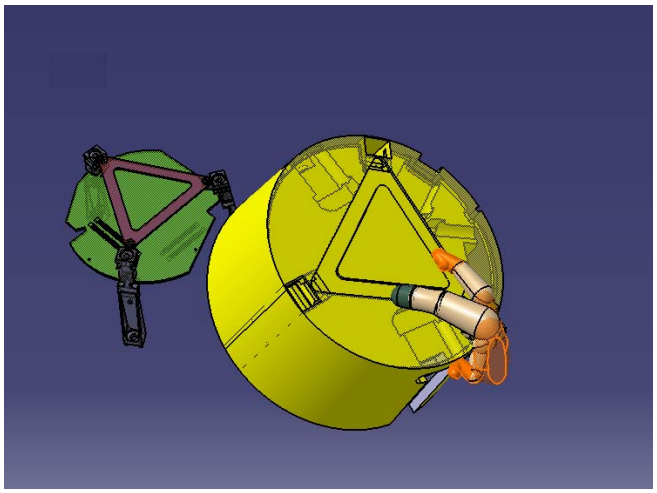

**Figure 3.10: Disassembly of a component**

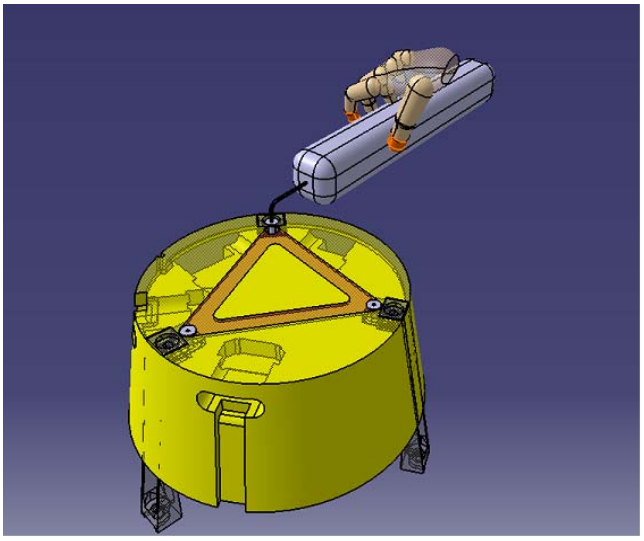

**Figure 3.11: Allen wrench demonstration**

Thus we were successfully able to demonstrate the proposed virtual assembly tools functionality through the prototype implementation of the VR Tools software. Subsequent chapters discuss in detail the proposed open architecture for VR Tools capability and its implementation for CATIA V5.

# CHAPTER 4

# PROPOSED SOLUTION

The proof of concept implementation discussed in chapter 3 gave a deep insight into the requirements of the proposed system. Due to all the limitations faced while working with VirtualHand for V5 and due to the fact that it was not an open source system it was decided to re-develop the core functionality. A completely new system based on an extensible architecture was envisioned. This chapter discusses the proposed solution for integrating VR (Haptic) devices with CAD for providing the virtual tools functionality. Figure 4.1 shows a high-level schematic for the proposed solution.

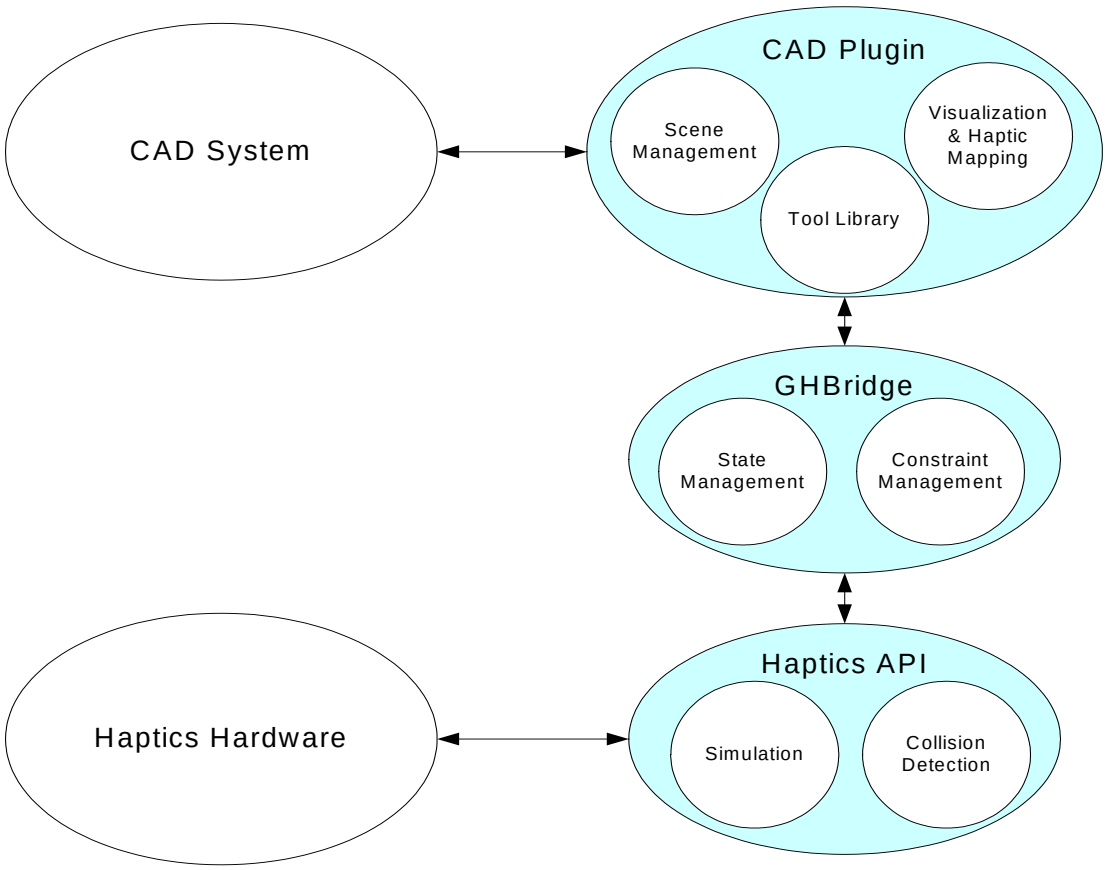


**Figure 4.1: Proposed Solution**

## *VRTools Components*

The proposed software (VR Tools) comprises of the following building blocks:

1. CAD system: Used for modeling and visualization.
2. Haptic Devices: For capturing human interaction and providing force-feed back.
3. CAD plugin: For interfacing with the CAD system.
4. *GHBridge* (Graphics-Haptics Bridge): For interfacing with the Haptic devices.
5. Tools/Fasteners Library: For managing the fasteners and compatible tools.

The following section provides an overview of the role of each component in VR Tools.

### CAD System

A Mechanical-CAD (M-CAD) system is a basic software tool that is used for modeling the parts and assemblies of real mechanisms that can be used in a virtual environment for simulating mechanical tool operations. CAD systems help build, manage and store all the geometrical information for a mechanism and also provide visualization capabilities. Models are built by using feature based modeling techniques and managed using PDM systems. Some CAD systems also provide library or archiving capabilities for organizing related models in a desired manner. CAD systems facilitate data storage in various different formats there by making the geometric information easy to share between various applications. ProEngineer from *Parametric Technologies*, Unigraphics from UG, CATIA V5 and SolidWorks from *Dassault Systems* are some of the popular M-CAD systems.

### Haptic Devices

Haptic means pertaining to the sense of touch. These are hardware devices that enable human interaction with the VR environments. They act as input as well as output devices. They capture inputs in terms of motion, speech etc. and provide feedback via touch, force, vibration etc. There are many kinds of haptic devices. Immersion Corporation's CyberForce, CyberGlove and CyberGrasp are some of the devices used in this research.

### CAD Plugin

This is a custom software module whose architecture is proposed to enable interaction with the CAD system through haptic devices. This module is responsible for extracting all the geometry and kinematics information from the CAD system for building the haptic scene graph for the collision engine. It provides the user with a graphical interface for driving the simulation. This module also handles and updates the graphics for the CAD system during the simulation. The implementation and deployment scheme depend on the specific CAD system that is being used.

### GHBridge

GHBridge (Graphics-Haptics Bridge), as the name suggests, is the link between the CAD system graphics and the haptic devices. The GHBridge builds on top of the haptics API and implements the haptic scene graph along with all the custom algorithms for manipulating it. It uses the geometry information extracted by the CAD plugin to build the haptic scene graph in the setup phase. During entry frame of the simulation, it extracts information from the haptic scene graph and drives the visual scene graph using the CAD plugin.

### Tools/Fasteners Library

The VRToolsCatalog acts as a library of fasteners and compatible tools. It facilitates classification of the models into various categories. It is also responsible for maintaining the link to the CAD model as well as storing vital meta-data used by the VR Tools program. The library is completely extensible and provides the user with tools for adding more fasteners and tools to be used with the VR Tools functionality. A set of modeling guidelines needs to be laid out to ensure that the new models are VR Tools compliant.

## *Simulation*

All the VR Tools components have specific roles to play during the simulation. The simulation can be thought of as a two-phase process comprising of a setup phase and a runtime phase.

### Setup Phase

In the proposed solution, VR Tools goes through a setup phase before the simulation is run. The following tasks are performed during the setup phase.

#### *Component-Fastener mapping*

This is an interactive phase. The CAD plug-in provides a GUI based utility for defining the mapping between assembly components and the fasteners of the model. This mapping enables the state managers of the components determine the state of the components during the simulation. For instance, state of an assembly component depends on the state(s) of the fastener(s) mapped to it (e.g. free, constrained by bolt, etc.)

***Preprocessing the tools library***

During this operation the program identifies the fasteners used in the model. Once the list of used fasteners is formed, it is used to identify the compatible tools based on their type and designation. The tools library is reorganized to separate out the compatible tools as a result of this operation.

***Selecting the Tool***

An interactive phase in which the user selects tools from the list of compatible tools in the library. The selected tools are added to the current model and are made available to be used during the simulation.

***Scaling the hand model***

Based on the size of the model the user can specify a scaling factor that is used to scale the virtual hand model. This enables the user to work with assemblies of different sizes and still be able to manipulate assembly components comfortably. This functionality is particularly useful while dealing with very small components, which are rather hard to grasp if the hand model is too big.

***Extracting assembly hierarchy***

The simulation engine uses the haptic scene graph maintained by the GHBridge during the simulation. The scene graph replicates the assembly hierarchy of the CAD model and keeps the transformations for all the components up to date. During this phase the assembly tree structure for the CAD model is parsed for extracting the hierarchy and transformation information. This information is eventually used for building the haptic scene graph.

***Extracting geometry information***

For building the haptic scene graph the geometric representation of each component is required. During this phase the program visits each assembly component and extracts vertices from its topological representation. The point cloud is used for constructing a convex hull representing the haptic node for each component.

***Extracting the graphical representation***

The CAD system internally maintains the graphical representation of every assembly component, which is used for displaying it. During this phase, the assembly tree structure is parsed in order to obtain the handles for all the graphical representations. This enables VR Tools program to update the visual scene graph of the CAD system during the simulation.

***Building the haptic scene graph***

During this phase the GHBridge builds the haptic scene graph using the information obtained during the previous stages. It also builds a list of tools and fasteners, which are later, used while running the simulation. The fasteners are matched with compatible tools and constraints are mapped. This information is used during the simulation while engaging the tool with the fastener.

***Registering scene update callbacks***

The CAD plugin registers a couple of callbacks with the GHBridge. These callbacks enable the simulation loop to update the scene by sending across the hand transformations along with the updated scene graph information. The callback functions get called every frame.

***Initializing the haptic devices and collision engine***

During this phase the GHBridge initializes the haptic devices selected by the user. It also initializes the collision engine.

## Runtime Phase

During runtime the GHBridge spawns a separate simulation thread that performs the following tasks during each frame:

### *Updating the hand transformations*

The GHBridge obtains the transformations for all the components of the virtual hand from the haptic device. These transformations are then sent to the CAD plugin using the callback function for the hand update. The CAD plugin then updates the transformations for the graphical representation of the hand thereby keeping the scene updated.

### *Checking for gripping*

During each frame, the GHBridge checks if any of the objects in the haptic scene graph is grasped. Collision detection and gripping algorithms are employed to detect a state change for a component that is being grasped.

### *Checking for constraints*

If an object is grasped, this task monitors the status of all the constraints associated with the grasped object. This mechanism provides the basis for doing the state management of a component. If all constraints are met, an assembly component can go from disassembled to assembled or vice versa. A grasped tool may go from 'disengaged' to 'engaged' state when it is in the vicinity of a compatible fastener. We will take a closer look at the constraint and state management in subsequent chapters.

### *Enforcing constraints*

When a grasped object is moved, this task ensures that the active constraints for that object are enforced. This may result in motion of other components or may cause the

grasped component to have limited degrees of freedom based on the types of constraints being enforced.

***Refreshing the scene using the scene update callback***

During each frame the haptic scene graph is updated due to the combination of user interaction, constraint management and state changes. These changes are communicated back to the CAD plugin via a scene update callback. This callback is then responsible for synchronizing the visual scene graph with the haptic scene graph.

Thus the proposed solution provides a basis for an open architecture for implementing virtual assembly tools functionality within any CAD a system. Various modules proposed in figure 4.1 facilitate loose coupling between the CAD system and the haptic devices thereby making it possible to use different combinations of CAD systems and haptic devices for implementing VR Tools. In the next chapter we discuss in detail the architecture of various VR Tools components proposed in this chapter. We take a look at some of the key classes in each software module and delve into the details of state and constraint management schemes.

# CHAPTER 5

# OPEN ARCHITECTURE

This chapter discusses the proposed architecture for integrating VR devices with CAD systems. It gives an overview of interaction between all the different components discussed in chapter 4. It also lays out architectures for the key software modules of the proposed solution namely, CAD Plugin, GHBridge and the Tools/Fasteners Library. Figure 5.1 depicts all the high-level components and their relationships with each other. The CAD plugin and the GHBridge constitute the two main software modules implemented by VRTools. They are responsible for integrating all the other components.

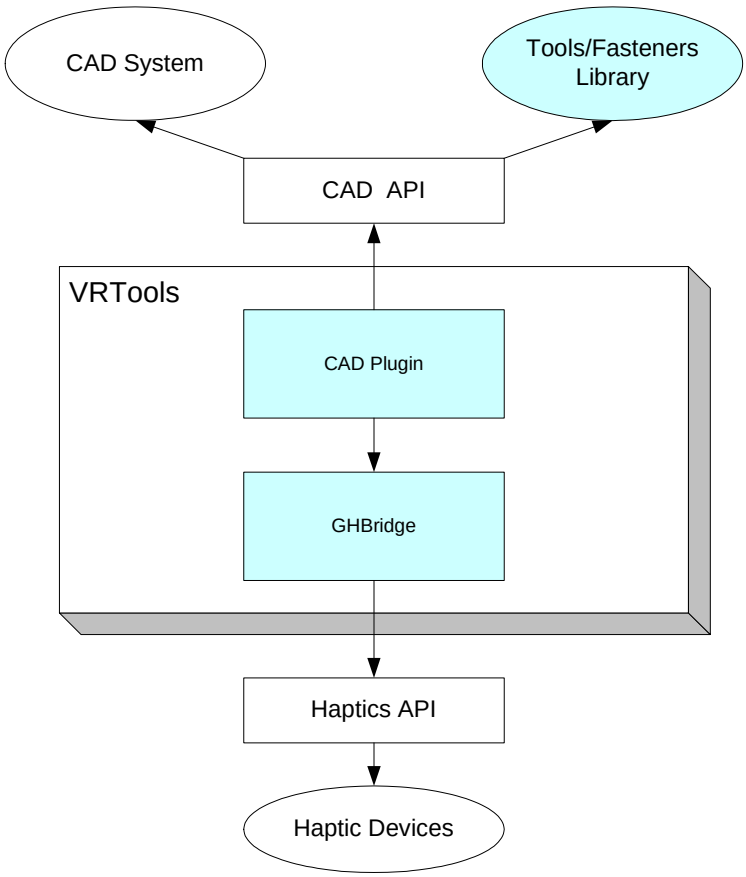


Figure 5.1: Overall Architecture

The CAD Plugin acts as an interface to the CAD system and implements all the CAD API code. The GHBridge acts as an interface to the haptic devices and implements the API code provided by the haptic devices. During the simulation phase the CAD plugin and the GHBridge work together and keep the CAD system graphics and the state of the haptic scene graph synchronized. The GHBridge captures any changes to the scene graph using the haptics API and conveys it to the CAD plugin, which in turn updates the CAD graphics.

## *CAD Plugin Architecture*

The CAD plugin is responsible for the following tasks in the VR Tools functionality:

1. Act as an interface to the CAD system.
2. Extract model tree data, assembly hierarchy and geometry information from the CAD model.
3. Extract and maintain the graphical representation for the CAD model.
4. Extract the constraint data from the CAD model.
5. Support the Tools/Fastener Library functionality either within the CAD system or externally.
6. Provide a graphical user interface for setting up and running the simulation.
7. Provide a utility for defining Component-Fastener mapping and storing that information with the CAD model.

Figure 5.2 shows a class diagram representing some of the basic classes proposed in this architecture. The detailed structure, functionality, implementation and deployment scheme may vary based on the CAD system being used. Chapter 6 discusses in detail how this architecture is applied to the implemented solution using CATIA V5 as a CAD system.

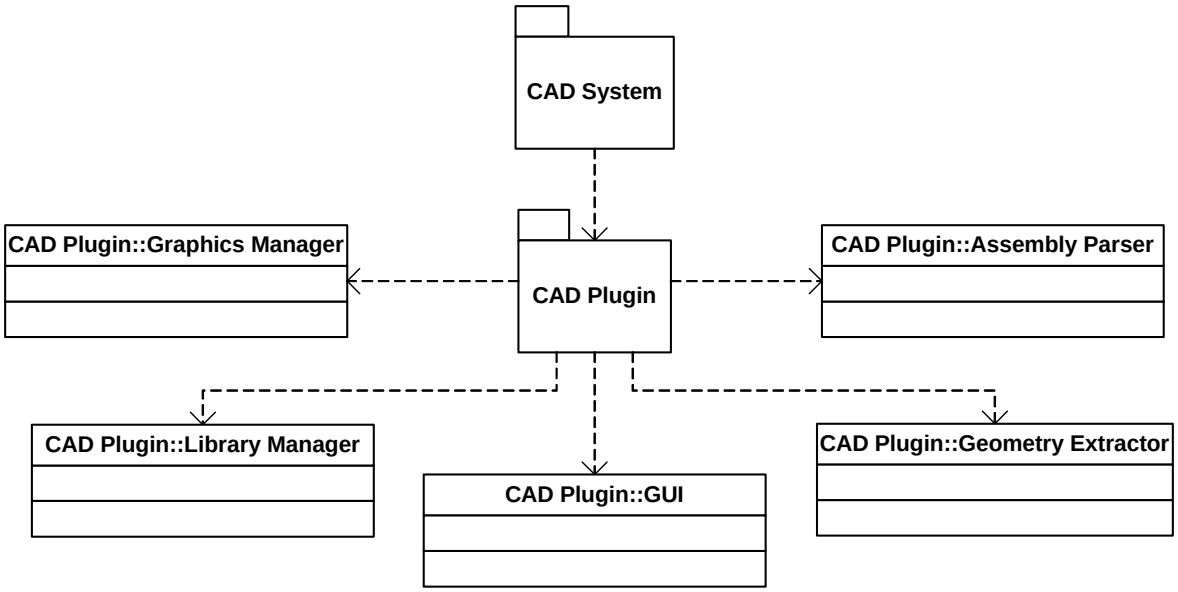


**Figure 5.2: CAD Plugin Architecture**

The CAD Plugin::GUI provides all the functionality for user interaction during the setup phase. It facilitates the component-fastener mapping and also allows the user to choose correct set of haptic devices to be used for the simulation. It can be extended to get other kind of inputs from the user for setting up the simulation.

The GUI also has all the logic for starting the simulation. It uses the 'Geometry Extractor' class for getting all the geometric and topological data needed for building individual component nodes of the haptic scene graph. The 'Assembly Parser' class is used to extract the assembly hierarchy along with the transformation information for each assembly component recursively. This information is used by the GHBridge for building the haptic scene graph.

The GUI also acts as a link between the CAD Plugin and the GHBridge. It provides all the information needed for building the scene graph. It also registers the graphics-update callback functions with the GHBridge. The callback functions enable the synchronization between the visual and the haptic scene graphs. The 'Graphics Manager' maintains the

graphical representation of the CAD model and has the functionality to update it every frame based on the information received from the GHBridge.

The CAD Plugin also provides access to the 'Tools/Fasteners Library' through the Library Manager. It provides functionality for finding appropriate tools based on the fasteners used in the CAD model. The implementation details for the library depend heavily on the CAD system being used.

## *GHBridge Architecture*

GHBridge constitutes the other half of the VR Tools architecture and deals with the haptic devices and their APIs. Following are some of the important tasks performed by the GHBridge component:

1. Act as an interface to the haptic devices via the haptic API.
2. Use the model tree data, assembly hierarchy and geometry information extracted by the CAD Plugin for building the haptic scene graph.
3. Initialize, setup and run the simulation and collision engines.
4. Perform the constraint management during the simulation.
5. Perform the state management for the simulation components.
6. Manage the virtual hand.
7. Maintain the haptic scene graph and update the visual scene graph via the CAD Plugin.
8. Implement algorithms for constraint and state management.

Figure 5.3 shows all the important classes proposed for the GHBridge component.

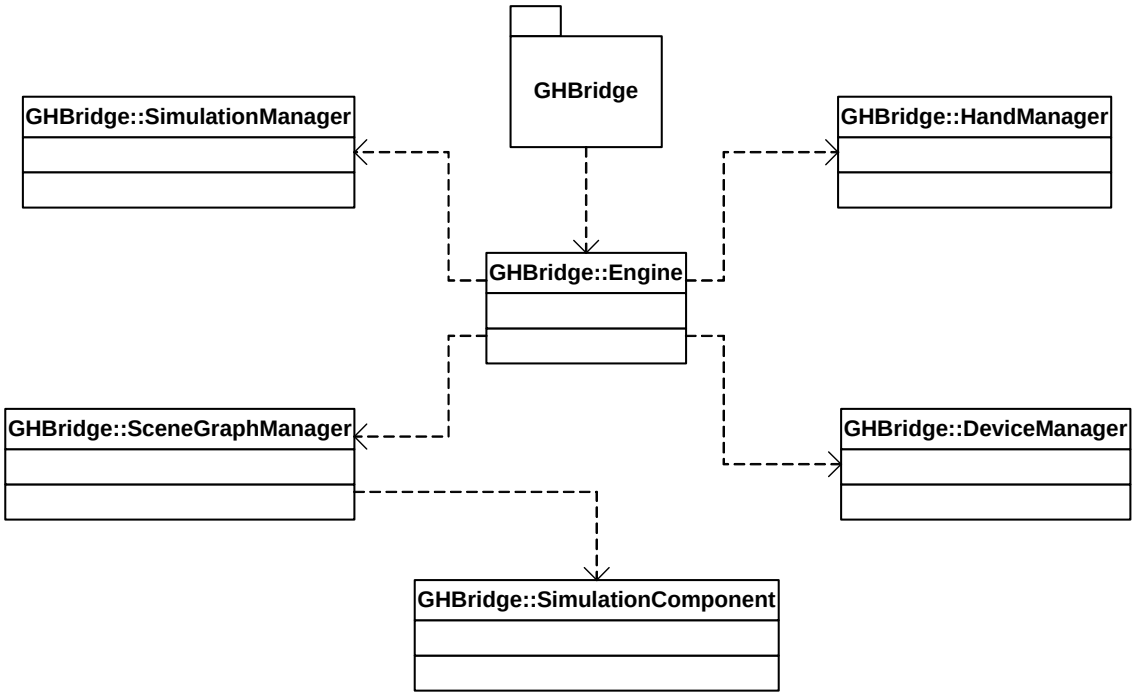


**Figure 5.3: GHBridge Architecture**

The *Engine* class acts as the interface of the GHBridge module. It is primarily responsible for exchanging data and messages with the CAD Plugin. It uses various manager classes for delegating different tasks during the simulation.

*SceneGraphManager* is a class dedicated for managing the haptic scene graph. It uses the assembly tree and geometry information extracted by the CAD plugin and builds the scene graph. *SimulationComponent* class represents 3D components in the simulation such as the assembly component, tool or the fastener. This class encapsulates all the information necessary for completely representing a simulation object.

The *DeviceManager* class handles the haptic devices. The *Engine* uses this class to initialize the devices selected by the user.

The *HandManager* class is used to interface with the virtual hand during the simulation. It implements the gripping algorithms and tracks the state changes of a gripped object. It also extracts the transformation data for all the components of the virtual hand.

## State Management

State management is one of the most critical aspects of VR Tools. The behavior of assembly components, tools and the fasteners depends completely on their states. The components undergo state changes during the simulation and show drastic changes in their behavior. Therefore it is essential to have a robust state management scheme to ensure smooth transition between states. The proposed scheme adopts the standard *Observer* design pattern. This scheme is extensible and can support more states fairly easily. Figures 5.4 and 5.5 show basic state and state management classes.

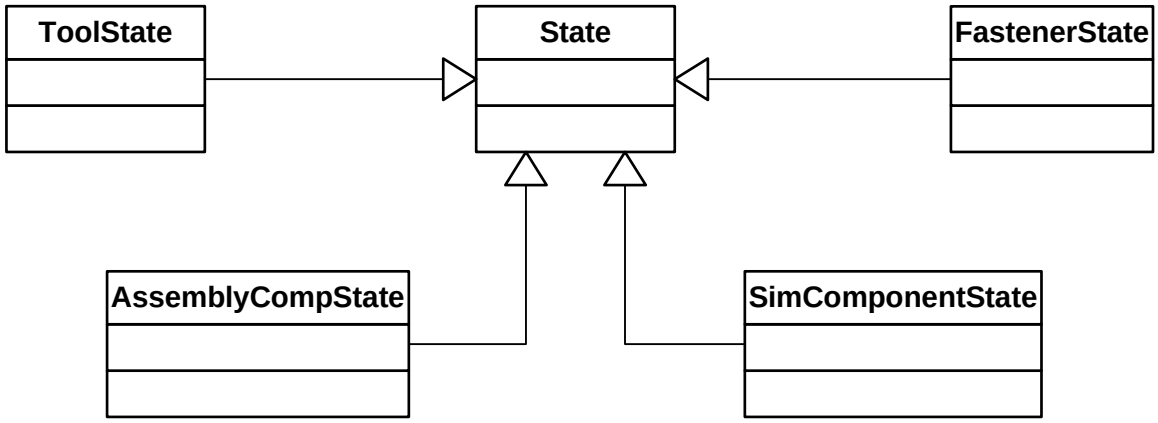


**Figure 5.4: State Classes**

A *SimulationComponent* can be an *AssemblyComponent*, a *Tool* or a *Fastener*. Each of them plays a unique role in the simulation and therefore has a unique set of states associated with it. Each of those state classes can further be extended based on the requirements of the application being developed. In chapter 6 we will see how this class structure is extended.

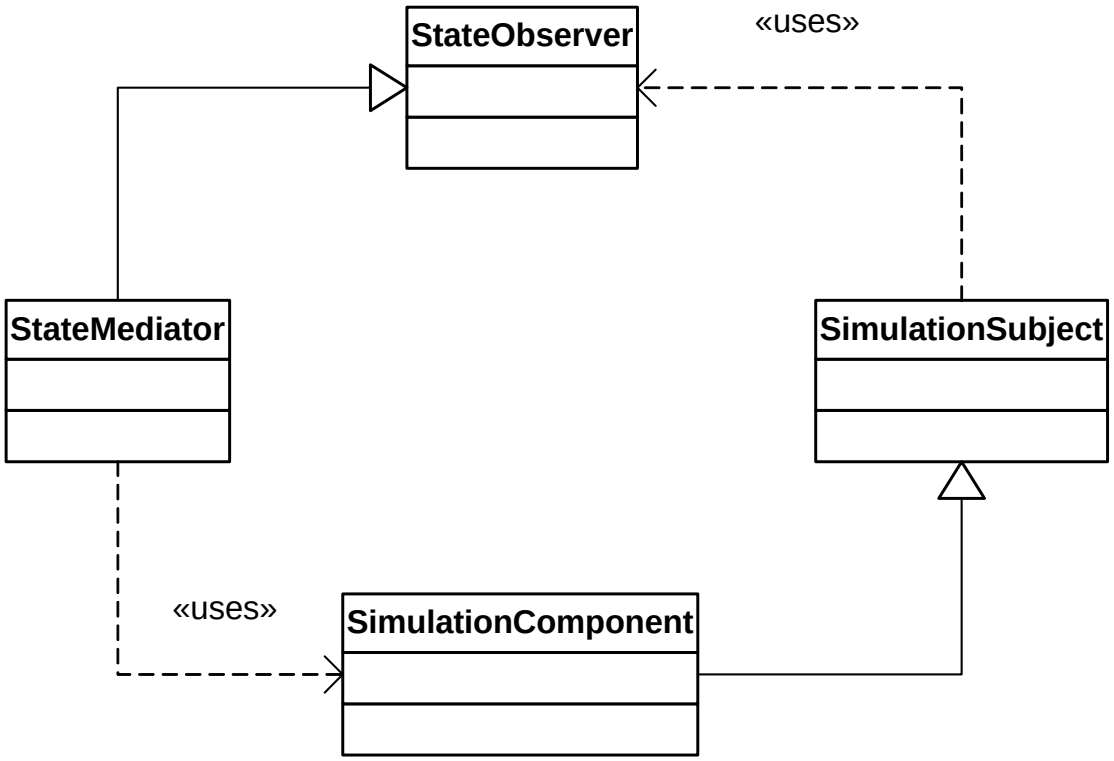


**Figure 5.5: Observer Pattern**

Figure 5.5 shows a schematic of the class structure for the Observer pattern. The *SimulationComponent* has a *StateObserver* associated with it through the *SimulationSubject*. The *StateMediator*, which extends the *StateObserver*, maintains a list of *SimulationComponents* that register interest in its *SimulationSubject*. So any state change the *SimulationSubject* goes through is notified to all the *SimulationComponents* that are registered with the *StateMediator* of the *SimulationSubject*.

## Constraint Management

Interactions between various simulation components such as the tool and the fasteners and the fastener and the hole are governed by the constraints between them. The

state of each component determines the kind of constraints that need to be enforced on it. Figure 5.6 shows a class structure proposed for performing the constraint management.

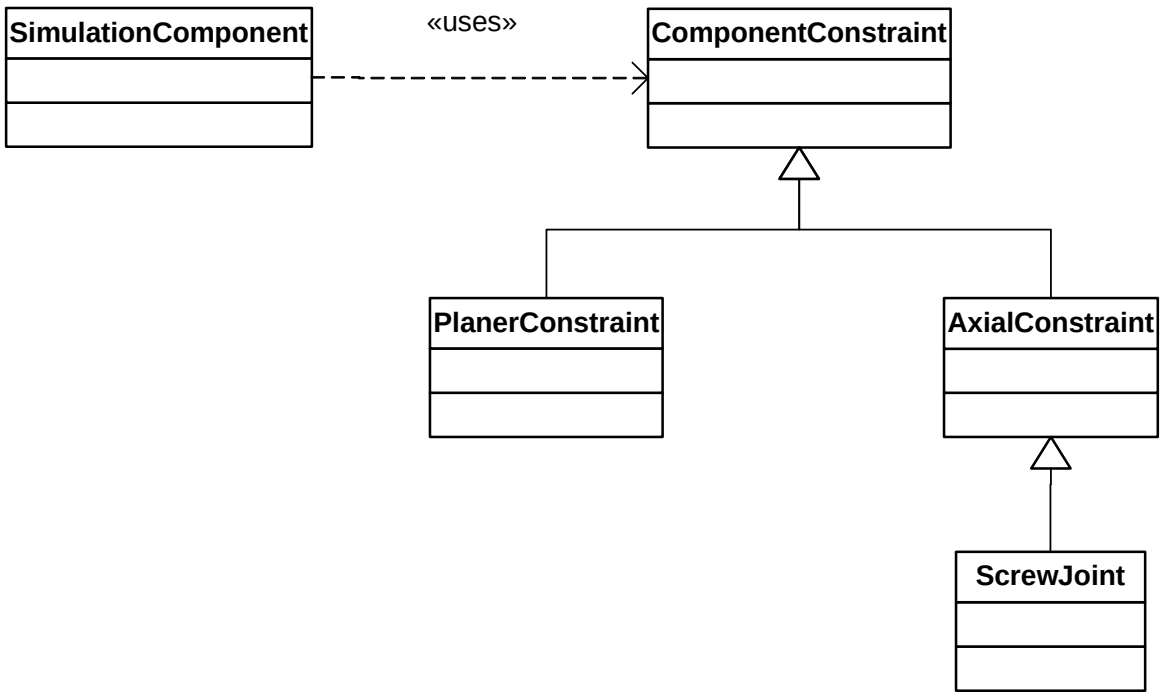


**Figure 5.6 Constraint Management Classes**

For the tool-fastener interaction, a set of planar constraints is sufficient to completely define their relative positions with respect to each other when the tool is engaged with the fastener. When the tool is in close proximity with a compatible fastener, the constraint planes that are close to being coplanar are identified and are used for enforcing the constraints.

An *AxialConstraint* and a *ScrewJoint*, which extends the *AxialConstraint,* are proposed for the fastener-hole interaction. Enforcing the *AxialConstraint* aligns the fastener-axis with the hole-axis. *AxialConstraint* lets the fastener move only along the axis and rotate about it.

The *ScrewJoint* is enforced when the fastener is inside the hole. Chapter 6 takes a closer look at the constraint enforcement algorithms.

## *Tools/Fasteners Library Architecture*

In this section we layout a list of specifications for a Tools/Fasteners library and propose a structure for such a library. Some of the requirements are addressed by the proposed structure while the rest are functionalities that need to be implemented. The implementation details will be specific to the software tools employed for development of the library and are outside the scope of this work. Following are the basic requirements for the Tools/Fasteners library to ensure that the design is extensible and easy to maintain:

1. The library should be an independent entity and should not be tightly integrated with the VR Tools software.
2. It should facilitate addition of new tools and fasteners without requiring changes to the software implementation.
3. It should follow well-defined set of modeling guidelines laid down to help model new VR Tools compatible tools and fasteners.
4. It should be capable of storing the necessary engineering information such as the pitch, designation, etc. for the library components.
5. It should provide a mechanism for classifying and aggregating models into various categories.
6. It should provide an easy way for adding the library models to the simulation.

Figure 5.7 shows a schematic for the proposed structure for the Tools/Fasteners library.

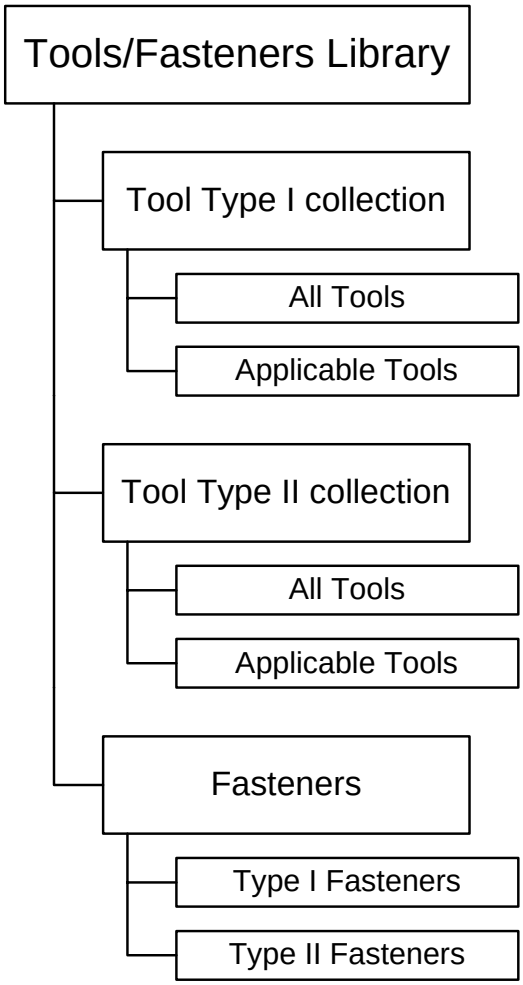


**Figure 5.7 Library Structure**

A tree structure is proposed for managing the tools and the fasteners in a library. The leaf nodes of the tree are collections of categorized components. This structure provides the ability to expand both the scope and depth of categorization. The implementation details of such a library using CATCatalog document supported by CATIA V5 are provided in the next chapter.

In chapter 6 we discuss the details of VR Tools architecture, which is based on the open architecture proposed in this chapter. We also take a close look at the key classes implemented by the CAD Plugin and GHBridge components. Finally we delve into the state and constraint management algorithms implemented for VR Tools.

# CHAPTER 6

# VRTOOLS ARCHITECTURE AND DESIGN DETAILS

Chapter 5 laid out a generic template architecture that can be applied to any combination of CAD system and haptic devices. In this research, CATIA V5 was chosen as the CAD system along with the haptic devices from Immersion Corporation (*CyberGlove*) and Ascension Technologies (Flock of Birds). This chapter discusses in detail VR Tools architecture, which is based on the open architecture. It takes a deeper look at the CAD Plugin implemented as a CATIA Workbench (VRTools workbench) and the *GHBridge* built using the VirtualHand Toolkit (VHT). It also discusses the structure and role of key classes from the VRTools workbench and the *GHBridge*. Finally it delves into the Tools/Fastener library (*VRToolsCatalog*), which is implemented using the Catalog document functionality supported in CATIA V5.

## *VRTools Workbench Design*

VRTools Workbench is a CATIA CAA application and hence follows a CAA specific deployment model. Figure 6.1 shows the deployment diagram for the VRTools workbench.

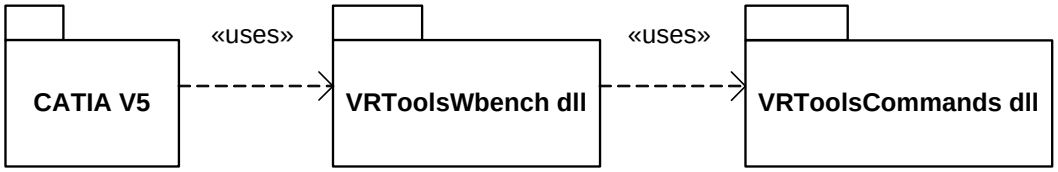


**Figure 6.1 VRTools Workbench Deployment**

The *VRToolsWbench.dll* and the *VRToolsCommands.dll* together represent the CAD Plugin introduced in the previous chapter. *VRToolsWbench.dll* is the workbench dll and is

responsible for making the workbench menus, toolbars and commands accessible through the CATIA user interface. The workbench relies on the *VRToolsCommands.dll* for implementing these commands. Therefore, *VRToolsCommands.dll* is the main module that implements all the CAD Plugin functionality. Figure 6.2 shows various *VRToolsCommands.dll* command classes used by *VRToolsWbench.dll* for implementing the commands.

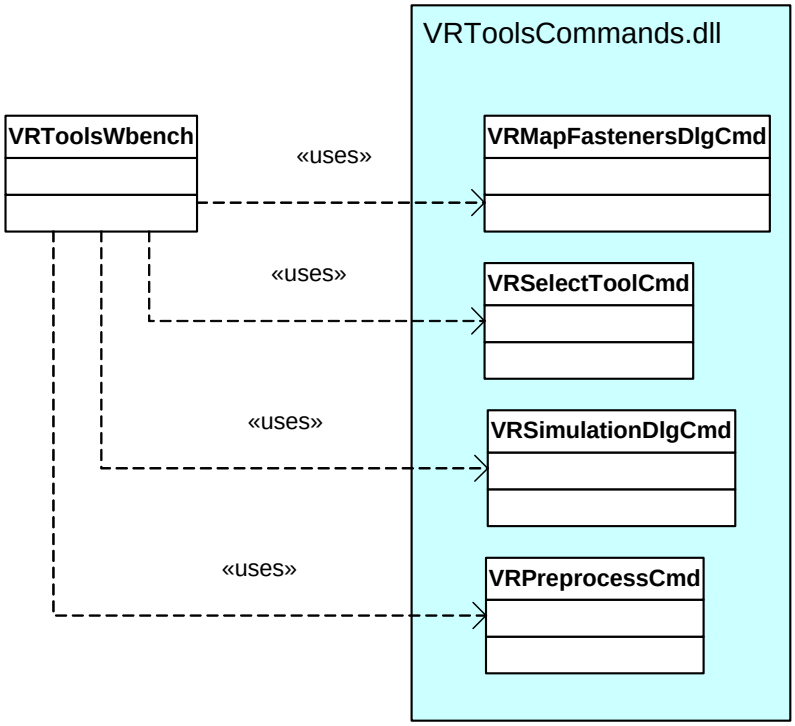


**Figure 6.2 VRTools Workbench**

## VRToolsCommands Design

*VRToolsCommands.dll* implements the CAD Plugin architecture. It exposes the *VRMapFastenerDlgCmd, VRSimulationDlgCmd, VRPreprocessCmd and the VRSelectToolCmd* to the VRTools Workbench. The *VRSimulationDlgCmd* also interfaces with the GHBridge module. All the classes implement CATIA CAA API code for interacting with the CAD system for their particular tasks. Figure 6.3 shows all the important *VRToolsCommands* classes.

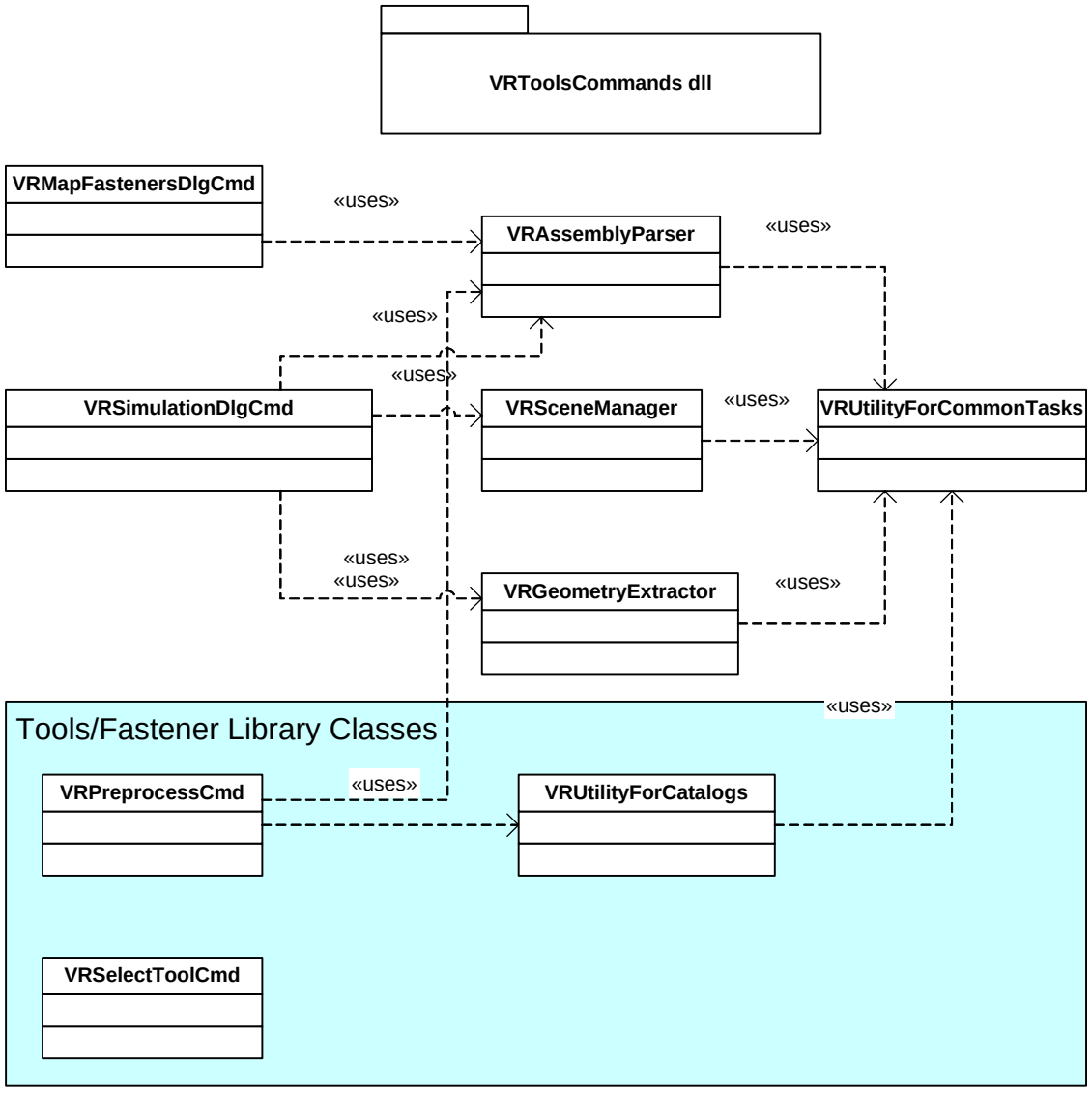


**Figure 6.3 VRToolsCommands Classes**

***Key Classes and Their Roles***

*VRMapFastenersDlgCmd*: This class provides the user with a graphical user interface (GUI) for defining the component fastener mapping for the CAD model during the setup phase of the simulation. It identifies the fasteners and assembly components in the model. It also implements the functionality for storing the component-fastener mapping with the CAD model so that it has to be done only once for each model.

*VRSimulationDlgCmd:* This is the most important class in *VRToolsCommands*. It provides the user with a GUI with the following functionalities:

1. Selecting haptic devices to be used during the simulation.
2. Specifying a scaling factor for the virtual hand.
3. Starting and stopping the simulation.

This class also interfaces with the *GHBridge* and provides it all the necessary information for setting up the haptic scene graph. It also registers callbacks with the *GHBridge*, which are used to keep the CATIA graphics synchronized with the haptic scene graph.

*VRAssemblyParser:* This class is responsible for implementing all the assembly tree structure related functionality. It mainly deals with the assembly hierarchy and the transformation for the assembly components. *VRMapFastenersDlgCmd* and *VRSimulationDlgCmd* use this class for performing their tasks.

*VRSceneManager:* This class handles the CATIA graphics. It extracts the graphical representation for the assembly components. It updates the transformations for the graphical representation on each frame and refreshes the scene. This class is responsible for keeping the CATIA graphics synchronized with the haptic scene graph.

*VRGeometyExtractor*: The primary role of this class is to extract the geometric and topological data for each assembly component. *VRSimulationDlgCmd* uses this class for extracting the point cloud data that is passed on to the *GHBridge* for constructing the scene graph nodes.

*VRUtilityForCommonTasks*: This is a utility class and implements basic CATIA functions such as document handling. All the other classes use it.

*VRPreprocessCmd*: This command preprocesses the VRToolsCatalog, which acts as the Tools/Fasteners library. This class uses the *VRAssemblyParser* and extracts the information for the fasteners used in the model. Based on this information it detects the tools that are compatible with the current model and filters them out to the *Applicable* chapter of the *VRToolsCatalog*.

*VRSelectToolCmd*: This command class enables the user to import new tools in the simulation environment. It connects to the *VRToolsCatalog* and brings up a dialog box that lets the user browse through the available tools in the catalog. The user can then import the tools of his/her choice into the current model.

*VRUtilityForCatalog*: This is also a utility class, which specifically deals with the CATIA catalog documents. The *VRPreprocessCmd* and the *VRSelectToolCmd* classes depend on this class for the basic Catalog related functionalities.

## *GHBridge Design*

This section takes a detailed look at the *GHBridge* architecture implemented for VRTools. It follows the template laid out in chapter 5 and customizes it for specific haptic devices and their API. This implementation of *GHBridge* is based on Immersion Corporation's *CyberGlove* and Ascension Technologies' *Flock of Birds* for the haptic devices. Immersion Corporation provides the *VirtualHand Toolkit* (VHT), which is an API for interfacing with the haptic devices used for this implementation.

The GHBridge has a complex class structure and is responsible for performing a number of complex tasks during the simulation process. We break down the *GHBridge* architecture into several pieces based on their roles. Then we look at the architecture of each individual piece. Figure 6.4 shows the high-level class diagram of *GHBridge*. All the classes are classified into 4 different groups. The following section takes a closer look at each group. All the classes whose names start with *vht* are VirtualHand Toolkit's classes. The classes whose names start with *GHB* are the *GHBridge* classes.

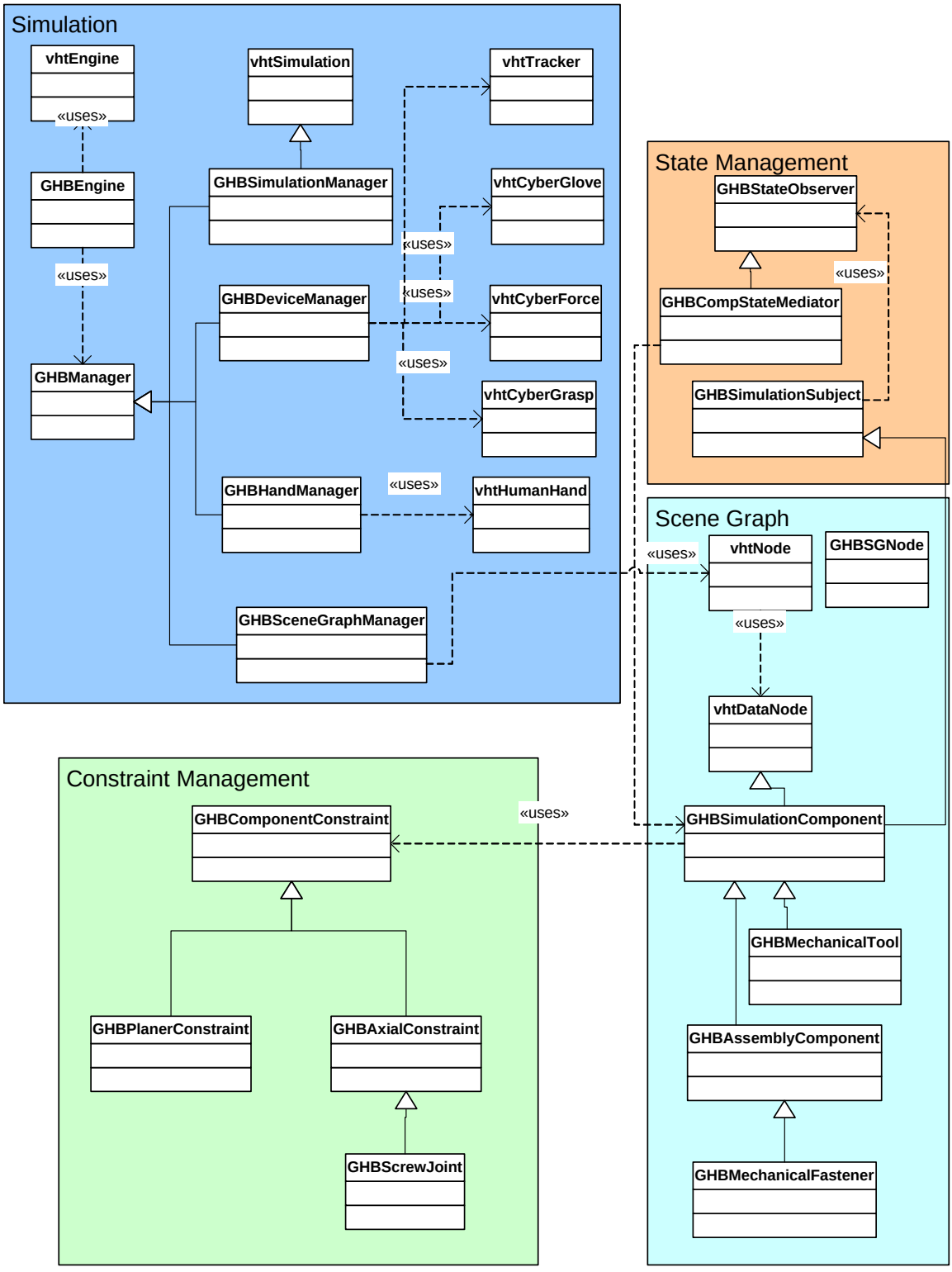


**Figure 6.4 GHBridge Class Diagram**

### Simulation Classes

*GHBEngine*: This is the key *GHBridge* class that manages the complete simulation. Some of the important tasks carried out by this class are as follows:

1. It acts as the *GHBridge* interface to the *VRToolsCommand* (CAD Plugin).
2. It initializes the haptic devices.
3. It initializes all the manager classes needed for the simulation.
4. It instantiates the *vhtEngine* class which has the functionality for starting and stopping the simulation.
5. It uses the *SceneGraphManager* for building the haptic scene graph.
6. It updates the scene graph and all the managers on each simulation frame.
7. It maintains the callbacks for the *VRToolsCommand* and uses them to refresh the visual scene graph.

Figure 6.5 shows an activity diagram depicting the sequence of tasks that are performed while starting the simulation.

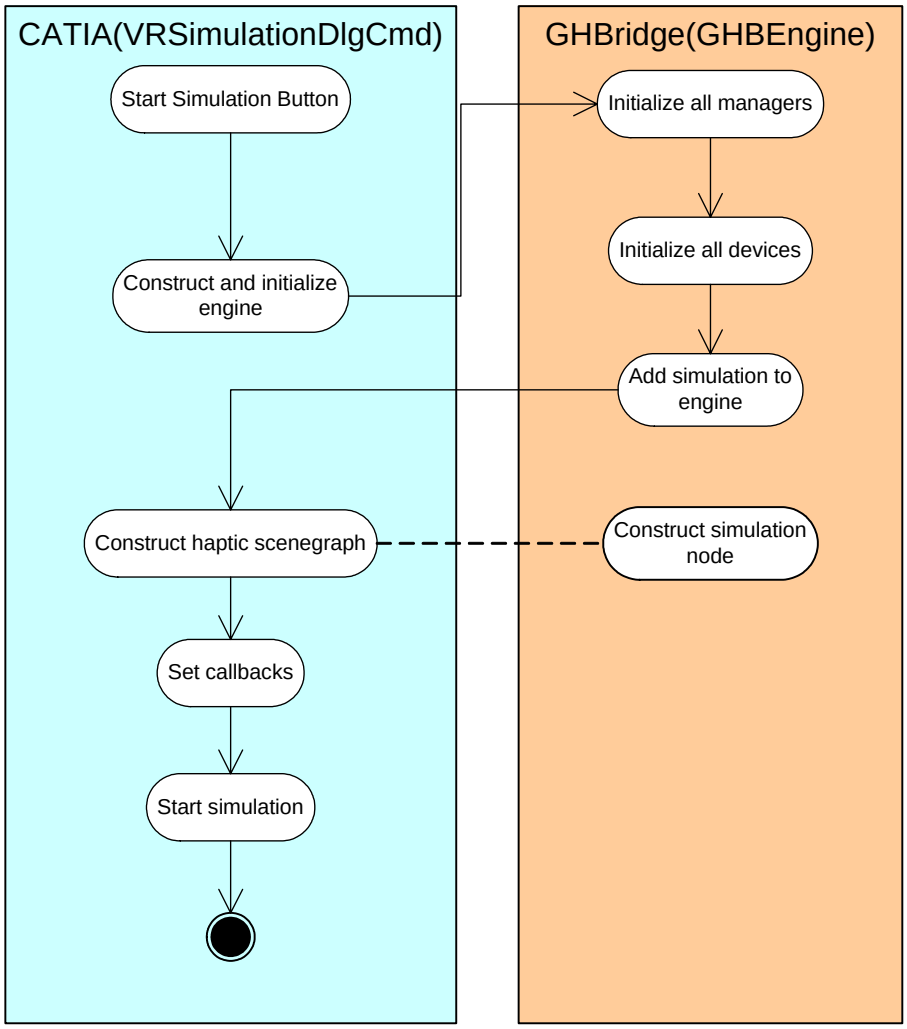


**Figure 6.5 Start Simulation Event Sequence**

*GHBSimulationManager:* This class derives from the *vhtSimulation* class and is registered with the *vhtEngine*. It overrides the *handleConstraints* function, which is called by the VHT simulation thread on every frame. This function gives the control back to the user on each frame.

*GHBDeviceManager:* This class handles all the haptic devices used during the simulation. The *GHBEngine* feeds the users choice of devices to the device manager during the setup phase. The *GHBDeviceManager* uses *vhtTracker, vhtCyberGlove, vhtCyberGrasp* and *vhtCyberForce* classes for initializing the *Flock of Birds, CyberGlove, CyberGrasp* and *CyberForce* respectively. It also

used by the *GHBHandManager* to instantiate the *vhtHumanHand* class that represents the virtual hand.

*GHBHandManager:* This class uses the *vhtHumanHand* class to access the virtual hand functionality. Its primary role is to implement the gripping algorithm and determine whether an object is gripped. It is also responsible for providing the information pertaining to the grasped object and the virtual hand transformation information. Figure 6.6 shows the flowchart for finding the gripped object.

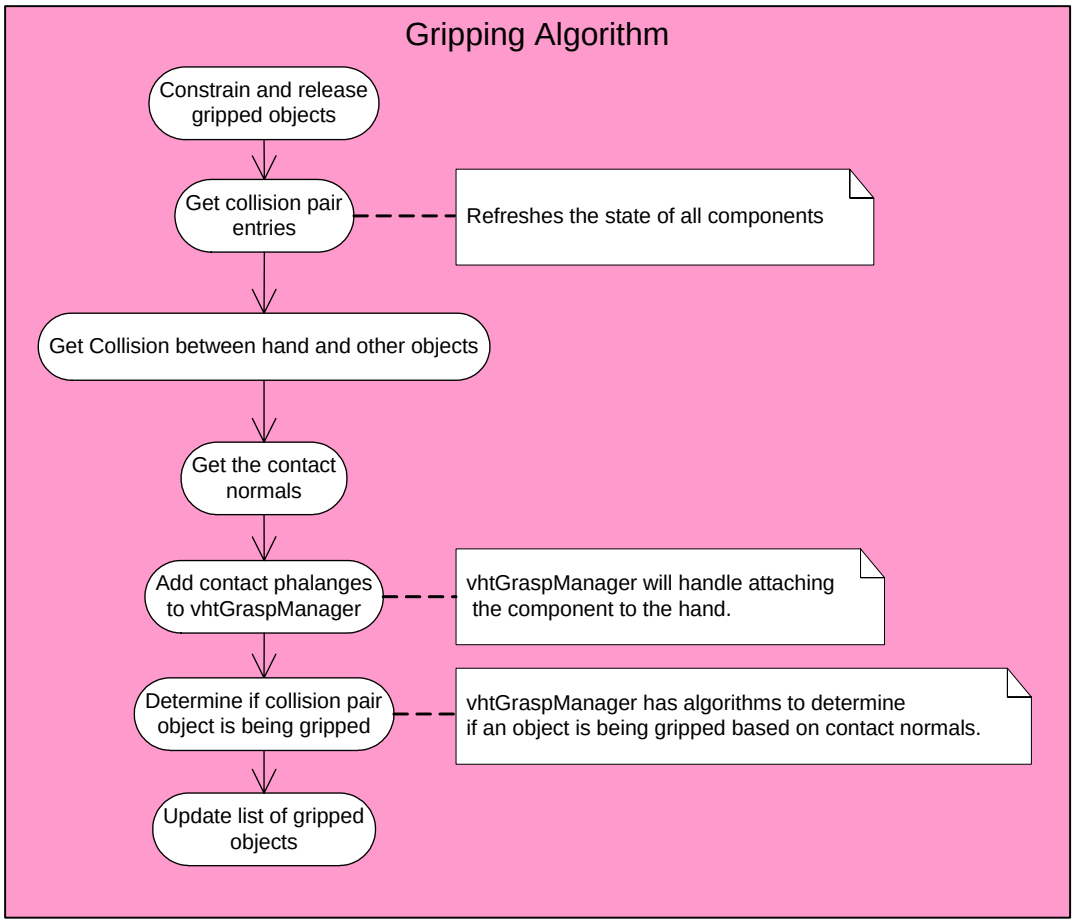


**Figure 6.6 Gripping Algorithm**

*GHBSceneGraphManager:* The primary role of this class is to build and maintain the haptic scene graph. It obtains the information for the assembly hierarchy, geometric data and type

of the components from the CAD Plugin and uses it create haptic scene graph nodes. It also initializes the collision factory, which enables collision detection between different simulation objects. It also initializes the states for all the simulation components and does the constraints mapping based on the component-fastener mapping defined by the user during the setup.

### Scene Graph Classes

*VhtNode*: This is the VirtualHand Toolkit's haptic node class. All the classes that can be used as node of the haptic scene graph extend this class. It maintains a link to the *vhtNodeData*, which is a neutral scene graph node.

*VhtNodeData*: This VirtualHand Toolkit's class can be extended to store any information that has to be stored in the neutral dual of the haptic node on the scene graph.

*GHBSGNode*: This class is used for building the haptic scene graph. It is implemented by the *GHBridge* and exported for the *VRToolsCommands* module. The *VRToolsCommands* builds the entire scene graph using the *GHBSGNode* class. The *GHBSceneManager* then uses this tree for building the haptic scene graph. Thus, this class acts as a medium for data transfer between the CAD Plugin and the GHBridge.

*GHBSimulationComponent:* This is the base class for all the types of simulation components. This class represents the neutral dual of the haptic node. This class extends the *vhtNodeData* class. It stores important information such as the grasped-state, current-state, constraints list, component name, etc. for the simulation components.

*GHBMechanicalTool*: This class extends the *GHBSimulationComponent* class and for the tool specific functionalities. It stores the tool state, list of compatible fasteners and the tool designation properties.

*GHBAssemblyComponent*: This class represents an assembly component. It maintains a list of mapped fasteners. The state of the assembly component depends on the states of the fasteners that are mapped to it. We will take a closer look at the state management architecture later in this chapter.

*GHBMechanicalFastener:* Fastener also being an assembly component extends the *GHBAssemblyComponent* class. It manages fastener specific data such as attached tool information, designation, its state, pitch and shank length.

### Constraint Management Classes

Simulation components in GHBridge maintain their own list of constraints, which is initialized, based on the tool fastener compatibility. This implementation does not use assembly constraints for placing the assembly components in their assembled locations except for the threaded fasteners. The assembly functionality is based on the initial transformations of the components at the beginning of the simulation. It is assumed that all the components are in their assembled state and their current transformation is stored. The constraint checking and enforcing is implemented by the state classes for all the components. Each constraint class implements the *CheckConstraint* function for checking if the mating entities are close enough for the constraint to be enforced. It also implements the *EnforceConstraint* method, which enforces a validated constraint on the simulation components. Next section takes a closer look at the constraint and state management implementation.

*GHBComponentConstraint*: This is the base class for all the constraint types. It provides functionality for maintaining a list of matching constraints. It also stores the current state of the constraint along with the applied constraint information if the constraint is in *applied* state. It also knows its owning simulation component, which is used by the state management algorithms.

*GHBPlanarConstraint*: This class extends the *GHBComponentConstraint* class for implementing the planar constraint. When applied it ensures that the two planes, defined by an origin point and a normal vector are coplanar. The sensitivity of the constraint is controlled by the following two factors:

- DEFAULT_ALIGNMENT_TOLERANCE: Tolerance for alignment between the normals for the mating planes.
- DEFAULT_SEPARATION_TOLERANCE: Tolerance for minimum distance between the mating planes.

*GHBAxialConstraint*: This class also extends the *GHBComponentConstraint* for implementing the axial constraint used for mating the fasteners with their holes. This constraint forces two axes to be coaxial when enforced.

It also uses the DEFAULT_ALIGNMENT_TOLERANCE and the DEFAULT_SEPARATION_TOLERANCE factors for controlling the sensitivity of the constraint.

*GHBScrewJoint*: This class extends the *GHBAxialConstraint* class for emulating the screw joint behavior. It enforces a screw joint along with the axial constraint. This constraint is applied to the fasteners when they are inside the hole. The axial translation along the hole axis can

only be achieved by rotating the fastener. The magnitude of axial displacement is governed by the pitch value for the fastener.

### State Management Classes

The standard observer pattern has been implemented for the state management. The states of the simulation components are interdependent. It is therefore necessary to propagate a state change event for a component to the other components that might be dependent on it for their states. Figure 6.7 shows the class diagram for all the different states that the simulation components can be in based on their type.

Following is a brief description for the states of the components:

1. *GHBSimulationComponent*
    a. GHBSimCompNotGrasped: The component is not grasped by the hand.
    b. GHBSimCompGrasped: The component is grasped by the hand.
2. *GHBMechanicalTool*
    a. GHBToolDisengaged: The tool is not attached to a compatible fastener.
    b. GHBToolEngaged: The tool is attached to a compatible fastener.
3. *GHBAssemblyComponent*
    a. GHBAsmCompUnconstrained: None of the fasteners mapped to the assembly component are in place. This component can be grasped and disassembled.
    b. GHBAsmCompPartiallyConstrained: Some of the fasteners mapped to the assembly component are in place.

c. GHBAsmCompFullyConstrained: All the fasteners mapped to the assembly component are in place.


**Figure 6.7: State Classes**

4. *GHBMechanicalFastener*

a. GHBFsntrUnconstrained: The fastener is completely out of its hole and no constraints are applied.

b. GHBFsntrPartiallyConstrained: The fastener is partially out of the hole and is under *GHBScrewJoint* constraint.

c. GHBFsntrFullyConstrained: The fastener is fully tightened in it hole and can move only in one direction subject to the *GHBScrewJoint* constraint.

***State Change Mechanism***

Each simulation component has a current state. On every frame the *GHBEngine* maintains a watch list of simulation components that are grasped or could potentially undergo a change in position. After the haptic scene graph is updated, each component on the list is asked to update itself. This update is handled by the current state of the component. All the state classes implement an *Update* function that has all the logic to decide if a state change is necessary. If necessary, the current state changes the state of the component and propagates this state change event to all the other components that have registered interest in its owner component. This is achieved because the *GHBStateObserver* class. Each component has a *GHBStateObserver* associated with it. This class is responsible for maintaining the list of simulation components that register interest in the owner component.

For instance, when a fastener is mapped to an assembly component, the assembly component registers itself with the *GHBStateObserver* of the fastener. If a tool is engaged with the fastener it gets added to the *GHBEngine's* watch list. The fastener is in *GHBFsntrPartiallyConstrained* state if it is being rotated and taken out of its hole. On every frame *GHBFsntrPartiallyConstrained* state's *Update* method uses the *GHBScrewJoint* constraint for the fastener to determine if it is completely out of the hole. As soon as the fastener is completely out of the hole a state change is triggered and the fastener goes to *GHBFsntrUnconstrained* state. At this point the *GHBState* class notifies the *GHBStateObserver*

associated with the fastener. The *GHBStateObserver* then notifies all the assembly components that have registered interest in this fastener. The assembly components then update their own state if necessary and similar sequence of events is triggered recursively until the state of the entire scene graph is synchronized. Figure 6.8 shows a sequence diagram for the relatively less complicated event of grasping a component.

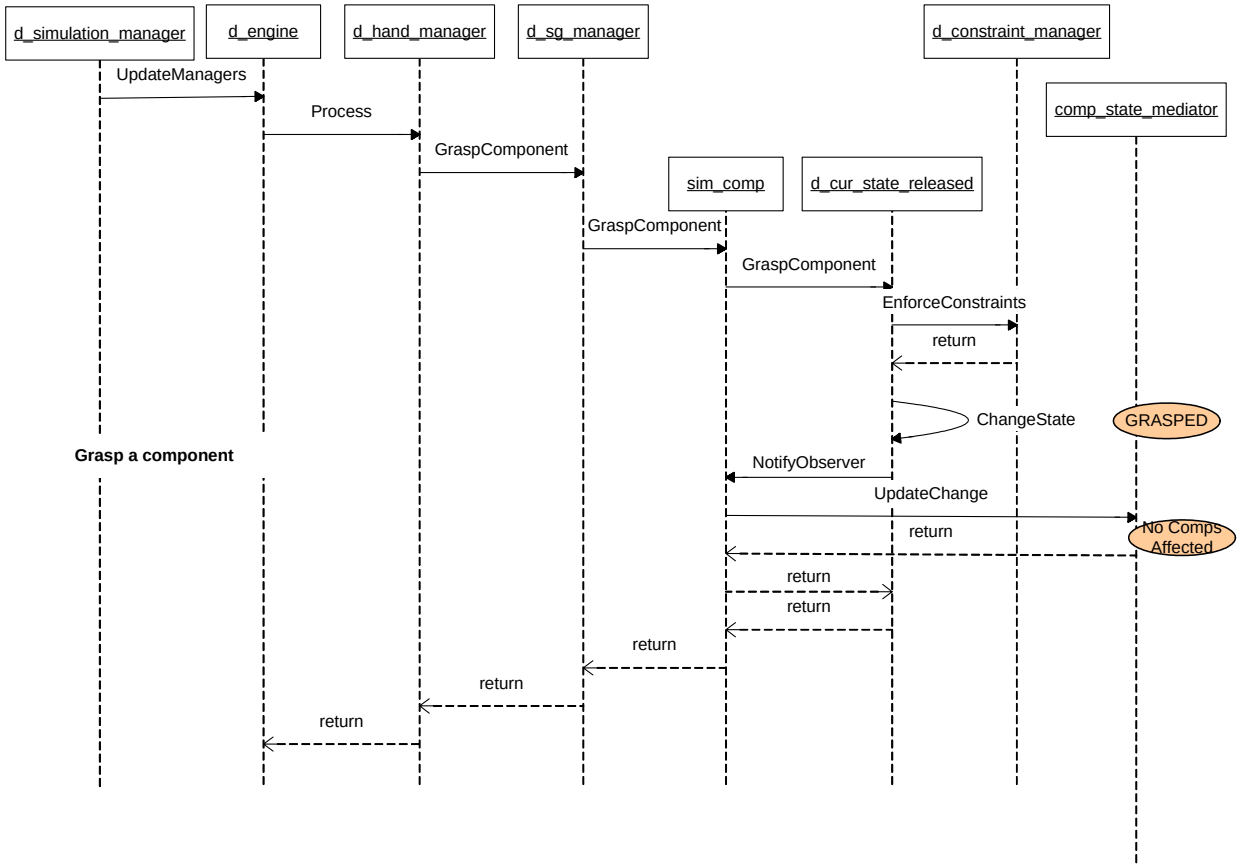


**Figure 6.8 Component Grasping Sequence Diagram**

## *VRToolsCatalog Design*

CATIA V5 supports a *Catalog* document capability, which satisfies all the requirements laid out for the Tools/Fasteners library in chapter 5. Hence the library is created as a CATIA V5 catalog and is named *VRToolsCatalog.Catalog*.

### VRToolsCatalog Structure

The VRToolsCatalog adopts the library structure proposed in the previous chapter. Following aspects of the catalog relate to the specifications from the proposed architecture.

1. The VRToolsCatalog has a tree structure consisting of chapters, sub-chapters and families for maintaining the tools and fasteners models as shown in figure 6.9.
2. The top-level chapter in this catalog is named '***VRTools'.***
3. Each type of tool has a sub-chapter under the VRTools chapter.
4. All the fasteners are managed under the 'Fasteners' sub-chapter.
5. Each of the tool sub-chapters has two families named as '***All'*** and '***Applicable'.***
6. The '***All'*** family lists the description for all the tools of a particular type.
7. '***Applicable'*** chapter is used to filter out the tools that are compatible with the current CATIA assembly on which the simulation is being run.
8. Each description in the catalog also stores information such as the *Designation*, *Pitch*, *Length* and *Name* for the components.

9. The VRTools WorkBench has a search for suitable tools command, which filters out the relevant tools for a given model and adds them to the '***Applicable***' chapter for each type of tool.

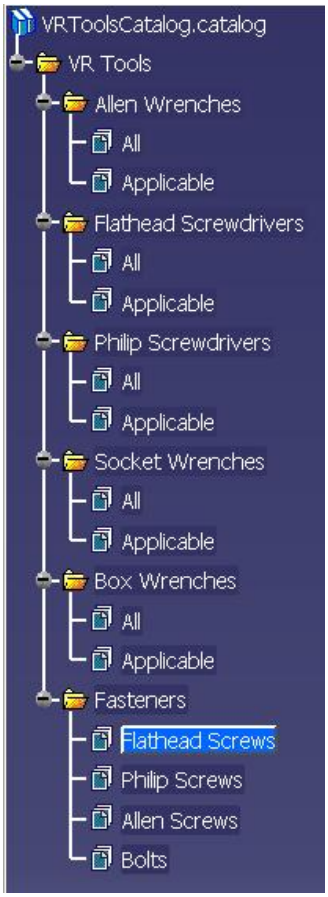


**Figure 6.9 VRToolsCatalog structure**

## Updating the VRTools catalog

CATIA V5 catalog documents are well structured and fairly easy to maintain. Following guidelines need to be followed to maintain the integrity of the catalog data:

1. The ***VRToolsCatalog.Catalog*** is located in *C:\Program Files\VRTools\Catalog*

2. The Parts corresponding to the catalog entries are placed *C:\Program Files\VRTools\Parts*. This is not an requirement and parts in the catalog can be located anywhere.

3. New VR Tools compatible models for tools and fasteners need to be added to the VRToolsCatalog along with the necessary metadata in order to be able to use them with the VRTools workbench.

4. A chapter or a family can be made current by double clicking on it in the tree view. The current node is highlighted in blue.

5. A component can be added to a current family by clicking on the ***Add Component*** button on the ***Data*** toolbar.

6. The user then needs to select a document for the component and populate the key word values for the component in the ***Description Definition*** dialog box as shown in figure 6.10.

7. The ***Name*** keyword for the component should match the ***PartNumber*** inside the CATIA document.

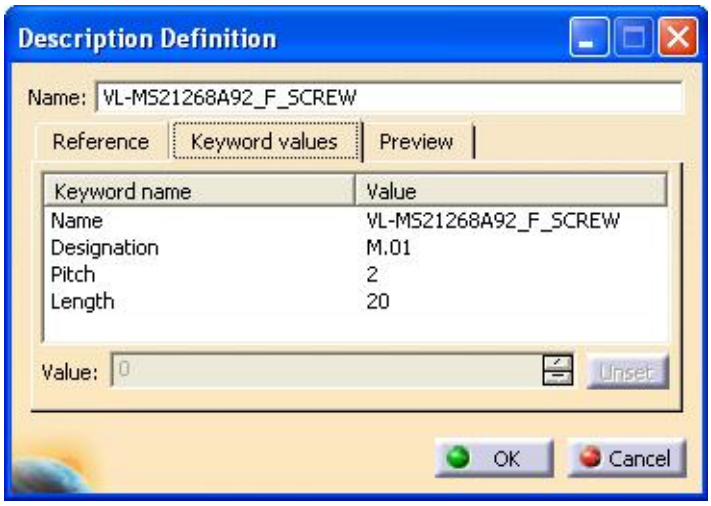


**Figure 6.10 Add Component**

## *Modeling Guidelines*

The VRToolsCatalog contains all fasteners and tools that are compatible with VR Tools. Tools and fasteners are mapped using the designation of the fastener. The name (part number) of the tool should have the designation of the fastener appended at the end with an underscore before the designation. For example, if a bolt has a designation of "M16", any appropriate tool in the catalog that has "_M16" at the end of the name will be considered a matching tool. This allows VR Tools to regulate the sizes of tools that can be used with specific fasteners.

### VR Tools Fasteners

This section lists all the fastener types supported in VR Tools. Each fastener type has a list of requirements that needs to be met for making the fastener VR Tools compatible.

***Hex Bolts***

1. Part Number should have "***BOLT***" in it

2. A reference line should be created along the bolt axis and named ***VRT_Axis***

    a. The axis should start at the base of the head and point downwards

3. Reference planes should be created on the 6 flat faces of the hexagonal bolt head and named ***VRT_Plane1*** through ***VRT_Plane6***

    a. These planes should be created offset from the surfaces with an offset of 0.0 as shown in figure 6.11

4. A reference plane ***VRT_A_Plane*** should be created at the base of the bolt head such that the plane normal points downward into the bolt shank

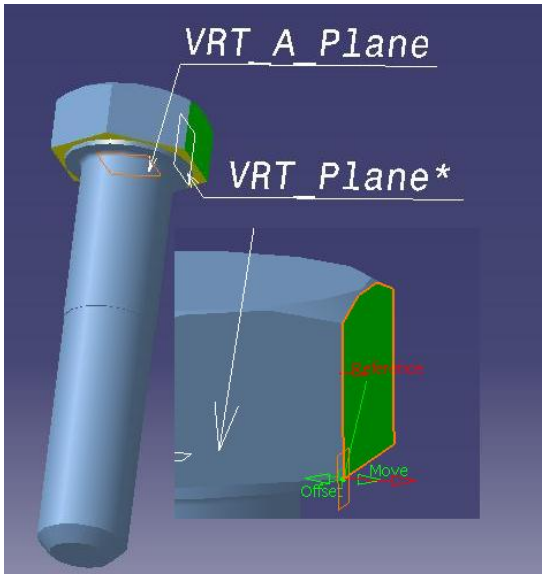


**Figure 6.11 Catalog Hex Bolt**

***Flathead Screws***

1. Part Number should have "***F_SCREW***" in it
2. A reference line should be created along the bolt axis and named ***VRT_Axis***
    a. The axis should start at the base of the head and point downwards
3. Reference planes should be created on both the vertical faces of the screw head slot and should be named ***VRT_Plane1*** and ***VRT_Plane2*** as shown in figure 6.12
    a. These planes should be created offset from the surfaces with an offset of 0.0
4. A reference plane ***VRT_A_Plane*** should be created at the base of the screw head slot such that the plane normal points downward into the screw shank

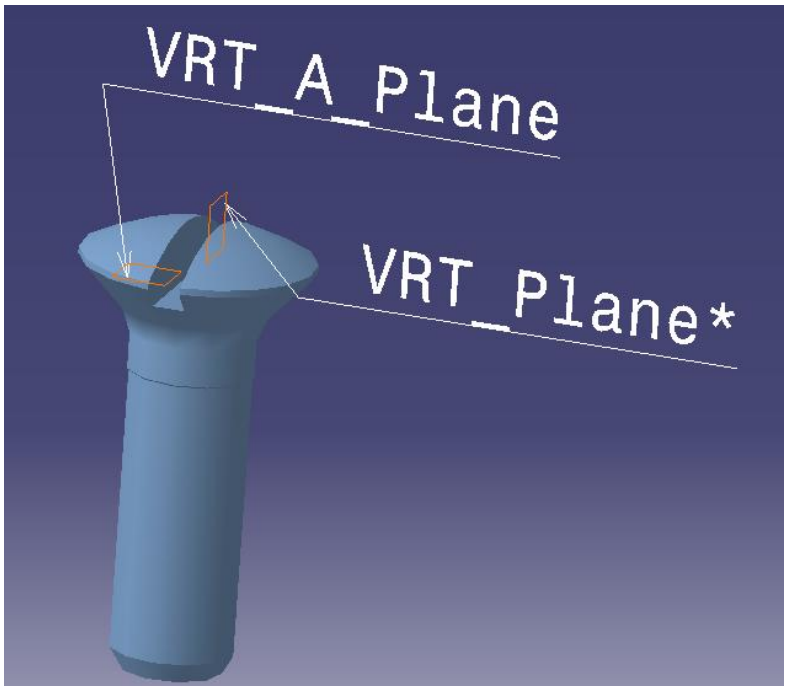


**Figure 6.12 Catalog Flathead Screw**

***Philip Screws***

1. Part Number should have "***P_SCREW***" in it

    a. A reference line should be created along the screw axis and named ***VRT_Axis***

2. The axis should start at the base of the head and point downwards

3. Reference planes should be created on 4 flat faces of the screw head slot and named ***VRT_Plane1*** through ***VRT_Plane4*** as shown in figure 6.13

    a. These planes should be created offset from the surfaces with an offset of 0.0

4. A reference plane ***VRT_A_Plane*** should be created at the base of the screw head slot such that the plane normal points into the screw shank

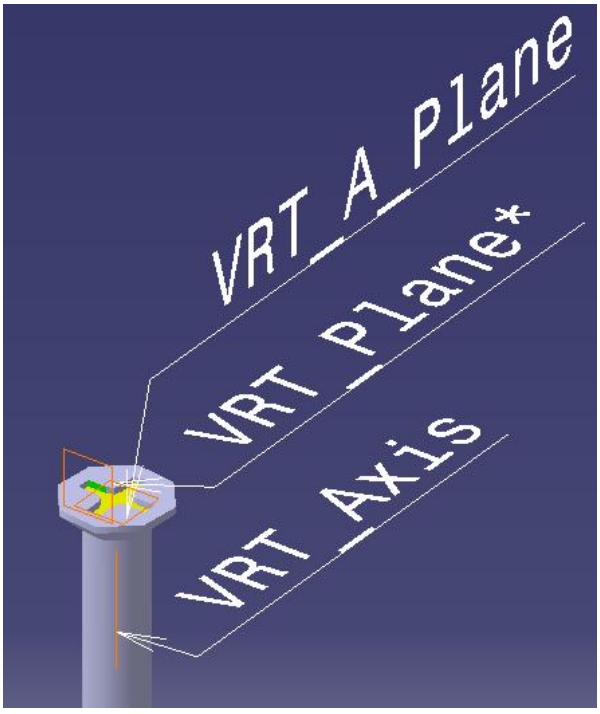


**Figure 6.13 Catalog Philips Screw**

### *Allen Screws*

1. Part Number should have "***A_SCREW***" in it
2. A reference line should be created along the screw axis and named ***VRT_Axis***
    a. The axis should start at the base of the head and point downwards
3. Reference planes should be created on the 6 flat faces of the hexagonal screw head slot and named ***VRT_Plane1*** through ***VRT_Plane6*** as shown in figure 6.14
    a. These planes should be created offset from the surfaces with an offset of 0.0
4. A reference plane ***VRT_A_Plane*** should be created at the base of the screw head slot such that the plane normal points into the screw shank

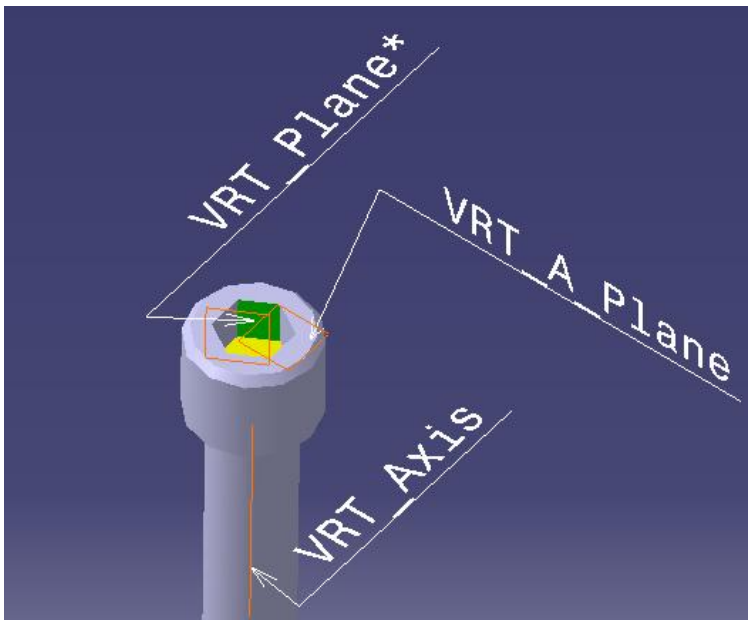


**Figure 6.14 Catalog Allen Screw**

# CHAPTER 7

# IMPLEMENTATION AND RESULTS

This chapter gives an overview of the implementation and deployment of the developed solution. It also describes the user interface and functionality details of VR Tools. Finally, it discusses the modeling guidelines for making tools and fasteners VR Tools compatible.

## *Implementation and Deployment*

VR Tools consists of three software modules namely, *VRToolsWbench.dll, VRToolsCommands.dll* and *GHBridge.dll. VRToolsWbench.dll* and *VRToolsCommands.dll* are developed in C++ using the CAA RADE (Rapid Application Development Environment) addin in Visual Studio 6.0.

### CATIA V5 Workshops and Workbenches

The CAA V5 application window includes commands in menus and in the standard toolbars that are common to all document types (Parts, assemblies etc.). Each document type is associated with a workshop that includes commands pertaining to that document type, which are added to the common menus and to the standard toolbars. Changing the active document to another document changes the active workshop if the new active document's type is different from the type of the previous document. A workshop can include workbenches to gather commands dedicated to specific tasks to structure the end user interface. Only one workbench can be active at any time.

*VRToolsWbench.dll* implements the VRTools workbench. CATIA V5 application knows about the workbench through an environment file that is used while launching the application. The *VRToolsCommands.dll* implements all the commands exposed by the workbench to the end user via toolbars and menus.

### GHBridge

The GHBridge as discussed in the earlier chapters uses the API for the haptic devices and implements the haptic scene graph. It is implemented as a C++ shared library using Visual Studio 6.0.

## *VR Tools User Interface*

---

The VRTools workbench exposes its functionality to the user via a set of commands made available through a docking *VRTools Manage* toolbar shown in figure 7.1.


**Figure 7.1 VR Tools Toolbar**

The toolbar provides buttons for the following commands.

### Component-Fastener Mapping

This command enables the user to define associations between the assembly components and the fasteners used in the model.

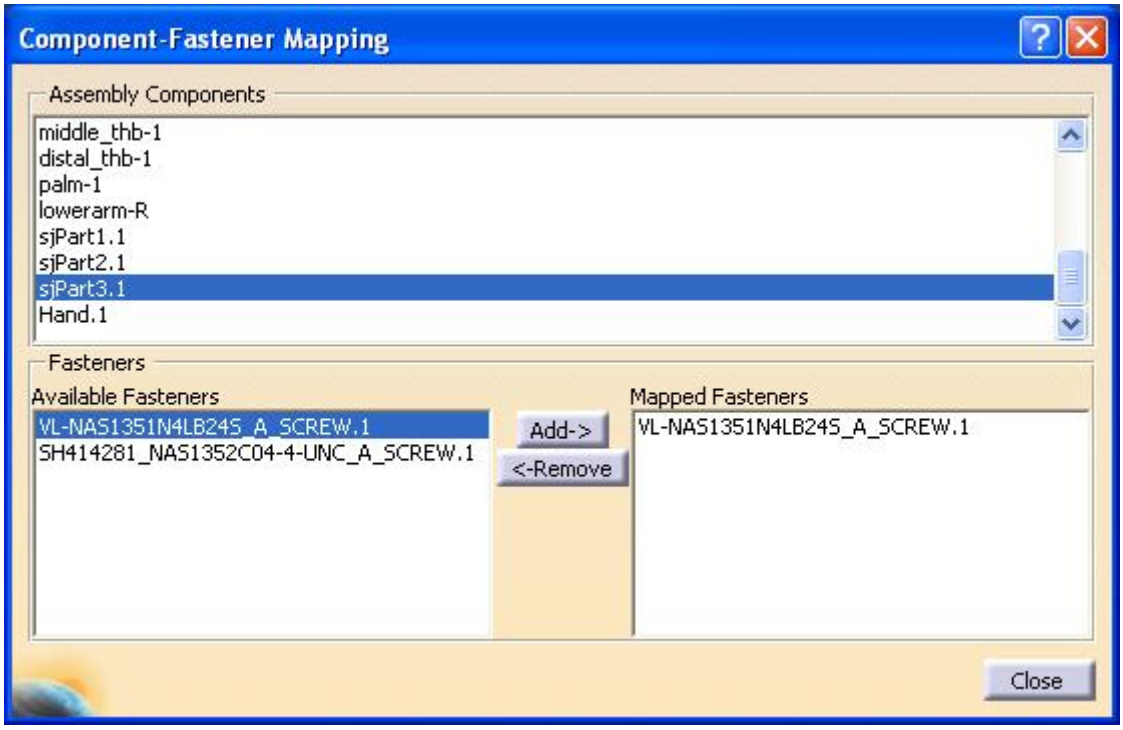


**Figure 7.2 Component-Fastener Mapping**

As shown in figure 7.2, the user can select one or more assembly components. If a single assembly component is selected, a list of fasteners associated with it is shown under the Mapped Fasteners list. The user can update the mapping by adding or removing mapped fasteners by selecting them from appropriate lists and clicking the Add/Remove buttons. The mappings defined using this utility are stored permanently with the CAD model as meta-data. Hence the user has to go through the process of mapping fasteners to assembly component only once for each assembly.

### Search for Suitable Tools

This command preprocesses the VRTools catalog and finds the tools that are compatible with the current assembly model. The tools are matched with the fasteners based on their type and designation and added to the Applicable chapter(s) as shown in figure 7.3.

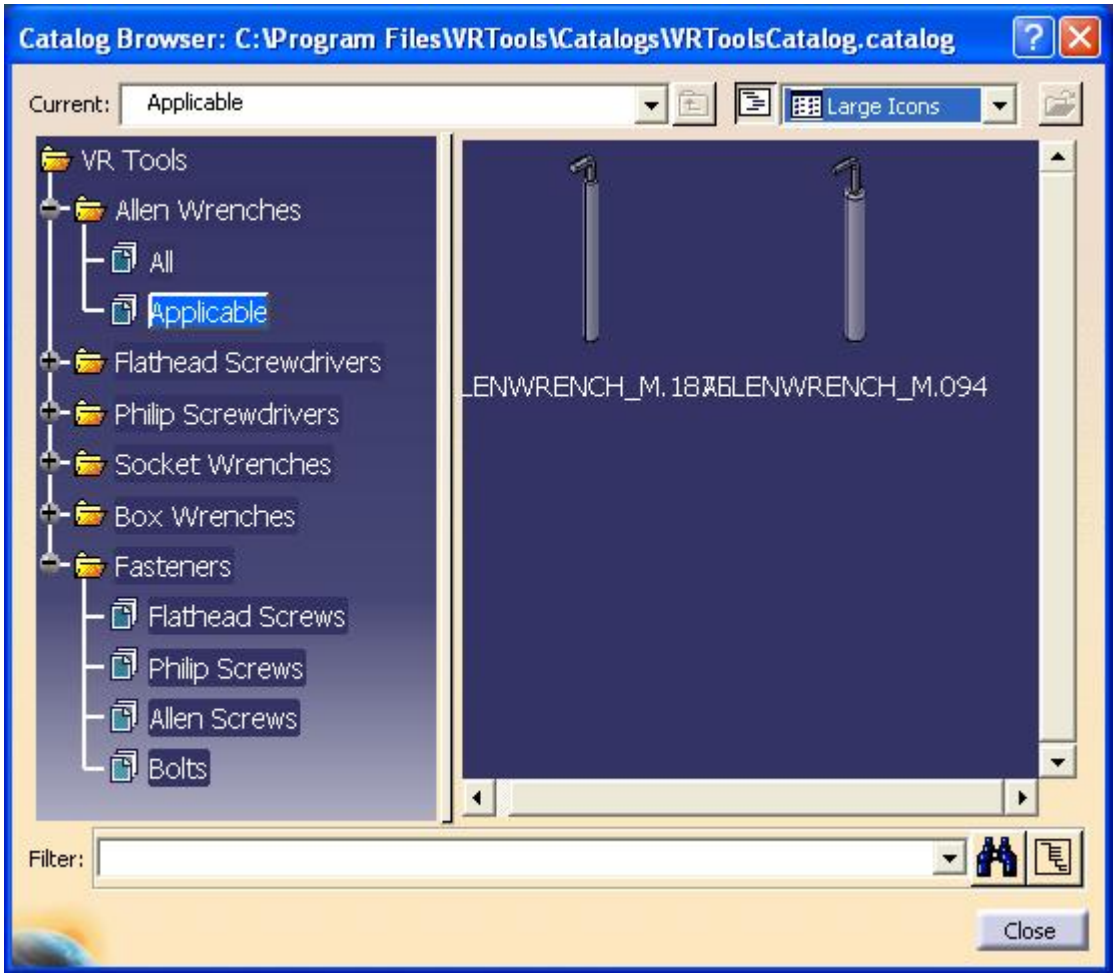


**Figure 7.3 Catalog preprocessing**

### Select Tools/Fasteners

This commands brings up the VRTools catalog and enables the user to add tools and fasteners from the catalog to the current model. The user can browse through the catalog tree structure and has access to all the information stored along with the models. Figure 7.4 shows the contents of Allen Screws family under the Fasteners chapter. Any catalog component can be added to the current model by double-clicking on it.

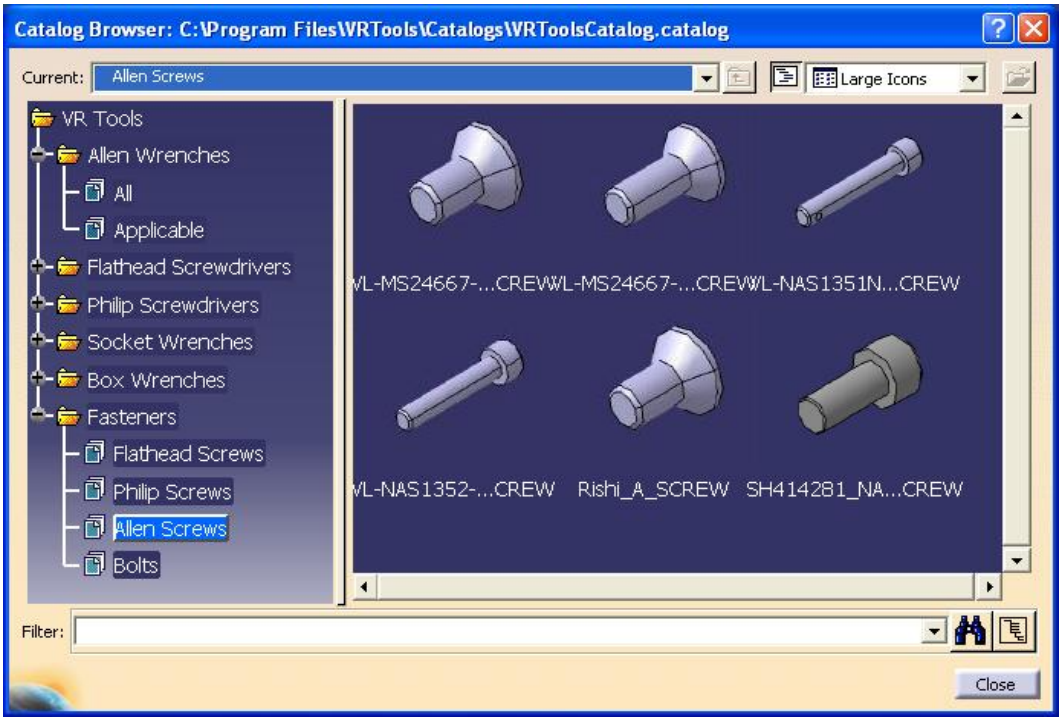


**Figure 7.4 Select Tools/Fasteners**

## Simulation

This command brings up the simulation setup and runtime dialog box. The user can select the haptic devices that will be used during the simulation. The hand-scaling factor can also be set on this dialog box. It also provides button for starting and stopping the simulation as shown in figure 7.5.


**Figure 7.5 Simulation Setup and Runtime**

## *Demonstration*

This section demonstrates a typical disassembly operation using a simple assembly. Figure 7.6 shows the assembly tree hierarchy for the demonstration model.

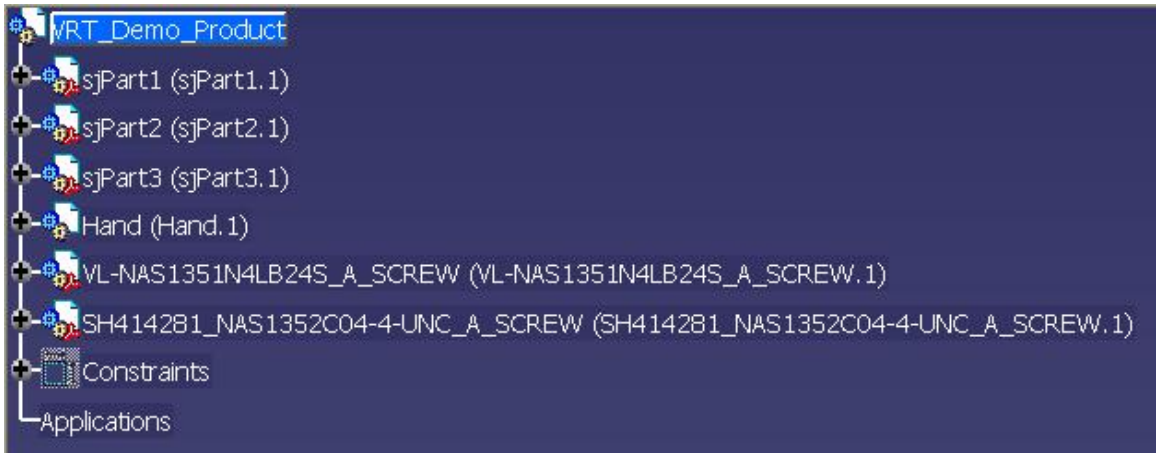


**Figure 7.6 Demonstration Model Assembly Structure**

The demonstration is depicted using a series of pictures taken during important stages of the simulation. Before starting the simulation, the component-fastener mapping has been defined for the simple 3-component assembly using the component-fastener mapping utility shown in figure 7.2. A compatible Allen wrench for one of the Allen screws is also imported from the catalog. Figures 7.7a through 7.7d walk through different steps of the simulation.

In figure 7.7a the user has grasped the tool and is able to move it freely. At this point none of the assembly components can be disassembled as all the fasteners mapped to them are in place.

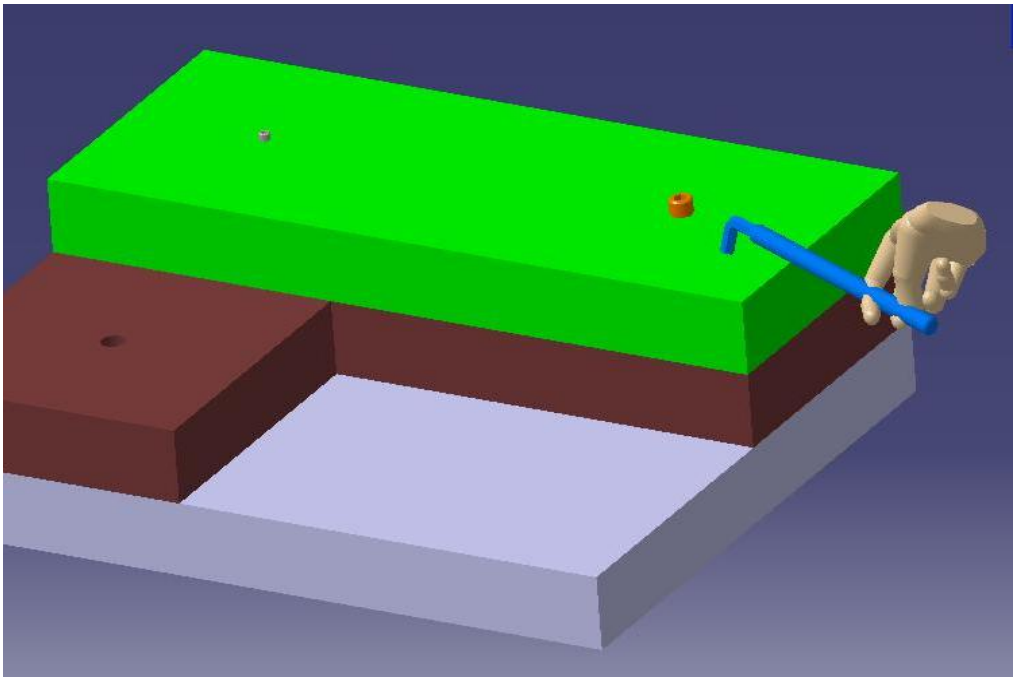

**Figure 7.7a Demonstration: Tool in hand**

In figure 7.7b the tool (allen wrench) is engaged with the fastener (allen screw) when it gets in a close to the fastener. At this point the user can rotate the tool to unscrew the fastener.

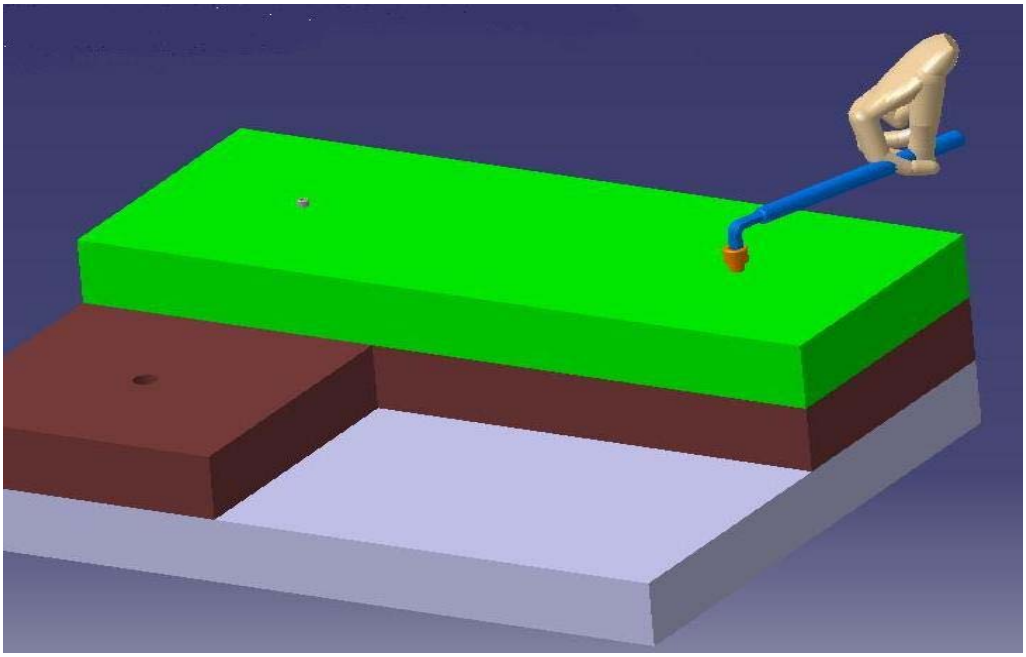

**Figure 7.7b Demonstration: Tool Engaged**

Figure 7.7c shows the fastener being removed after it is completely out of the hole. The user can now try to disassemble the components held down by this fastener.

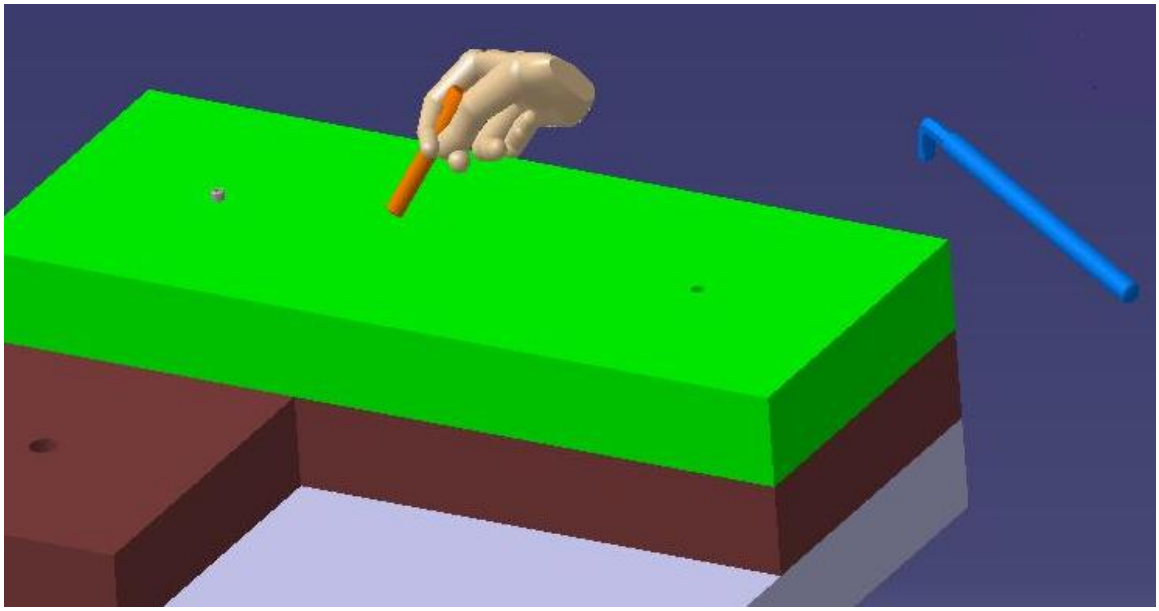

**Figure 7.7c Demonstration: Fastener Removed**

Figure 7.7d depicts the disassembly step where the component held down by the removed fastener can be displaced from its assembled location.

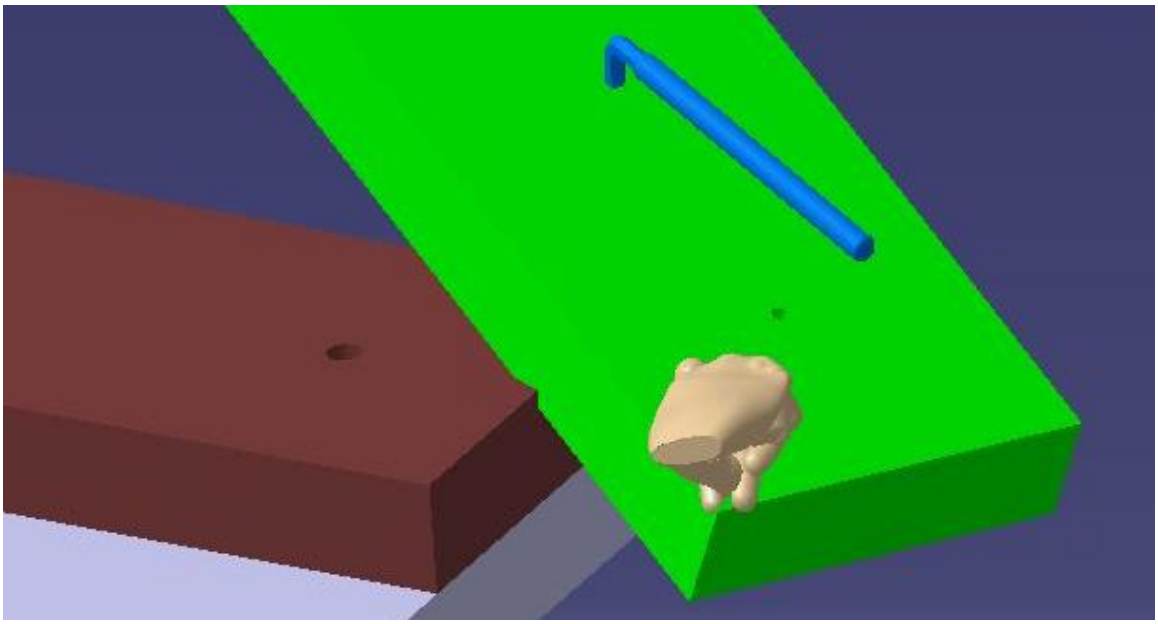

**Figure 7.7d Demonstration: Disassemble Component**

# CHAPTER 8

# SUMMARY AND FUTURE WORK

This chapter summarizes the goals achieved while developing the open architecture for implementing virtual assembly tools functionality. It also gives a brief account of development of a prototype for VR cutting tools functionality to demonstrate the flexibility of the proposed architecture. Finally it lists limitations of the current architecture, which also provides a direction for future development of virtual assembly tools functionality.

## *Summary*

In this research we proposed and demonstrated an open architecture for integrating CAD systems and Haptic devices while developing virtual assembly environments supporting virtual tools. The high level architecture proposed the CAD plugin and the GHBridge as the two main software modules. We further developed a detailed architecture and defined key classes and their roles for the CAD Plugin and the GHBridge. Thus, the proposed model facilitates building of immersive environments for performing virtual assembly operations within CAD systems using various haptic devices for user interaction. Some of the key features of the proposed architecture are as follows:

1. It converts the familiar CAD environment into virtual environment for assembly operations.
2. It leverages off of the CAD system's visualization capabilities.
3. It is extensible and can be applied to various types of mechanical tools.
4. It is flexible in facilitating usage of various haptic devices.

5. It provides a guideline for building and managing a library of virtual tools.

In order to demonstrate the extensibility of the proposed architecture we implemented a prototype for supporting virtual cutting tools called VRCut. The following section gives a brief description of the prototype implementation for cutting tools functionality.

## VRCut Prototype

The VRCut prototype is also developed for CATIA V5 and uses the CyberGlove from Immersion Corporation as the haptic device. In the Haptics Enabled Virtual Tools Environment, the user is able to grasp a cutting tool in a CATIA V5 environment with the virtual hand and perform a planar cut through the assembly. Thus, the intent is to allow the user to interactively split an assembly and manipulate the fragmented pieces. The module developed has the capability to be used for any CAD assembly, thus providing a very wide range of applications. The cutting tools supported by the current version of the software are a circular saw and a jig saw. The only preprocessing required for the simulation is that the cutting tool, with a certain naming convention, needs to be added at the top of the assembly that is being cut and manipulated.

The focus of this phase was on developing algorithms for visualizing and implementing assembly cut operations for CAD models. An extensive literature review led to the possibility of using voxels. A test-case cutting program was developed in C++ to further investigate the use of voxels. Since the algorithms had to be implemented for a CAD model, functionality supported by the CAD system (CATIA V5 in this case) was also taken into account. CATIA supports a variety of Boolean operation that can be used for emulating material removal operation. The "Remove" feature supported by the Assembly Workbench

in CATIA V5 enables the user to remove the intersecting geometry between two bodies. This prototype uses the "Remove" operation for simulating the cutting process. For any cut that results in two disjoint solids, the "Remove Lump" command from CATIA is used via a macro to create two separate bodies. These bodies can then be manipulated independent of each other. Figures 8.1 through 8.3 show various stages of the cutting operation.

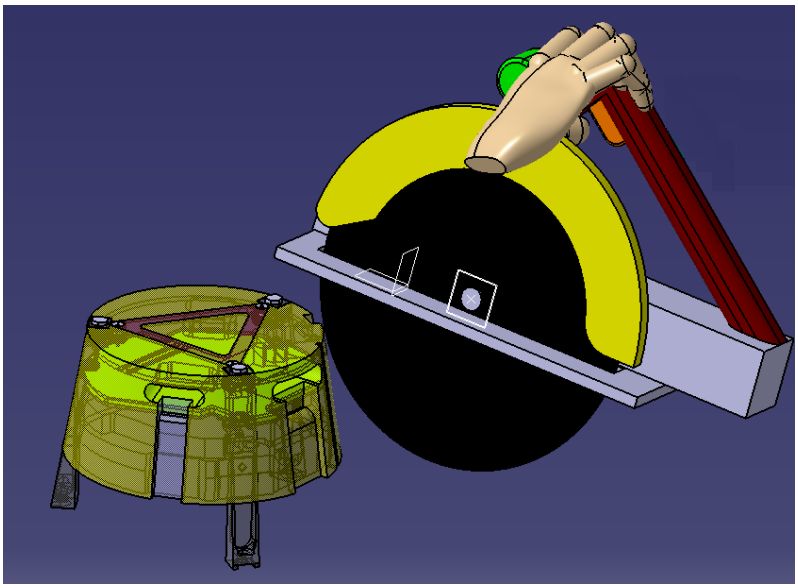

**Figure 8.1 Cutting Saw in Hand**

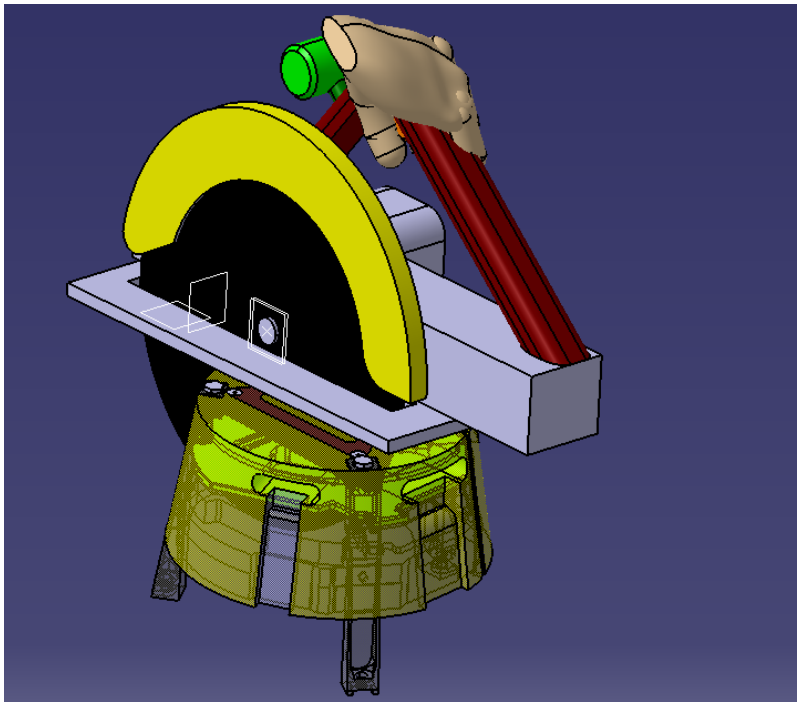

**Figure 8.2 Assembly Cut**

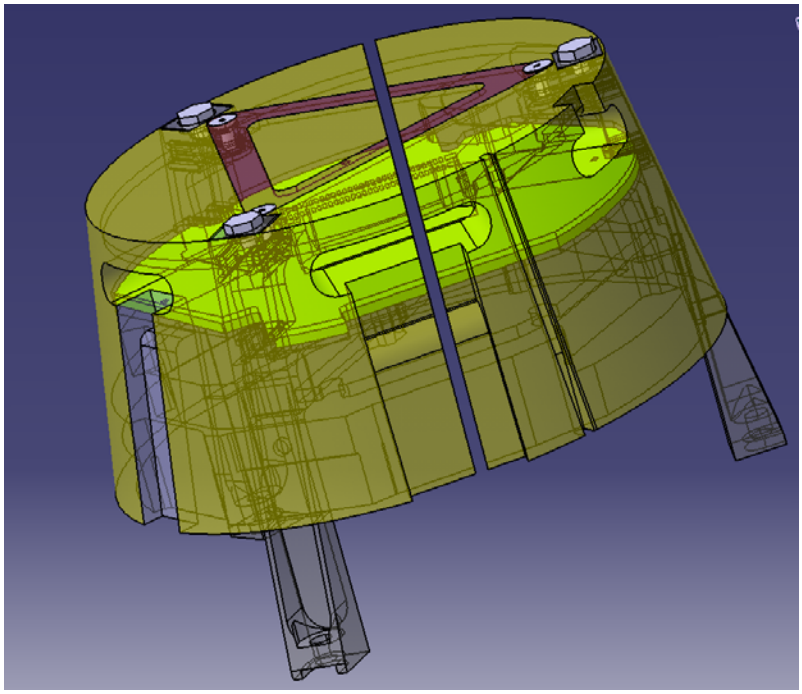

**Figure 8.3 Assembly cut into two halves**

## *Use of the Open Architecture in VRCut*

VRCut prototype adopts the open architecture proposed in this research. Except for the algorithms that were implemented to perform the cutting operations on the assemblies all the functionality from VR Tools implementation was used. The visualization scheme, scene graph management, constraints and state management were directly used without any changes. Thus we were able to demonstrate the extensibility of the proposed open architecture via implementation of the VRCut prototype.

## *Future Work*

While implementing the VRTools functionality and the VRCut prototype we came across some of the limitations of the architecture proposed in this work. These limitations can be overcome by further developing the architecture. Following are some of the limitations in the current architecture that can be addressed in future work:

- Current architecture for the CAD Plugin is designed to be tightly integrated with the CAD API. Implementing the VRTools functionality for a new CAD system will involve complete re-writing of the CAD Plugin code. This effort can be reduced to minimum by breaking down the CAD Plugin into a CAD-neutral component that has all the algorithms and logic and a utility library implementing the CAD API.
- Current architecture does not allow design modifications to the assembly components while the simulation is running. The user has to stop the simulation to make design changes to the parametric CAD model. The architecture can be further developed to expose the CAD parameters in the virtual environment and allow the user to change

them while running the simulation. The simulation environment can then be automatically updated once the CAD system regenerates the updated CAD model.

# REFERENCES


[1] Burdea, G., 2000, "Haptics Issues in Virtual Environments," *The 18th Computer Graphics International 'Humans and Nature' Conference,* Geneve, Switzerland.

[2] Zachmann, G., and Rettig, A., 2001, "Natural and Robust Interaction in Virtual Assembly Simulation," *8th ISPE International Conference on Concurrent Engineering: Research and Applications (ISPE/CE2001),* July, Anaheim, California, USA.

[3] Constantinescu, D., Salcudean, S. E., and Croft, E. A., 2005, "Local Model of Interaction for Haptic Manipulation of Rigid Virtual Worlds," The International Journal of Robotics Research, **24**(10), pp. 789-804.

[4] Regenbrecht, H., Hauber, J., Schoenfelder, R., and Maegerlein, A., 2005, "Virtual Reality Aided Assembly with Directional Vibro-Tactile Feedback," *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia,* Dunedin, New Zealand.

[5] Coutee, A. S., Mcdermott, S.D., and Bras, B., 2001, "A Haptic Assembly and Disassembly Simulation Environment and Associated Computation Load Optimization Techniques," ASME Journal of Computing and Information Science, **1**(2), pp. 113-122.

[6] McDermott, S., and Bras, B., 1997, "Development of a haptically enabled dis/re-assembly simulation environment," *Proceedings ASME Virtual Environment Systems Conference*, Las Vegas.

[7] Coutee, A. S., and Bras, B., 2004, "A Comparison of Two Collision Detection Libraries in a Haptic Simulation Environment," *Proceedings of DETC' 04, ASME 2004 Design*

*Engineering Technical Conferences and Computers and Information in Engineering Conference,* Salt Lake City, Utah.

[8] Zhu, Z., Gao, S., Wan, H., Luo, Y, and Yang, W., 2004, "Grasp Identification and Multi-Finger Haptic Feedback for Virtual Assembly," *Proceedings of DETC' 04, ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* Salt Lake City, Utah.

[9] Liu, X., Doods, G., Mccartney, J., and Hinds, B.K., 2004, "Design and Deformation of Cad Surface Models with Haptics," *Proceedings of DETC' 04, ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* Salt Lake City, Utah.

[10] Wan, H., Gao, S., Peng, Q., Dai, G., and Zhang, F., 2004, "Mivas: A Multi-Modal Immersive Virtual Assembly System," *Proceedings of DETC' 04, ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* Salt Lake City, Utah.

[11] Johnson, D. E., and Cohen, E, 2004, "Unified Distance Queries in a Heterogeneous Model Environment," *Proceedings of DETC' 04, ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* Salt Lake City, Utah.

[12] Vance, J., 2003, "Interactive Product Development in a Virtual Environment Utilizing Haptics," *2003 NSF Design, Service and Manufacture and Industrial Innovation Grantees and Research Conference Proceedings,* Birmingham, AL.

[13] Borro, D., Savall, J., Amundarain, A., Gil, J.J., Garcia-Alonso, A., and Matey, L., 2004, "A Large Haptic Device for Aircraft Engine Maintainability," Computer Graphics and Applications, IEEE, **24**(6), pp. 70-74.

[14] Ng, F. M., Ritchie, J. M., Simmons, J. E. L., and R. G. Dewar, 2000, "Designing Cable Harness Assemblies in Virtual Environments," Journal of Materials Processing Technology, **107**(1-3), pp. 37-43.

[15] Dewar, R. G., Ritchie, J.M., Carpenter, I.D., and Simmons, J.E.L., 1997, "Tools for Assembly in a Virtual Environment," *Proceedings of ICMA '97,* University of Hong Kong.

[16] Ritchie, J. M., Dewar, R.G., and Simmons, J.E.L., 1998, "Assembly Process Planning Using Immersive Virtual Reality – an Industrial Case Study," *Proceedings of PRASIC' 98 - Robotica,* University Transylvania of Brasov, Romania.

[17] Gupta, S. K., Paredis, C., and Sinha, R., 2001, "Intelligent Assembly Modeling and Simulation," Assembly Automation, **21**(3), pp. 215-235.

[18] Gomes de Sá, A., Jakob, U., and Zachmann, G., 1999, "Virtual Reality as a Tool for Verification of Assembly and Maintenance Processes".

[19] McDermott, S., and Bras, B., 1997, "Development of a haptically enabled dis/re-assembly simulation environment", Proceedings ASME Virtual Environment Systems Conference, Las Vegas, September 1997.

[20] Jayaram, S., Jayaram, U., Wang, Y., Tirumali, H., Lyons, K., and Hart, P., 1999, "Vade: A Virtual Assembly Design Environment," IEEE Computer Graphics and Applications, Virtual Reality, **19**(6), pp. 44-50.

[21] Wang, Y., Jayaram, U., Jayaram, S., and Shaikh, I., 2003, "Methods and Algorithms for Constraint Based Virtual Assembly," Virtual Reality, **6**, pp. 229-243.

[22] Jayaram, S., Vance, J., Gadh, R., Jayaram, U., and Srinivasan, H., 2001, "Assessment of VR Technology and Its Applications to Engineering Problems," ASME Journal of Computing and Information Sciences in Engineering, **1**, pp. 72-83.

[23] Jayaram, U., Jayaram, S., Dechenne, C., Kim, Y., Palmer, C., and Mitsui, T., 2004, "Case Studies Using Immersive Virtual Assembly in Industry," *Proceedings of DETC' 04, ASME 2004 Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* Salt Lake City, Utah.

[24] 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, Accepted for publication.

[25] 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,* Washington D.C.

[26] 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 Limb," *ASME Proceedings of DETC2003,* Chicago, IL.

[27] Jayaram, U., Tirumali, H., Jayaram, S., and Lyons, K., 2000, "A Tool/Part/Human Interaction Model for Assembly in Virtual Environments," *Proceedings of DETC' 00,*

*ASME 2000 Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* Baltimore, Maryland.

[28] Gurocak, H., Jayaram, S., Parrish, B., and Jayaram, U., 2003, "Weight Sensation in Virtual Environments Using a Haptic Device with Air Jets," ASME Journal of Computing and Information Science, **3**(2), pp. 130-135.

[29] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcug98/html/_asug_home_page.3a_.spy.2b2b.asp