
A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Paul M McIntosh
MSc. Internetworking

School of Computer Science and Information Technology
College of Science, Engineering and Health
RMIT University

September 2009
Declaration

I certify that except where due acknowledgement has been made, the work is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program; any editorial work, paid or unpaid, carried out by a third party is acknowledged, and ethics procedures and guidelines have been followed.

Paul M McIntosh
School of Computer Science and Information Technology,
RMIT University,
Melbourne, Victoria, Australia.

4th September 2009
Acknowledgments

I would like to thank my supervisors Dr. Margaret Hamilton and Dr. Ron van Schyndel for the help, encouragement and constructive criticism that they have provided to me throughout this thesis. I would also like to thank my wife Laura and son Max for their support. I would especially like to thank RMIT for funding my scholarship and enabling me to complete the PhD full-time. There have been many people that have also provided input at various times, and I would like to extend my thanks to the following people for taking time out of their busy days to make this study possible: Dr. Ghassan Al-Qaimari, Prof. Heinz Schmidt, Dr. Ian Peake, Jens von Pilgrim, Mark Hermeling, Tim Coddington, Bran Selic, Grady Booch, Prof. Wilhelm Schäfer, Stefan Henkler, Jin Qian, Garry Keltie, Matthew Ovens and the numerous anonymous research participants.
Credits

Portions of the material in this thesis have previously appeared in the following publications:


- Supporting material is available at the URL: [www.x3d-uml.org](http://www.x3d-uml.org)

---

1 The paper “X3D-UML: 3D UML State Machine Diagrams” was also presented at the RMIT CRSC’08 Computer Science and I.T. Research Students Conference and won the best paper award.
Contents

Declaration ................................................................................................................................... i
Acknowledgments .......................................................................................................................... ii
Credits ............................................................................................................................................. iii
Contents .......................................................................................................................................... iv
List of Figures ................................................................................................................................. vi
List of Tables ................................................................................................................................... x
Glossary ........................................................................................................................................... xi
Abstract .......................................................................................................................................... xii

Chapter 1 – Introduction .................................................................................................................... 1
  1.1 Overview .................................................................................................................................. 1
  1.2 Technology ............................................................................................................................... 2
  1.3 Research Contributions and Thesis Structure .......................................................................... 10
  1.4 Summary .................................................................................................................................. 13

Chapter 2 - 3D, UML and Evaluation ............................................................................................... 14
  2.1 Overview .................................................................................................................................. 14
  2.2 Definition of 3D UML ................................................................................................................ 15
  2.3 Related Research ...................................................................................................................... 16
  2.4 Evaluation Techniques ............................................................................................................. 31
  2.5 3D Technologies ...................................................................................................................... 36
  2.6 Summary .................................................................................................................................. 48

Chapter 3 - Implementing X3D-UML ............................................................................................... 49
  3.1 Overview .................................................................................................................................. 49
  3.2 X3D-UML Library ..................................................................................................................... 50
  3.3 “Compilable” Visualisations ..................................................................................................... 59
  3.4 XSLT from JavaML to X3D-UML ............................................................................................. 61
  3.5 Summary .................................................................................................................................. 67

Chapter 4 - 3D UML Sequential Evaluation Methodology ............................................................... 68
  4.1 Overview .................................................................................................................................. 68
  4.2 3D UML Sequential Evaluation in Theory .............................................................................. 69
  4.3 3D UML Sequential Evaluation in Practice ............................................................................ 77
  4.4 User Task Analysis .................................................................................................................... 78
  4.5 Heuristic Evaluation ................................................................................................................ 81
| 4.6 | Heuristic Evaluation Workshop .................................................. | 82 |
| 4.7 | Formative Evaluation ................................................................. | 111 |
| 4.8 | Summative Evaluation ................................................................. | 114 |
| 4.9 | Summary .................................................................................. | 117 |
| Chapter 5 - 3D UML State Machine Diagrams ........................................... | 119 |
| 5.1 | Overview .......................................................................................... | 119 |
| 5.2 | 2D State Machine Diagrams .............................................................. | 120 |
| 5.3 | User Task Analysis ............................................................................ | 124 |
| 5.4 | Heuristic Evaluation ........................................................................ | 137 |
| 5.5 | Formative Evaluation ....................................................................... | 141 |
| 5.6 | Summative Evaluation ..................................................................... | 148 |
| 5.7 | Summary ............................................................................................ | 150 |
| Chapter 6 - 3D UML Mechatronic and Holistic Diagrams ............................... | 151 |
| 6.1 | Overview .......................................................................................... | 151 |
| 6.2 | 3D UML Mechatronic Diagrams .......................................................... | 151 |
| 6.3 | 3D UML Holistic Diagrams ............................................................... | 164 |
| 6.4 | Summary ............................................................................................ | 173 |
| Chapter 7 – Conclusions, Validity and Future Research .................................. | 175 |
| 7.1 | Overview .......................................................................................... | 175 |
| 7.2 | Conclusions ....................................................................................... | 177 |
| 7.3 | Threats to Validity .......................................................................... | 181 |
| 7.4 | Future Research .............................................................................. | 187 |
| 7.5 | Closing Remarks ............................................................................. | 189 |
| Appendix A – UML User Questionnaire .......................................................... | 191 |
| Appendix B - Heuristic Evaluation Sheet ....................................................... | 194 |
| Background ............................................................................................. | 194 |
| User Tasks ............................................................................................. | 195 |
| 3D UML State Machine Diagrams and Event Notation ..................................... | 196 |
| Evaluation Section .................................................................................. | 198 |
| Bibliography ........................................................................................... | 201 |
List of Figures

Figure 1-1 – An example of a UML visualisation. This is a simple class diagram that should be recognisable to most software engineers. ..................5

Figure 1-2 – 3D UML Mechatronic Diagrams can be defined as the integrated mechanical, electronic and software visual models of the system with a focus on the UML perspective of the system. .................................12

Figure 2-1 – Contract Box. An example of a 3D diagram from “Three Dimensional Software Modelling”, Proceedings of the 20th international conference on Software engineering, ©1998, Joseph Gil and Stuart Kent.[34] .................................................................19

Figure 2-2 – Example of 3D UML using force directed layout[26], Australian symposium on Information visualisation. © 2001 Tim Dwyer..........20

Figure 2-3 – GEF3D based editor visualising transformation chains. From “Gef3D: a framework for two-, two-and-a-half-, and three-dimensional graphical editors”[50], © 2008, Jens von Pilgrim.................................22

Figure 2-4 – “Heating State” shows the current system state through the red sphere and green arrow state machine diagram. From “The 3D behavioural model design for simulation and software engineering” © 2000 Fishwick. .................................................................23

Figure 3-1 – Example of the Diagram Interchange class structure, generated from XMI and extended for use as a 3D UML diagram. “View” classes provide the UML graphical representation and the Point class shows a 3D extension using the new “z” field.........................53

Figure 3-2 – X3D-UML OpenSceneGraph implementation. This implementation consists of a specialised viewer, which utilises a Diagram Interchange structure for storing and viewing UML.........................56

Figure 3-3 – Diagram structure within osgX3D UML referring to elements by name.................................................................59

Figure 3-4 – Transforming Java to JavaML to X3D-UML...............................63

Figure 3-5 – FirstApplet.x3d displayed through an X3D browser demonstrating the display quality of class visualisation.................................64

Figure 3-6 – Classes from the Java 3D packages shown as UML through an X3D browser demonstrating the display quality of class visualisation. Classes are grouped in planes based on the package (directory
Figure 4-1 – Sequential Evaluation methodology as we have refined it for researching advanced 3D UML visualisation through X3D

Figure 4-2 - Example of chromacoding of C# source code in Visual Studio. In this example comments appear in green text, strings in red, operators and expressions in black and keywords in dark blue.

Figure 4-3 – RoseRT “Traffic Lights” model. The tree on the left shows the “navigator” hierarchy of states within the state machine. The state machine diagram on the right shows the top level. Both “initializing” and “working” are composite states containing substates diagrams, as can be seen by the icon bottom right and the child states in the “navigator”

Figure 4-4 – Initial UML class model/diagram

Figure 4-5 – Second diagram, UML class model with IDs

Figure 4-6 – Third diagram, UML class model with getters and setters

Figure 4-7 – Last diagram, ER model

Figure 4-8 – Question card for heuristic evaluation. Workshop participants were asked to answer questions like this for each guideline.

Figure 4-9 – Heuristic Evaluation sheet. Workshop participants were asked to evaluate 3D UML diagrams using this form.

Figure 4-10 – Group 1 State Machine Diagram Layout. This shows the realisation of the 3 heuristic guidelines.

Figure 4-11 – Proposed 3D UML state machine diagram layout, with connection cones of light

Figure 4-12 – Layout Group “2D” solution to meeting the overview heuristic.

Figure 4-13 – Example of problems that unpredictable user environments produce. This shows a corrupt 3D UML visualisation purely due to the computer’s regional settings.

Figure 4-14 – Summative Results 2D Vs 3D

Figure 5-1 – RoseRT example of a state machine with substates displayed in different window tabs. The ‘green’ state contains an icon indicating that substates exist. This example if from a training model provided with RoseRT called the “TrafficLights” model.
Figure 5-2 – An example of a non-layered approach to describing a state machine. This diagram represents the RoseRT “TrafficLights” state machine presented Figure 5-1 recreated with IBM Rational Software Modeller. Instead of each state machine layer being displayed in a separate tab, the entire state machine is presented in a single diagram..........................122

Figure 5-3 – Our prototype 3D State Machine Diagram displayed in X3D. With this diagram it is possible to rotate the entire diagram to different views and zoom in on any part of the diagram.........................123

Figure 5-4 – Hierarchical state machine diagram from RoseRT documentation .............................................................................................................133

Figure 5-5 – In the current situation the engineer is only able to view one state machine diagram at a time. In considering how a single event may be processed by the state machine, they must think about all superstates and how they may handle that event.........................135

Figure 5-6 – With a 3D state machine diagram the engineer is able to view all state machine diagrams in the hierarchy and only needs to think about how each may handle that event. ........................................135

Figure 5-7 – With combined 3D and event notation the engineer need only think about the events that have impact on specific states levels, irrelevant diagrams can be ignored. ..........................................................135

Figure 5-8 – 3D UML state machine diagram derived from user task analysis. The state machine diagram presents all diagrams in a hierarchy in a single 3D dynamic view and provides additional information about the events that drive the state machine behaviour...............................138

Figure 5-9 – Refined 3D UML state machine diagram example. The “working” branch is laid out for easy viewing and the “initializing” branch, which is not of interest, is "shrunk" (top right). Event notation is shown as a summary at the bottom of applicable state machine.........................143

Figure 5-10 – A complete 3D UML state machine diagram generated from actual user data. This shows the main state machine for a subsystem. This particular state machine is owned by User 1 who, prior to this visualisation, had to “stare into space for 15 minutes at a time” to imagine relationships between diagrams. The diagram is interactive and allows the user to zoom in to areas of interest from this perspective. ..........................................................................................145

Figure 6-1 – This shows a state machine diagram overlaid on an embedded processor.................................................................153
Figure 6-2 – The original RCX Rover main robot state machine diagram. This was designed for a robot which could sense left and right presses of a sensor .......................................................... 157

Figure 6-3 – New NXT “HeadBanger” state machines based on the original RCX “Rover”. This is the redesigned logic of the state machine to work with a robot with only a single sensor. The top state machine represents the new single sensor state machine ........................................ 158

Figure 6-4 – The system for animated 3D UML state machine diagrams .......... 159

Figure 6-5 – 3D UML Mechatronic Diagram. The view of the robot and diagrams that developer sees through the computer screen in debug mode. 161

Figure 6-6 – Example of animated 3D UML Mechatronic Diagrams in action. The current state in each diagram is highlighted with red. The left frame shows the states of the robot in “forward” state and the sensor in “free” state. The right frame shows the robot as it touches the wall, with a state transition to “WaitingForReverse” state in reaction to the sensors state transition to “Pressed” state ........................................ 162

Figure 7-1 - Summative Evaluation of 2D Vs 3D .................................................. 177
List of Tables

Table 2-1 – X3D-UML Design Constraints.................................................................29

Table 2-2 – Summary of 3D Technology Issues. For convenience purposes, this table provides a summary of the blocking issues stated above to using the technology for this research. ........................................47

Table 4-1 – Refined Heuristics for State Machine Diagrams ...............................101

Table 4-2 – Evaluators Assessment of the X3D-UML 3D state machine diagrams ........................................................................................................101

Table 4-3 – Refined Heuristics for 3D UML Translation Chains.........................103

Table 4-4 – Heuristics Ratings. State Machine (SM) and Translation Chain (TC) Groups. Evaluation groups (Eval.) rated heuristics 1-5, with 5 being “essential”. Layout groups picked the 3 most important heuristics to demonstrate with a diagram layout........................................107

Table 4-5 – Comparison between Evaluators’ and Users’ Responses (detailed in Section 5.5.3 On-Site Formative Session with references to user feedback items) ................................................................................109

Table 5-1 – Substate survey of commercial RoseRT models, showing extensive substate usage. The results are from four industries and have been generalised to areas of “office equipment”, “robotics control”, “systems control” and “networking equipment”. ........................................125

Table 5-2 – RoseRT User Profile. Results of closed questions from Appendix A for RoseRT Users.................................................................128

Table 5-3 – Metrics for 4 models from industries surveyed in Table 5-1 and the user task study subsystem. Off-screen diagrams is a measure of the average number of diagrams that need to be considered for any given diagram within a hierarchical state machine....................137

Table 6-1 – Results of closed questions for Text Editor + Diagram Tool Combination Users. ......................................................................................167
## Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>Two Dimensional</td>
</tr>
<tr>
<td>3D</td>
<td>Three Dimensional</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CAD</td>
<td>Computer Aided Design</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided Software Engineering</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DI</td>
<td>UML Diagram Interchange Specification</td>
</tr>
<tr>
<td>GEF</td>
<td>Graphical Editor Framework</td>
</tr>
<tr>
<td>HCI</td>
<td>Human Computer Interaction</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>JavaML</td>
<td>Java Markup Language</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SVG</td>
<td>Scaled Vector Graphics</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>VRML</td>
<td>Virtual Reality Modelling Language</td>
</tr>
<tr>
<td>Web3D</td>
<td>Web based Three Dimensional Graphics</td>
</tr>
<tr>
<td>X3D</td>
<td>eXtensible 3D</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>Extensible Stylesheet Language Transforms</td>
</tr>
</tbody>
</table>
Abstract

This thesis presents an in-depth investigation into the practical use of 3D for software visualisation. This work presents the first comprehensive user-centred study which examines the software engineering tasks users undertake currently, the issues that 3D addresses and a measure of benefit of the 3D solution compared to traditional approaches. This thesis also presents a mechanism for creating 3D software visualisations, a refined evaluation methodology and visualisation heuristics that together provide a valuable resource for further research into this area.

The research results have been structured so they are directly applicable to industry and as such are already undergoing industrial adoption. This has been achieved through the following:

Firstly the research augments current and accepted software visualisation approaches by basing the visual notation on the Unified Modelling Language (UML). This has enabled the current visual software engineering tasks to be studied and for representative user tasks to be captured and quantified. The 3D visualisations then complement the current working practices by solving "real world" issues that are experienced using 2D visualisation approaches. These tasks are captured from both software development professionals and students working on software development projects.

Secondly the research is based on open standards and open source software. Our implementations are compatible with the X3D (eXtensible 3D) ISO standard and allow visualisations to be created and shared across X3D viewers. Further to this, as a result of user needs uncovered by this study, a specialised 3D UML viewer has been created based on OpenSceneGraph. These visualisation techniques have been defined, created and tested to work on standard desktop computers and integrated with software engineering tools currently utilised by the software engineering tasks.
Based on X3D, UML, actual tasks and data, with existing computers and tools, this thesis demonstrates that there is clear and measurable benefit in 3D UML software visualisation for industry. However, rather than explicitly stating simply that “3D for software visualisation is good”, outlined is a repeatable and structured approach to developing and evaluating 3D UML interfaces. The results demonstrate that through correct implementation positive aspects of 3D can be leveraged and negative aspects minimised.

The main focus of this study has been in the area of 3D UML state machine diagrams, also known as statecharts. These have been investigated in two main areas. Firstly they have been evaluated as an extension to IBM Rational Rose Technical Developer, also known as RoseRT, and the use of this extension evaluated in industry. Secondly they have been evaluated as an augmented reality extension, coined a “3D UML Mechatronic Diagram”, and the use evaluated against a student Lego robot project.

Although the focus of the study has been on 3D UML state machine diagrams, the methodology described is intended to be applicable to all UML diagrams and similar visual notations. In addition, the methodology adds value to researching 2D improvements to UML diagrams, as the use of 3D has been demonstrated as an effective framework for analysing problems with current diagrams.
Chapter 1 – Introduction

Graphics have been used for thousands of years to relay abstract information. Australian Aboriginal art rock painting of Mimi spirits in the Anbangbang gallery at Nourlangie in Kakadu National Park. Photo by Dustin M. Ramsey (Krallizecl) ©2002²

1.1 Overview

The research presented in this thesis evaluates the use of 3D (Three Dimensional) visualisation as a means of aiding a software engineer to more easily understand the system that they are working on. The concepts outlined in this thesis take into account that much effort has already gone into techniques for helping engineers view software problems more intuitively. The research builds on existing software development infrastructure, and builds on tools and knowledge already at the software engineer’s disposal.

The results of this research are intended to be immediately applicable to industry. To achieve this, our results are not based on “high-end” and expensive graphics workstations, but instead are based on “mid-range” desktop pc’s (personal computers) that most software engineers use (and are also found in the home). Depending on the
software development being undertaken the average computer will have a typical operating system of Microsoft Windows or some variant of Linux. Also depending on the type of development being undertaken, it will have a typical graphics card, which would easily cope with most 3D games available at the time it was purchased. It is this underutilised 3D capability that this research taps into, to investigate its benefit for software engineering.

In the very near future (if not already) the average computer, that an average software engineer uses to develop software, will arguably have more processing power than the brain of the engineer using it [63]. Regardless of whether this “brain power” estimate is correct, the trend clearly indicates that computer processing power is increasing at a much faster rate than human processing power. For us to keep pace with and control of the technology that we develop we must redirect a larger percentage of the computers power to present us with the problem in a way that our brains will more readily comprehend. To cope with this comprehension “bottleneck” in the software development process, we must make more effort to consider human factors in software engineering. As one usability researcher stated ‘What’s the most important operating system? …its Homo sapiens version 1.0. It shipped about one hundred thousand years ago, there is no upgrade in sight but it’s the one that runs everything’ [41].

In this chapter we present a high level view of the background to the research. Presented is the research context in terms of the technology and historical limitations that may have hindered the use of 3D software visualisation. Also presented is the proposed solution to address these limitations and the subsequent research contributions contained in this thesis.

1.2 Technology

This section presents a brief overview of some historical reasons as to why 3D software visualisation has not been adopted previously. We then present the technologies utilised in this research (UML and X3D) and how the integration of these technologies address past limitations and add further possible value.

---

2 This image is licensed under the Creative Commons Attribution ShareAlike 2.5. Image sourced from Wikipedia at http://en.wikipedia.org/wiki/File:Anbangbang_gallery_Mimi_rock_art_cropped.jpg.
1.2.1 Historical 3D Software Visualisation Limitations

This section lists some of the limitations, which have contributed to 3D visualisation not being adopted. These are not conclusive but do give an outline of how the concepts of this research have evolved. The issues have arisen through personal informal observation of the area over 15 years in industry prior to undertaking this research. These observations are mainly derived from working as a software engineer and through 5 years of supporting software engineers using UML tools at Rational Software. The issues can be summarised to a) computer graphics requirements; b) the need to evolve from and integrate into existing work practices; c) the need for standards; and d) the cost involved in creating 3D visualisations especially in relation to e) changing software.

a) **Computer Graphics** – In the past computer graphics capabilities have been a major limitation to the presentation of 3D software visualisations. Since any form of 3D graphics is limited by current hardware capabilities, it is apparent that these hardware restrictions have also applied to software visualisation. Further to this, any visualisation is most valuable at the point where it is most needed, on the desktop of a software engineer, where it can provide an extension to the current workflow. Unless the visualisation can work within a standard software engineer's environment, which typically has no specialised graphics capability, then there can be little or no benefit to the majority of engineers.

b) **Evolution** - One possible reason given for lack of 3D use is that software engineering is an evolving discipline. For new concepts to be adopted they need to evolve from existing accepted practices and tools. Software engineers, like rock climbers, ensure firm footholds in what is around them before reaching for the next new thing. As 3D is still a new concept, it also needs to be introduced without the need for software engineers to completely let go of traditional practices (such as the “all powerful” command line interface). Even if we ignore human factors, without evolution, we cannot leverage existing tools, which add value and also evolve. As in the example of HTML adoption, everyday text editors enabled basic text-based editing to first occur and from there evolved specialised HTML editors.

c) **Open Standards** – Another possible reason is that there are no open standards in the area of 3D software visualisation, and therefore 3D software visualisation
cannot be readily produced and shared. This is further complicated by the fact that standards generally are created in response to a need, so such standards will not evolve unless 3D visualisation is first used. The issue of openness is also important as far as the ability of software engineers to trust the technology and alleviate fears of being tied to a proprietary system, which is out of their control.

d) **Perceived Return on Investment** – Advances in software engineering which require a large amount of effort, are generally balanced against perceived return on investment (either in time or money). 3D requires a large amount of effort, Walsh and Bourges-Se venier state ‘*the field of computer science that deals expressly with creating, manipulating and navigating computer content in three dimensions – is difficult. Extremely difficult*’ [95]. The difficulties in creating 3D software visualisations greatly hinder exploration into its benefits. Furthermore, without initial empirical evidence of benefits, it is difficult to justify research effort or industrial use.

e) **Software Changes** – A software system is ever changing during the software development life cycle. If effort is to be expended in creating a software visualisation, that visualisation has to also adapt to changes in the software. This adds complexity to an already complex process of producing the initial 3D visualisation.

Although the reasons above are pure conjecture, Diehl also summarised similar ideas when referring to the future directions of software visualisation research, ‘*Software visualization will be doomed to stay an academic endeavor, if we do not succeed to integrate it into working environments and thus into the work flow of programmers, designers and project managers. To facilitate such integration existing standards must be adopted or extended...*’ [23]. The next sections describe two existing standards and how they may be adopted and extended to integrate into the working environments of software engineers.

### 1.2.2 UML

The UML (Unified Modelling Language) is a trademark of the OMG (Object Management Group, Inc [71]). The UML is described by Booch et al. as ‘*a graphical standard for*
visualizing, specifying, constructing, and documenting the artefacts of a software-intensive system’ [13]. The research described in this thesis focuses on the visual notation aspect of the UML.

The UML effort started in October 1994 and the first specification 1.1 accepted in November 1997 [13]. Since then the UML has experienced commercial acceptance and, as such, the visual notation is well recognised by software engineers from industry. Given the example in Figure 1-1, software engineers with knowledge of UML should be able to understand and create code reflected in the diagram given.

![Class Diagram](image)

*Figure 1-1 – An example of a UML visualisation. This is a simple class diagram that should be recognisable to most software engineers.*

An indication of the success of UML can be seen in the proliferation of UML tools used by industry, with Objects by Design listing more than 100 [72]. The number and variety of these tools also provides evidence of the flexibility of the UML to be used in different situations through the use of such tools.

### 1.2.3 X3D

Extensible 3D (X3D) is a trademark of the Web3D Consortium [98] and is the next generation of VRML (Virtual Reality Modelling Language). The Web3D Consortium [97] defines X3D as ‘a software standard for defining interactive web- and broadcast-based 3D content integrated with multimedia. X3D is intended for use on a variety of hardware devices and in a broad range of application areas such as engineering and scientific visualization, multimedia presentations, entertainment and educational titles, web pages,
and shared virtual worlds. X3D is also intended to be a universal interchange format for integrated 3D graphics and multimedia."

VRML is now more than 10 years old, being conceived at the first international World Wide Web Conference in 1994 [10]. Since then the specification has been continually refined and in 2004 the Web3D Consortium announced that the X3D specification was approved by the International Standards Organization (ISO) for formal publication.

Compared to UML, the success of VRML/X3D is less easily quantified. Despite more than 10 years of maturity the VRML/X3D technology still struggles to deliver to expectations. VRML co-inventor Mark Pesce [76] noted that even now potentially useful projects are ‘hamstrung at a fundamental level because the basic VRML “player” … hasn’t matured much (to be brutally honest, at all) in the last decade’. Despite this, the initial enthusiasm for VRML means that many tools today still export/import 3D models in this format and as such can easily be reused in the context of X3D.

1.2.4 Addressing the Limitations

The research outlined in this thesis investigates a solution that addresses all the limitations listed in Section 1.2.1. Through technological advances and the use of X3D and UML the historical limitations are shown to be no longer valid and 3D software visualisation, in the context of industry, can be researched effectively.

a) Computer Graphics - The performance of computer graphics has improved dramatically. The popularity of computer games has meant that the average computer, as used by software engineers, now possesses graphics systems which appear capable of rendering the types of visualisation required for 3D software visualisation. The most important feature required is the ability to render graphics that are primarily designed to meet the needs of software visualisation, rather than simply designed around the limitations of computer graphics. Due to advances in computer graphics it is now possible to tailor a visualisation to suit a task and then measure the benefit of the visualisation.

b) Evolution - The Unified Modelling Language (UML) has become an accepted visual notation. This means that current visualisations are understood by engineers and 3D extended visualisations based on UML should also be
understood with no specialised training. By basing visualisations on UML and extending them with 3D, the visualisations become an evolutionary step and it is possible to more effectively measure the benefit of 3D.

c) **Open Standards** – By basing the 3D software visualisation on the open standard eXtensible 3D (X3D) and the Unified Modelling Language (UML), visualisations meet standards that in theory can work across different tools (depending on the quality of the specification and implementation). This enables users’ artefacts to be imported/exported across tools and platforms and provides a greater variety of tools for different tasks. For example, a variety of X3D viewers exist for different OS platforms.

d) **Perceived Return on Investment** – As visualisations are an extension to existing environments, empirical evidence as to the benefits of 3D visualisation can be more easily obtained by directly comparing existing workflows to new “3D assisted” workflows.

e) **Software Changes** – As the visualisation is an extension, it can be tested within a software engineer’s environment. The visualisation is tied to the existing software infrastructure and therefore closely tied to the changing software system under development.

### 1.2.5 X3D-UML

In the previous sections the past limitations hindering 3D software visualisation have been discussed along with how these limitations can be overcome by combining X3D and UML (or have been overcome with improvements in computer graphics capability). In this section X3D-UML is presented as the combination of visualisation standards, this combination provides features above and beyond purely overcoming the limitations.

The features of X3D for 3D web content are apparent. With respect to software visualisation, some of the features are outlined below. This list, derived from the feature list defined in the Web3D Consortium X3D specification [97], demonstrates that X3D captures functionality far beyond just 3D. X3D provides a feature rich modelling environment which, when combined with UML, should enable advanced software visualisation in many areas. These features can be viewed from both a development
perspective, to aid in understanding of a system from a design point of view, and a runtime perspective, to aid in the understanding of a system as it executes.

The features of integrating X3D and UML are:

a) **3D graphics** – X3D provides a rich set of 3D modelling features, designed for creating simulated worlds but which could also be applied to software visualisation. Application of 3D can be as simple as adding depth to a UML diagram or as complex as modelling the software system, its mechanical interfaces and the context environment (e.g. modelling an air traffic control system, the control panel, the airport, aeroplanes and the users of the system).

b) **2D graphics** – As most current software development involves 2D, such as text editors, manuals and computer screens, some of these aspects can be directly translated and displayed as 2D in a 3D world. Similarly standard 2D UML diagrams could be directly rendered.

c) **Animation** – The ability to move objects in real-time allow features such as automatic “force directed” layout of diagrams, which Dwyer [26] has shown to provide some qualitative benefit for understanding complex software system architecture. Other examples of use could be a form of visual debugging similar to that proposed by Jacobs and Musial [49], or a means of modelling environment behaviour (as with the air traffic control example). Animated UML, using VRML, has already been demonstrated by Thaden and Steimann [86] as a means of intuitively teaching object-oriented program execution.

d) **Spatialized audio and video** – Video could be applied to projecting actual output onto a 3D UML stereotype of an “interface” class, to depict the computer screen runtime object. Audio could be applied in terms of data sonification. For example, audio sounds could be applied to class objects in digital telephone exchange software, allowing telephone technicians to detect faulty call switching by sound, in the same way they did in the past for mechanical telephone exchanges.

e) **User interaction** – This can be utilized to allow the user to manipulate the software modelled during development, in real-time. If modelling a complete
runtime environment, user interaction can be used to model the interfaces to simulate the software systems before they exist.

f) **User-defined objects** – These allow the definition of new objects through a prototyping mechanism, which in turn allows a library of UML objects to be created. Because an object's interface gives a point of abstraction, it is then possible to experiment with different visualisations based on the same information.

g) **Scripting** – This feature allows intelligence to be added to nodes and views. Furthermore, scripting will allow integration with existing software development tools such as compilers, debuggers and existing UML tools.

h) **Networking** – X3D is designed to enable shared components (i.e. user created nodes and content) and worlds (the combination of the components into a view). This feature should allow software engineers to collaborate on software by providing a network based collaboration environment of shared work.

i) **Physical simulation** – X3D provides features such as Humanoid animation which may be valuable in (HCI) Human Computer Interaction definitions of a system. For example a UML "Use Case" may specify the way a user interacts with a system through a sequence diagram, such as a button press to initiate some system behaviour. Through humanoid animation, the same specification may be able to be enhanced by including the physical aspects and requirements of a human pressing the button within the system environment.

j) **CAD geometry** – In the design of embedded systems, 3D CAD (Computer Aided Design) modelling is becoming more prevalent due to the need to fit components on circuit boards with space constraints. This need has lead circuit design software to utilise and export 3D elements. Also hardware systems increasingly are dependent on software, with software becoming a structural component of many systems. For example aircraft structures will fail or car engines will be damaged if the software fails to keep the forces exerted within the structural design limits. It is therefore becoming more important to treat the hardware and software models as a single entity and X3D-UML should facilitate this.
Individually the above features can be seen to add benefit to software visualisation. Combined they represent a powerful set that, if functioning as specified, would allow many combinations of features to be experimented with.

1.3 Research Contributions and Thesis Structure

In this section we present an overview of the research contributions and the structure of the thesis. As has already been described, X3D-UML provides a large variety of possibilities, however the main focus of this research is to measure benefit and this has been done in a few targeted areas. To do this, a methodology has been chosen and refined for use with X3D-UML. This methodology has then been applied to evaluate:

- the 3D extension of a specific UML diagram
- the integration between UML and physical world models
- the use of 3D to bring together multiple UML concepts in to a single view

Within this context the contributions and structure of this research are:

a) User-Centred Design, Implementation and Evaluation Methodology

The background to existing research in the area of 3D UML is presented in Chapter 2. This describes the extent and the limitations of the research into this area. Also described are the issues to be overcome and the approach required to overcome those issues.

The first issue is that despite advances in 3D computer graphics, 3D UML visualisations are extremely complex to achieve. To address this issue Chapter 3 documents a number of approaches to enabling 3D UML visualisations.

The second issue is that the area of 3D modelling of UML is novel and poorly understood. There are many variables both known and unknown to be considered with evaluating the benefit of a particular visualisation approach. Due to this, a specialised user centre design and evaluation approach is required to allow researchers to make best use of their research effort. Chapter 4 provides a refined user-centred evaluation approach specifically for 3D UML. This approach has been
refined through application to studying 3D UML and through a workshop session at the Layout in Software Engineering Diagrams ’08 [60].

b) 3D UML State Machine Diagrams

Using the refined methodology, evaluated is the concept that using the third dimension would significantly aid the software engineer in navigating substate and superstate layers in hierarchal state machine diagrams (also known as statechart diagrams). The traditional 2D approach is to place all levels of states on the same diagram or have substates abstracted to completely separate diagrams. The use of 3D allows all states to exist in the same view but be maintained as separate layered diagrams.

3D UML State Machine Diagrams represent the main focal area of this research; they have been used as the main means of refining the evaluation methodology and have had the most detailed analysis. 3D UML State Machine Diagrams are covered in more detail in Chapter 5. Such state machine diagrams have also been used to explore other areas (described below) such as integrating UML diagrams with hardware views.

c) 3D UML Mechatronic Diagrams

The term “Mechatronics” is not clearly defined [11] and for the purposes of this thesis “Mechatronics” is simply and loosely defined as systems that have integrated mechanical, electronic and software sub-systems that are co-dependent. An example of such a system is a robot where the system requires the interaction between hardware and software to function. The concept can also be applied to more subtle relationships between hardware and software, such as the interaction between an embedded processor’s output pins and the software driving those pins. If we consider the diagram Figure 1-2 which represents the coupling of these areas, 3D UML Mechatronic Diagrams can be defined as the integrated mechanical, electronic and software visual models of the system with a focus on the UML perspective of the system i.e. the software subsystem and its relationship to the other subsystem models.
Figure 1-2 – 3D UML Mechatronic Diagrams can be defined as the integrated mechanical, electronic and software visual models of the system with a focus on the UML perspective of the system.

Evaluated is the concept that with UML diagrams in a 3D space, diagrams can be integrated more closely with other aspects of the system which are three dimensional, such as the mechanical aspect of a system. The first section of Chapter 6 explores this concept against Lego NXT robots, where views of the software and the robot are combined into a single view. The area investigated is the debugging of state machine based software and the use of augmented reality to project state machine diagrams against their associated components.

d) 3D UML Holistic Diagrams

3D UML Holistic diagrams cover the concept that 3D views can provide a more complete view of a system. Current 2D UML diagrams only provide snapshot views into a model, which means the software engineer is presented with multiple diagram views into a system and it is left up to engineer to form a more complete model view in their heads. The holistic diagram attempts to remove this burden from the engineer and present a 3D diagram with which the engineer can tailor to suit the engineering task.
The second section in Chapter 6 also presents the extent of the research undertaken. Due to the approach taken and the lack of available users with the required expertise, the results are limited to reporting user tasks and concepts. However, these results reveal a promising area of further research and the chapter documents the approach that can be applied to gain further valuable information.

Finally, based on the results of the above evaluations, Chapter 7 provides conclusions to this thesis and future research direction.

### 1.4 Summary

In this chapter we have discussed the promise of 3D UML and X3D-UML. We have outlined past issues preventing practical use of 3D software visualisation and discussed how integrating X3D and UML not only addresses these issues but provides further advantages.

In the next chapter we look at prior 3D UML research and the issues surrounding evaluating whether 3D UML using X3D lives up to the theoretical promise.
Chapter 2 - 3D, UML and Evaluation

2.1 Overview

This chapter defines the context of the research related to the combination of 3D and UML. Firstly in Section 2.2 the definition of “3D UML” is presented to provide the scope for this area of software visualisation research. Then Section 2.3 presents related research within this scope and positions the contributions of this thesis. Section 2.4 reviews evaluation techniques and details how they have been applied to this research. Finally Section 2.5 presents a review of 3D technologies with relation to 3D UML visualisation.

3 Permission to use image for this thesis was kindly granted by Auste Mickunaite, Permissions, British Library, 96 Euston Road, London, NW1 2DB
2.2 Definition of 3D UML

The definition of “3D UML” is an important consideration as both the terms “3D” and “UML” can have different meaning for different people and have various meanings dependent on context. For this research 3D UML is defined as the visualisation of the Unified Modelling Language visual notation using 3D computer graphics. The implications of these individual terms are described in detail below and the intent can be summarised to using a 3D geometric description and associated graphics technology, to create a 2D projection for the purposes of visualising an existing notation which is familiar to software engineers.

3D (3D Computer Graphics)

For this thesis we define “3D” as 3D Computer Graphics, described by Watt [96] as the area that ‘deals with the processes involved in converting a mathematical or geometric description of an object – a computer graphics model – into a visualization – a two-dimensional projection – that simulates the appearance of a real object’.

Our research uses the concept of modelling UML as a complete geometric description, with the main purpose of creating a visualisation on a two-dimensional screen. The term ‘geometric description’ distinguishes our 3D diagrams from 2D diagram images projected on planes in a 3D space, which are more commonly referred to as “2.5 D”. Although our 3D geometric description may indeed be given the appearance of planes, every element is independent and can be used in different contexts due to the underlying 3D model.

The term ‘two-dimensional projection’ stresses that the final outcome is intended with a standard “2D” interface in mind. One common argument against the use of “3D” is that we are only using a 2D interface to view the scene so “3D” can add no value. However technically speaking 3D computer graphics is the science of using 2D interfaces for 3D geometric descriptions and this concept is used successfully in many applications, for example computer games.

---

4 For detailed definitions of “2.5 D” refer to chapter 2 of [27] T. Dwyer, “Two and a Half Dimensional Visualisation of Relational Networks”, PhD Thesis, The School of Information Technologies, The University of Sydney, 2005
For this thesis the definition used for UML is derived from the UML specification itself [70] as ‘The Unified Modeling Language is a visual language for specifying, constructing, and documenting the artifacts of systems.’

This thesis focuses on the ‘visual’ aspect of the UML as the means of communicating information to the viewer. The term ‘visual language’ is important to note because the use of the term “UML” need not necessarily automatically imply a link to a model, it may be as simple as a line drawing on paper or on a whiteboard. Also the term “modelling” does not necessarily imply a visualisation, as modelling can be achieved with textual languages. This distinction can also be seen with the UML series of conferences being renamed to Model Driven Engineering Languages and Systems [61] as a more appropriate description.

This definition also does not focus on any “particular version” of UML. The version of the UML is largely irrelevant and only of concern if the specifics of the version have impact on the research outcomes. Such areas where later versions solve problems in earlier versions do not impact on the research covered in this thesis.

2.3 Related Research

There are many areas of research which have implications for 3D UML such as software visualisation, information visualisation, visual languages, human computer interaction, 3D computer graphics, virtual reality, cognitive science and software engineering to name a few. A recent 2009 journal paper by Teyseyre and Campo provides an overview of the area of 3D software visualization which alone contains 194 citations [85].

To focus our research we first restrict the relevant research to general 3D UML, in which there is a direct relationship to the use of UML with 3D. We then expand this scope to include related research in the specific areas of our application.

While reviewing research in this area, we must acknowledge the difficulty in the research of 3D interfaces and usability. It is difficult to create 3D computer graphics tailored to a novel and specific task. If we take for example 3D games, these can have development
budgets in the millions of dollars, even with well defined requirements and API’s which are well tailored and tested for the application and experienced production teams. The individual researcher is unlikely to have any of these attributes and yet has to produce the tool to base their research on. Extending the research to test for benefit adds more known problems in user testing and the associated complication of involving human participants. To research each area alone is difficult enough but the combination can lead to compromise and limited research results, despite the best efforts of the researcher.

In this section we first discuss the related research in the area of 3D UML and then focus further on the research related to particular aspects of diagrams evaluated in this thesis. We revisit prior research and look at the evaluation of the visualisations and the previous absence of clear research results.

2.3.1 3D UML

In general the research directly related to the combination of 3D and UML is limited and patchy. A recent overview of 3D software visualisation listed X3D-UML as the only current 3D visualisation tool targeting UML notation [85], but neglected the recent implementation of GEF3D [33].

There are many forms of 3D software visualisations however most are based on metaphors which are unfamiliar to the software engineer, such as coloured boxes and spheres. Some of the contributing factors to the lack of 3D UML is the fact that 3D computer graphics in general is difficult to create. This coupled with the requirement that the visualisation must then reflect a specific familiar notation such as the UML, greatly adds to the complexity of creating a visualisation. Although high quality 3D computer graphics have been available for many years, the difficulty of implementation has meant that visualisations require specialised skills and are highly labour intensive to create. A review of 3D computer graphics technologies and the issues are detailed in Section 2.5.

We have based the visualisation on UML as it has gained wide acceptance in the software engineering workflow. By using an accepted standard, we can test the ability to render “real” software visualisation, rather than something contrived to suit the 3D
technology. By using the UML as a benchmark we gain greater understanding of the problems of software visualisation.

There are further advantages, in that the UML provides flexibility to expand its concepts from the standards given. Booch et al. [13] give explicit reference to this when referring to the nine common diagrams provided by the UML, stating that ‘To fit the needs of your project or organization, you can create your own kinds of diagrams to view UML elements in different ways.’

Booch et al. [13] also give explicit reference to 3D, in the same section, noting ‘In practice, all the diagrams you’ll create will be two-dimensional, meaning that they are just flat graphs of vertices and arcs that are drawn on a sheet of paper, a whiteboard, the back of an envelope, or on a computer display. The UML allows you to create three-dimensional diagrams, meaning that they are graphs with depth, allowing you to “swim” through a model. Some virtual reality research groups have already demonstrated this advanced use of the UML.’

The ‘advanced use of UML’ referred to above is the first paper to talk specifically about 3D extensions to UML which is that of Gil and Kent [34] who present ‘graphical notations demonstrating effective use of the third dimension in modelling’. The focus of the paper is the 3D formal notations which use the Z axis to represent specific types of edge connections between nodes or to replace textual annotations. An example of such a diagram is the contract box diagram shown in Figure 2-1. Gil and Kent [34] also note that to test the use of UML in this way some form of 3D environment would need to be developed. The paper did not have a specific implementation but suggested VRML may be the means of cheaply creating such diagrams.
Figure 2-1 – Contract Box. An example of a 3D diagram from “Three Dimensional Software Modelling”, Proceedings of the 20th international conference on Software engineering, ©1998, Joseph Gil and Stuart Kent.[34]5

Some further work in this area was presented by Radfelder6, Gogolla and Richters [35, 79], and propose ways of presenting 3D UML class and sequence diagrams, with the addition of animation. The advantages claimed in this approach are that both static and dynamic aspects of a system can be visualised in a single diagram. Also in viewing a diagram, the 3rd dimension can be used to emphasise important objects within the diagram.

5 Permission to use this image for this thesis kindly granted by the copyright holder Joseph Gil
In contrast to extending UML diagrams, Iran and Ware [46] propose an alternative approach to UML using 3D geon primitives and provide some evidence to the benefit of this approach for identifying structures in software. Casey and Exton [16] have extended this metaphor with a Java 3D implementation.

Dwyer [26] utilised a tool called “Wilma” to evaluate a force directed layout algorithm for use with 3D UML diagrams. An example of this is pictured in Figure 2-2, which was evaluated by a panel of experienced system architects.

Figure 2-2 – Example of 3D UML using force directed layout[26], Australian symposium on Information visualisation. © 2001 Tim Dwyer

---

Permission to use this image for this thesis was kindly granted by the copyright holder Tim Dwyer.
The first suggestion of including 3D into a UML specification by the Object Management Group [67] can be found in the UML Diagram Interchange specification Request for Proposal (RFP ad/2002-12-20). The proposal was aimed at enabling diagram information to be exchanged between UML tools. It was suggested in the proposal that an optional Z value be included to enable the layering of diagrams [67]. Although it was not included in the final specification it was noted as still being desired.

In the area of education Thaden and Steimann [86] utilise VRML to animate object oriented program execution through UML as a teaching aid. The visualisation presents an animated scene which illustrates how classes are instantiated in memory at runtime.

In understanding large generic C++ program structures, Hoipkemier et al. [42] describe an approach to visualising open source projects with a 3D UML class template diagram.

Lange et al. [54] demonstrate the use of 3D with UML diagrams to provide a view which supports the task of understanding model metrics. The visualisations of interest are “Metaview”, which although not a 3D view, combines a number of UML diagrams in a single view. “MetricsView” uses the Z axis to present model metrics overlaid on a single existing class diagram and “UML-City View” combines this approach with “Metaview” (i.e. MetricsView across many diagrams in a single view).

Anslow [6] has undertaken research in the general area of X3D software visualisation, which has included UML. The research looks at the possibilities of X3D and replicates some of the earlier X3D-UML work as part of the evaluation.

The most significant step towards practical 3D UML has come recently with the GEF3D framework [93]. The GEF3D framework builds on the Eclipse Graphical Editor Framework (GEF) and as such enables existing Eclipse 2D UML editors to be extended into 3D. GEF3D has been created with the task of working with multiple models and diagrams in mind and its architecture is open to allow any 2D GEF based editor to be extended. An example implementation using GEF3D for transformation traces can be seen in Figure 2-3. This project is an Eclipse project [33] currently in the “Incubation” phase.
This section has given an overview of the research that is directed related to 3D UML. The next sections cover related research not specifically related to 3D UML but which have some relation to the diagrams evaluated. The evaluation undertaken for all related research is reviewed in the subsequent Section 2.3.3.

2.3.2 Related Diagrams

This section discusses some related diagrams which are not strictly UML but are still directly related to our research.

The area of **3D UML State Machine Diagrams** in UML is derived from Harel [37] statecharts and from the early inception Harel had indicated that 3D may be a solution to problems with the higraph formalism. Harel [37] raised the issue of the inability of

---

8 Permission to use this image for this thesis kindly granted by the copyright holder Jens von Pilgrim
higraphs to specify both set inclusion and set membership and stated: ‘Most of the solutions to this notational problem that come to mind are somewhat unsatisfactory, with the exception of the one that calls for a three-dimensional basis for higraphs, in which the third dimension is responsible for such distinctions (e.g., by having set inclusion take place in the same plane and set membership be reflected by different levels of planes).’

In a much later publication a Eforni, Harel and Cohen [29] use a ‘pseudo statechart’ for describing biological processes utilising a distance perspective to give a statechart a three dimensional look. Lower level statecharts appear smaller and higher appear larger, to indicate closeness to the viewer looking down on the diagram.

Fishwick [31] has presented a fully immersive implementation of 3D state machines through VRML and the ‘rube’ methodology. The case study explores modelling a number of aspects of a teapot heating, including a finite state machine as shown in Figure 2-4.

Figure 2-4 – “Heating State” shows the current system state through the red sphere and green arrow state machine diagram. From “The 3D behavioural model design for simulation and software engineering” © 2000 Fishwick.

In the area of 3D UML Mechatronic Diagrams, there is not direct research in this area. Although there are many physical 3D models and a large body of research in all aspects

---

9 Permission to use this image for this thesis kindly granted by the copyright holder Paul Fishwick
of 3D modelling, the absence of 3D UML diagrams to integrate into those models means no research has been undertaken. The use of UML however has been applied, in the *modelling* sense, to the Mechatronic domain. Of particular note is the University of Paderborn’s Fujaba project [90] and *Mechatronic UML* [81], which extends UML to model Mechatronic systems. It must be noted that our use of the term *Mechatronic* is purely due to accuracy of the term in describing systems that are part software part mechanical, and is not related to *Mechatronic UML*. Part of our investigation however did involve investigating the use of 3D UML with Fujaba models.

Another area of our investigation with *3D UML Mechatronic Diagrams* is in using augmented reality with 3D UML diagrams. Again there is no direct research in this area, however again there is UML *modelling* research in the form SSIML/AR [91]. SSIML/AR uses a modelling approach to generate augmented reality interfaces automatically. Although SSIML/AR isn’t aimed at visualising the model itself in 3D UML, rather in modelling a 3D UI, the technology appears to compliment *3D UML Mechatronic Diagrams* by enabling the diagrammatic interface to be directly generated from the model.

In the area of *3D UML Holistic Diagrams*, combining views to give a more complete visualisation of a system, Ali’s MUDRIK [1] has close synergies with what we wish to achieve with 3D UML diagrams. MUDRIK is aimed at providing a software engineer with a complete view into a complex system and then allow the engineer to manipulate that view with an interactive flexible focus mechanism. The purpose of MUDRIK is to allow the engineer to comprehend a system as a whole and maintain a mental model while filtering out aspects not needed for a task.

Having discussed the approaches to presenting UML in 3D, we now discuss the evaluation undertaken for these approaches.
2.3.3 Evaluation of 3D UML

Whether 3D for software visualisation is of any real benefit is still an unanswered research question. As one example, Bowman et al. [14] list the three “Million-Dollar” research questions for 3D user interfaces. These questions below hold true for 3D UML and no one has yet claimed the million-dollars:

- ‘Can we quantify the real benefits of 3D UI’s?’
- Will there ever be a standard 3D UI?
- What is the killer app for 3D UI’s’

The million-dollar price tag associated with these questions is in the difficulty in answering them definitively. Firstly to quantify real benefit requires empirical user research which is can be extremely difficult in particular areas; empirical research into the use of UML alone is very sparse. A systematic literature review of research into UML for software maintenance by Dzidek [28] found only 23 out of 1572 UML papers containing user based empirical evidence. Of the 23 papers found, only 8 papers involved professionals as participants. With quantifying benefit in existing UML tools already known to be extremely difficult, we then have to apply 3D technology which is even less understood and extremely difficult to implement.

As outlined in the previous sections the research into 3D UML is limited. This section revisits 3D UML research from an evaluation perspective.

In the case of Gil and Kent [34] the 3D UML notation is only proposed and there was no supporting CASE tool to enable evaluation to take place, this has been noted as further work. Radfelder, Gogolla and Richters [35, 79] have implemented the visualisation but have stated also that it ’is only a first step on a journey[35]’ and have not evaluated the diagrams against any strict criteria.

Irani and Ware [46] provides experimental evidence that 3D geon diagrams can out perform UML diagrams in the area of analysing system structure. Their experiments show that users can more easily identify and remember structures visualised as geons rather than UML. The results are based on student participants and UML diagrams
constructed for the purpose of the experiment. Not reported was the training time required for participants to learn the new diagram. Casey and Exton [16] do not evaluate their Java 3D geon implementation and rely on the results from Irani and Ware [46].

Dwyer [26] provides qualitative evidence as to the benefit of 3D UML through a heuristic evaluation undertaken by ‘experienced system architects’. The results show some evidence that the presentation of UML in 3D, with force directed layout, helps in understanding the architecture of complex systems. The data used for the visualisation was derived from ‘real software’ however the actual complexity was not quantified. The paper states the results are to be considered a pilot study, with further studies required to provide conclusive evidence.

Thaden and Steimann [86] provide no evaluation of their teaching tool. Hoipkemier et al. [42] evaluate only the performance of their tool for rendering open source projects and provide no user evaluation of benefit.

The most comprehensive evaluation is undertaken by Lange et al. [54] which covers the tool MetricView Evolution. The tool only makes limited use of 3D with UML, with one set of diagrams displaying metrics as 3D bar graphs on top of UML classes, however the evaluation is thorough. Lange et al. undertook a controlled experiment involving 100 MSc students, with the results showing the effort needed for tasks reduced by approximately 20% and the correctness of comprehension increased by approximately 4.5%. The data used was ‘larger than a pure toy-example’ but with only 38 classes. The size of the data has been noted as a threat to external validity as well as the comprehension tasks tested.

Anslow et al. [5, 6] and Fishwick [31] provide a self assessment evaluation of the technology used to create the 3D diagrams but provide no user evaluation. The diagrams are presented as case studies with the intent of providing evidence of the advantages and disadvantages of using VRML/X3D for such visualisations. Ali [1] presents MUDRIK with a similar intent for the specific tool and provides no user evaluation.

In the area of “2D” statecharts there is a ‘family’ of experimental work in the understandability of statechart diagrams with relation to composite states [17-21]. With one experiment of special note involving professional engineers [20]. The data used in
the experiment is that of a digital watch open source example [99]. The focus of the experiment was to test if the use of composite states aided in human understanding but did not account for the use of composite states as a formal means of defining system behaviour (i.e. for code generation). Participant groups were presented with two diagrams, one ‘modelled using composite states’ and another being ‘exactly the same system but modelled without using composite states’. It was not stated how the removed composite state common behaviour were replaced to achieve the same system (e.g. entry/exit actions) and the impact of this change was not tested. The conclusions, from an informal perspective, is that composite states did not aid understandability, however we also note that diagrams were small enough to be a single diagram. We will see later in Chapter 5 that with formal use of composite states, actual diagram sizes in industry are much larger than used in this experiment.

For GEF3D [93] aspects of user evaluation have been undertaken as part of our research in collaboration with Jens von Pilgrim. GEF3D in relation to X3D-UML is described in the next section, which positions the contributions of the research in relation to existing research.

2.3.4 X3D-UML in Context

In this section we now look at the research undertaken in context of the related research. Reviewing the 3D UML implementations and evaluation we find that there is very limited research in this area, despite the popularity of UML, most 3D software visualisation research ignores UML as a basis for a visualisation approach. The research that is presented is mostly exploratory, “one off” and only provides limited insight into the possible benefit of 3D UML. The most comprehensive evaluation, that of Lange at el. [54] reveals some measurable benefit for a small aspect of 3D UML (overlayed bar graph metrics), however this is measured on students, assumed data sets and assumed user tasks. The experimental approach is thorough and the results are valid in that context, however to truly extend these results to industry requires the same experiment to be conducted on experienced engineers, large industry models and against existing user tasks (or undertake an industry survey to compare the variables used with actual engineers, models and tasks). In a similar fashion Irani and Ware [46] provide evidence based on students, assumed data and assumed user tasks. The evaluation results of
Dwyer [26] in contrast are based on actual data and experienced system architects, however the study is qualitative and stated as being only a pilot.

The body of research shows that there is evidence that the area of 3D UML provides some promise, however the research is limited and combined does not form a complete picture. Our research is aimed to address these limitations and to form a more structured and comprehensive approach to applying and evaluating 3D UML. To achieve this we first place constraints around how the research is approached and these constraints are described in Table 2-1. The reasons for these constraints are to focus the research results so they are as broadly applicable to industry as possible. The rules do not prevent novel areas being explored but they tether the underlying data back to existing UML development infrastructure. The constraints can be generalised to ensuring that visualisations are standards-based and that they will work with existing infrastructure, data and processes. These constraints are described on the following page:
<table>
<thead>
<tr>
<th>X3D-UML Design Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a) Actual Data</strong></td>
</tr>
<tr>
<td><strong>b) Actual Tasks</strong></td>
</tr>
<tr>
<td><strong>c) UML</strong></td>
</tr>
<tr>
<td><strong>d) X3D</strong></td>
</tr>
<tr>
<td><strong>e) Standard PC Hardware</strong></td>
</tr>
<tr>
<td><strong>f) Development Environment Imitation</strong></td>
</tr>
</tbody>
</table>
Using the above constraints we then apply a user centric design approach, to implement and evaluate a 3D UML approach. The aim of this approach is to first understand the user tasks, implement a solution to address real world problems and to evaluate the success of the 3D UML solution. Using this approach addresses the limitations in prior research by giving a more comprehensive evaluation.

In the context of the above the recent GEF3D framework [93] meets many of the X3D-UML design constraints, however the research goals differ. The differences can be summarised to the approaches implementing 3D UML from different perspectives. GEF3D builds up 3D capability from an Eclipse IDE, whereas X3D-UML applies already existing 3D capability to UML. The differences are explained in more detail below:

a) GEF3D is based on Eclipse and therefore research is restricted to that platform.

b) GEF3D is based on Eclipse and is leveraging a development platform primarily designed for 2D graphics. The advantages are that integration into development tasks is more natural. However GEF3D is disadvantaged in 3D aspects, for example interaction techniques such as “picking” objects have to be written into the framework.

c) X3D-UML is based on the X3D specification and is leveraging platforms primarily designed for high performance 3D graphics. The advantages are that all areas of 3D are better supported. However the disadvantage is that aspects such as integration into a development platform or editing have to be created.

GEF3D and X3D-UML have a natural synergy and as such collaboration has taken place with Jens von Pilgrim in the area of evaluation. The goals of the X3D-UML research are to understand the application of 3D UML and our research results should directly benefit users of the GEF3D framework. Being 3D graphics based, X3D-UML allows more novel visualisations to occur and can more easily explore areas which are not practical with GEF3D. The evaluation of GEF3D in collaboration with Jens von Pilgrim is covered in Section 4.6.
2.4 Evaluation Techniques

In the previous section we have documented the limitations in existing 3D UML evaluation. This section explains the aims and the context of the research and how the evaluation technique was selected.

2.4.1 Aims

Fully implemented 3D UML diagrams do not exist within software development tools, so it is therefore impossible to test the benefit of the use of 3D UML diagrams compared to 2D UML diagrams. For a test of 3D UML benefit to occur, a 3D UML extension must first be created. The details of our implementation are described in Chapter 3.

The aim of our research methodology is to use X3D-UML to push UML in the direction of 3D from a familiar and commercially used 2D starting point and be able to trace the path from that starting point to the final 3D end point (where ever that may be based on time and resources). At that 3D end point, the visualisation is tested to determine the extent of benefit in terms of actual software engineering tasks.

As 3D UML is novel and untested, the visualisations presented in this thesis are restricted to initial 3D extensions which provide some measureable benefit. However, because the research is X3D based, research possibilities are opened up to all areas X3D supports such as shared virtual worlds or fully immersive virtual reality. This research aims to probe the surface of 3D UML research but it does so with tools and techniques which provide the ability to “keep drilling” further into all areas of 3D UML visualisation.

2.4.2 Context

It would be highly impractical to attempt to create a fully functional 3D UML tool, with all the features available in today’s current UML tools. In current UML tools, teams of diversely skilled software engineers struggle to provide all the UML features the market requests, there is little hope for a single researcher project to match this and then to further extend a tool with 3D. The research is therefore undertaken so that the visualisations are an extension to the current suite of features available in each developer’s environment. The visualisations are created to be as close as possible to the
existing 2D equivalent and are provided as an added diagram, rather than a replacement tool. In cases where the UML tool provides ease of extensibility the 3D UML diagram is embedded directly into the tool itself as an add-in.

Unfortunately, the X3D-UML extension will not be a 100% seamless integration. For example users will not be able to edit the diagram on the fly like they may be able to do within their 2D UML tool (as 3D design suites are just as hard to create as UML tools). The research thus has to be selective about what is tested and noted down through user response if further implementation is required or desired.

2.4.3 Evaluation Techniques

Bowman [14] provides an extensive summary of 3D user interface evaluation techniques and guidelines to evaluation technique selection. Further to this, combinations of these techniques are also presented with two multi-method approaches being defined as “Testbed Evaluation” and “Sequential Evaluation”.

“Testbed Evaluation” is focused on empirically evaluating low level interaction techniques. This form of evaluation is not suitable as the area of 3D UML visualisation is not mature enough to concentrate on such low level performance measurements without understanding higher level goals. In the future however, if tools start to present a number of 3D UML diagrams, this evaluation technique could be utilised to provide user performance data for defining common interaction techniques and standards across different diagrams (i.e. defining the best consistent 3D interface).

“Sequential Evaluation” is found to be most suitable because it considers higher level usability. Although the results are application specific, UML tools and the uses of those tools are similar enough that the results are broadly applicable. The approach also suits new concepts and allows for the evolution of visualisations based on research outcomes.

Diehl [24] provides an overview of software visualisation specific evaluation techniques, and identifies aspects of the outlined Quantitative and Qualitative evaluation techniques which are addressed by “Sequential Evaluation”. It may also possible to use the Cognitive-Dimensions Framework outlined as a guide during the Heuristic Evaluation stage, which we explain later in this Chapter 4.
For our research we base our evaluation on “Sequential Evaluation” and extend this methodology further to be specific for 3D UML. We refer to this as “3D UML Sequential Evaluation” and the details of the methodology and our refinements are described in Chapter 4. The evaluation issues this methodology addresses are detailed in the next section.

2.4.4 Issues

Bowman et al. [14] provides list of issues that need to be considered while planning the evaluation of 3D user interfaces. This section addresses those issues (a-e below) which are relevant to this particular research. Most issues are not directly relevant because the research is not undertaken using specialised hardware. Diehl [24] also lists typical problems in “toy data sets”, incorrect “visual artefacts” and “bias” are also addressed in this section (f-h below).

a) **Physical Environment Issues** - This research does not utilise specialised user interface devices (such as head mounted displays) which would physically impede the researcher from observing and communicating with participants. Therefore there are no specific problems related to evaluating user interaction during their use of the application. However in general 3D UML research, there may be instances where the physical environment impacts the ability for the researcher to observe the participants. Even with standard pc hardware there may be instances where this issue would need to be considered, such as observing multiple dislocated users simultaneously collaborating on a networked 3D UML view.

b) **Evaluator Issues** - This research does not rely on the participant having a strong sense of presence within a 3D UML view for the view to be effective; therefore there are no issues with the evaluator causing breaks in presence while observing the participant. Again the research does not utilise specialised user interface devices, so there will be no issues with the evaluator keeping track of complex interactions the participant might need to do to complete a task.

c) **User Issues** - Issues related to the novelty of interacting with 3D user interfaces play a major factor in this research and part of the aims of this research is to
document the specific interactions participants find difficult. Again our research does not utilise specialised user interface devices, only equipment that participants are already familiar with, so there are no issues with novel equipment causing discomfort during evaluations.

d) Evaluation Type Issues - Issues related to the novelty of capturing and evaluating information with respect to 3D user interfaces are part of the aims of this research. Specifically this research is to make this area better understood and less novel. The research also uncovers heuristic techniques in the area of evaluating 3D UML and performance models (i.e. calculated benefit based on model analysis). Issues related to the quantity of variables in tests are addressed by restricting the initial research to testing basic extensions. The aim is to determine if solid evidence of benefit exists with simplistic visualisations and to have that evidence warrant further investigation into more complex visualisations.

e) Miscellaneous Issues - Issues related to variability in input devices are of concern and this concern can be extended to variability in all aspects of the environment. Our research aims to provide evidence of benefit on standard pc hardware; however 3D technologies that may work on a Microsoft Windows machine may not work on a Linux machine. Also the initial development environments are generally different across different operating systems. The thesis documents the environments used so that it is clear which configuration has been tested and to which environments the conclusions pertain. There is some scope to generalise the results across different environments in areas where the visualisation has no environment specific dependencies or where equivalent alternative technologies exist.

f) Toy Data Sets - The issues related to toy data sets not being a true reflection of the usefulness of a visualisation has been addressed by the research being constrained to Actual Data as described by our X3D-UML design constraints in Table 2-1 The reasons for this are to directly address the issue of the data being tailored to suit the visualisation. For usefulness of a visualisation to be accurately ascertained it must be tested against the types of data currently found in software engineering environments. In addition to this, certain types of data are expected to suit a 3D visualisation more than others. To accurately express conclusions
about benefit, our research also measures the frequency that the types of data studied actually occur in reality.

g) **Incorrect Visual Artefacts** - The possibility of visual artefacts suggesting nonexistent relationships are addressed as part of the methodology. The tasks the users are undertaking and the information that the visualisation is imparting are examined through different techniques to reduce such problems with the visualisation.

h) **Bias** - The issue of bias is a major issue; Diehl [24] states the problem as being specific to the evaluations being undertaken by the developer of the visualisation (i.e. motherhood issues), however the problem of bias can be extended to prejudices of all participants in a study. The experience throughout this research is that people have strong preconceived notions of the benefit of 3D, or not and this has noticeable impacts on the person’s involvement in the research. This bias impacts research at all levels, from the initial research proposal to the data gathered and the final thesis review.

It has been found that an “anti-3D” prejudice reduces the persons desire to participate in the study in any form. This has impact even when the person is asked to state as many negative things about a visualisation as possible. “Anti-3D” people have been found to favour non-participation rather than providing negative but constructive input.

It has been found that a “pro-3D” prejudice increases the persons desire to participate. This has the advantage of attracting users to a study, which would not have previously volunteered; however this skews any results from volunteer participants. This skew is not always in favour of a “prototype” 3D UML visualisation, as participants who are “pro-3D” may be so from using highly polished and easy to navigate interactive 3D applications

The only solution to this issue is to reduce the impact of bias by documenting bias situations and collecting as much unbiased quantitative supporting data as possible, such as model metrics and from unbiased experiments. Although “3D bias” is found to be significant, bias is to be expected in any user study and there
are accepted research methods for reducing the impact and reporting results accurately.

2.5 3D Technologies

In the previous sections we reviewed the research related to 3D UML and evaluation techniques. In this section we review the technology that may be used to form the basis of 3D UML implementation.

2.5.1 X3D Issues

This section describes the issues with using X3D for 3D UML visualisation. The aim of this section is to highlight the issues that need to be considered when developing visualisations with X3D. These issues complicate the already complex task of producing a 3D visualisation. The sections describe the problems with X3D browsers and X3D specification limitations related to describing and interacting with UML visualisations.

2.5.2 The Browser Minefield

Problems with X3D browsers greatly impact UML visualisation. Many of the software visualisation advantages noted in the introduction to X3D (Section 1.2.5) are advantages in theory only, due to how the specification has been realised. Despite 10+ years of maturity VRML/X3D technology still struggles to deliver to expectations due to browser limitations [76].

The first problem encountered is that, depending on the computer system and the X3D scene, many browsers just do not work. Problems can vary from the browser crashing before doing anything, to blank screens or partially rendered worlds with no error message to indicate why. The second problem is that if they do work, they have an incomplete implementation so not all X3D features can be utilised. Further to this problem, different browsers implement different aspects of the specification, so “jumping” between working browsers is not possible.

The result is a browser “minefield”, where getting to the other side, is a process of careful trial and error, using different browsers and different X3D features. The problems
were found to be more evident for software visualisation due to the use of features that are not commonly used in traditional 3D rendering (which are discussed further below).

Fortunately the browser technology is under active development and the X3D technology is improving. For instance, our initial research (Section 3.4) had found that a 3D UML class visualisation of a typical system size was not possible and therefore X3D could not be used effectively for UML. However, advances in browser technology starting in late 2004 enabled these visualisations to become possible with at least one browser and since then more browsers have become capable of at least visualising large amounts of text and lines.

For X3D-UML research, Bitmanagement's *BS Contact* browser [12] was found to be the most advanced and best suited to UML visualisation. *BS Contact* does not however, implement features such as 2D objects and does not support a wide variety of operating systems. Despite this, it provided the most complete set of features and was the plug-in found to be the most predictable in performance.

### 2.5.3 Problems Specific to Software Visualisation

As stated previously, X3D is designed for defining interactive web- and broadcast-based 3D content. It is not specifically designed with software visualisation in mind and when applied in this way some limitations and idiosyncrasies are uncovered.

X3D has a limitation on user defined types in that they are not designed with abstract data in mind. With user defined types, an X3D content creator can combine X3D nodes to create new types. For example, a user defined “chair” could be created by combining box nodes of various predefined shapes and then binding properties of the chair to user defined attributes. For abstract data visualisation however, the user defined attributes are not sufficient and there is no way to manage the storage of large quantities of information and the binding of that information to a visualisation. For example, it is not possible to define a “source code” attribute and then bind the attribute to form a visualisation based on the attribute contents.

X3D primitive objects are not suited for constructing software visualisation scenes. X3D, as with VRML, comes with primitive types, such as box, sphere and cone. At first glance these primitive objects appear ideal for a quick and easy visualisation (and are included
in many VRML “helloworld” examples) however, for software visualisation using them introduces dynamic limitations as they have limited ability to change at runtime. Primitive objects are intended to be used to construct a more complex visual object and are only intended to be scaled, rotated and repositioned through a transform node. As an example, for “real-time” UML software visualisation, treating a “box” primitive object as a “class” would make adding and removing attributes and methods difficult.

Software visualisation based on UML notation requires a large amount of unique text and lines, numbering in the thousands. Ideally “primitive” UML types, such as boxes, circles, ellipses, “roundtangles” and other common UML notations are required; however these also have to be constructed from lines. This requirement for large amounts of unique text and lines is highly unusual in typical polygon based 3D scenes that are aimed towards photo realistic real world rendering. This need for many lines and text causes major problems with UML visualisation as there is no inbuilt optimization as there can be for polygon meshes.

Even if all X3D specification aspects are catered for by the browser, there is a strong chance that only small scenes will work effectively. The high-end browser, Bitmanagement’s BS Contact, needs non-default optimization settings enabled to allow effective rendering of large amounts of text. It is possible for the X3D implementer to create scene optimisation mechanisms, such as “Level of Detail” nodes to reduce scene content, however such compromises are time consuming and not ideal.

2.5.4 User Interaction

This section describes two issues with X3D user interaction. X3D provides user interaction through a number of sensor nodes, which “sense” events from the user and these events can then be acted on. These individually are powerful and easy to use; however there are issues, described below, with their use for software visualisation. The possible actions as a result of these events are also limited in terms of software visualisation.

In a typical X3D scene, the user is exploring a physical world and the X3D interaction easily supports that paradigm. The avatar may navigate this world, pick up objects, press a button, push an object etc. These actions are mostly independent and objects
are not designed to do complex actions, for example a “button” object will only need to handle a touch event. Software visualisation requires much more complex actions per object than the physical world paradigm; every software object may need a number of interactions such as “expand/collapse”, “movement”, “filtering” and “hyper linking”. X3D does not easily cater for multiple types of events per object, such interaction is possible but complex to implement.

The ability to change the scene as a result of user interaction is again limited by the physical world paradigm. Common actions from the physical world or actions supporting building a scene are easily achieved in X3D. For example, scaling an entire object structure (such as a chair) within a scene can be achieved by scaling the node that contains the object. For a change to a software scene, such as filtering an irrelevant code branch from a scene, actions such as fading an entire object are not supported from a single container node and again are possible but complex to implement.

As an example of the implementation complexity for software visualisations in this area, we present the previous example of requiring the fading of an object. In software visualisation, we may wish to make a branch of a state machine diagram transparent to filter unneeded data from the scene. This state machine branch may contain hundreds of software objects (states, transitions, junction points, text items, etc) and although they are part of the same object (i.e. grouped by the same node) to achieve the fading we must treat each individual object as a separate entity. The implementation would require the fading logic to be created for every object in the state machine branch and individually linked to the event. This implementation requires hundreds of event routes to cater for just one user interaction feature; and the process would have to be repeated for each new feature.

One solution to the problem is to make use of the X3D Scene Access Interface (SAI). This would in theory allow an external application to become the central controlling point for complex behaviours outside of the scene graph and then update the scene graph as a result. However it requires the SAI feature to be an available feature in the browsers capable of rendering UML. Although the BS Contact browser had this capability, it was not possible to test this theory as it required the purchase of an SDK.
2.5.5 3D Technologies

As a result of the issues experienced with X3D, alternative 3D technologies were periodically reviewed throughout this research. This section explains the important attributes we considered when evaluating 3D technologies for use with 3D UML and then the results of the periodic reviews of alternative 3D technology based in these attributes. These sections together provide a resource for 3D UML implementers to quickly select and evaluate the technologies which best support the UML visualisations they wish to achieve.

2.5.6 Desired Technology Attributes for 3D UML

Researching 3D UML visualisations can be compared to deep ocean exploration. Relatively little research has been undertaken because any exploration requires overcoming rapidly increasing pressures accumulating from all directions. Many attributes have to be considered, because a failure of any one would compromise the ability to get even close to the research aim of generating and testing a 3D UML visualisation. The section outlines the important attributes to consider when evaluating 3D technology for undertaking 3D UML research.

a) **Ease of Use** – The 3D technology should provide an appropriate level of abstract to aid in the creation of the 3D visualisation. 3D graphics are extremely difficult to create and technologies which reduce the need to program everything from the ground up, can aid the researcher to focus on the visualisation research rather than on learning low level 3D programming. Ease of use can be measured by 3D technologies that support the attributes listed in this section.

b) **Specialised Text Rendering** – UML and source code by their nature utilise large quantities of unique text. Many 3D engines attempt to draw text as 3D models of characters and struggle to render UML scenes. The 3D engines which have specialised text rendering in the form of texture mapped fonts, enables large quantities of high quality text to be viewed and navigated. Text rendering must be tested to confirm that many thousands of words of unique text can be rendered without hindering smooth navigation.
c) **3D Lines** – The 3D technology must support the drawing of lines that maintain their visibility in a 3D space. It must be possible to draw a relationship between objects and have consistent visibility of that relationship from differing angles, distances and perspectives. The technology should also be tested to see if it can render many thousands of unique lines and still allow smooth navigation.

d) **User Interaction** – The 3D technology should support user interaction as the UML visualisation will not be static. User interaction in a 3D space is very complex and technologies that already support navigation and selection of objects are an advantage.

e) **Extendable** – The 3D technology should be extendable to allow new functionality to be added. As there are no technologies specifically aimed at UML visualisation, it is essential to have technology that allows the researcher to create custom types and actions.

f) **Appropriate User Interaction Metaphor** – The 3D technology should have the appropriate metaphor for user interaction. For example, 3D technologies aimed at “first person shooter” games, may only cater for particular styles of user interaction and limit research to that interaction metaphor.

g) **Object Oriented** – Although not essential, it is an advantage that the visualisation technology has a similar underlying data structure to that which is being visualised. This should lead to a more natural mapping of visual artefacts to UML artefacts.

h) **Maturity and User Base** – Technologies that are mature and have a large user base have fewer problems and the problems that do exist are likely to have solutions available through searching forums. Since development costs for 3D are high, very new technologies run a high risk of becoming obsolete if they do not succeed in the market. Mature technologies also have more 3rd party tool support such as easily exportable/importable formats.

i) **Completely Supported Import/Export Formats** – 3D technologies claim to import and export specific formats (e.g. VRML, X3D), however these capabilities are rarely fully functional. Implementations vary greatly in quality and many tools
completely ignore scene text during the import process. Converting between formats is difficult and most effort has gone into importing external formats to enable adoption of the technology using the import. It is important to not rely on claimed import/export feature sets and determine through testing that the import/export capability can meet the research requirements.

j) **Cross-Platform** – Although not essential, it is beneficial to have the ability for the visualisation to work on different operating systems (e.g. Windows and Linux). The 3D technology should however be aimed firstly at the operating systems of the target users, with portability being a secondary consideration. Cross-platform technology also enables flexibility in the users targeted, the research carried out in this thesis was primarily aimed at Microsoft Windows users however later in the research a Linux user base was catered for.

k) **Cost** – 3D graphics are complex and development costs are high. These costs can lead to technologies that are expensive for use in research, such as licensing schemes that require ‘per user’ purchases. Cost can also lead to technologies that contain proprietary implementations that help protect the vendor but restrict research options.

l) **Open Source** – For Extendibility and User Base, open source software are of benefit. When choosing an open source project it is important to choose software which has an active developer community and that each feature that the UML visualisation will depend on, also has active developer support. A commercial aspect to the open source project may also lead to more robust releases due to financial obligations to maintain working software.

### 2.5.7 3D Technologies Reviewed

Throughout this research, we have periodically reviewed 3D technologies. In recent years, the 3D technology has evolved rapidly due to advances in graphics hardware capability which in turn has fuelled better 3D software support. This section captures the technologies that were reviewed and explains why the technology was not adopted. The extent and type of the limitations stated will change over time as certain technologies gain more support and others become obsolete.
XML Formats

While X3D is an implementation utilising XML, there are a number of alternative formats to X3D which also support 3D content stored as XML. The three main formats reviewed are discussed below:

a) XAML Extensible Application Mark-up Language (www.microsoft.com) – XAML is an XML based presentation layer aimed at Microsoft Windows based applications as well as web-based front-ends. XAML is intended as a mark-up language for user interfaces, with the language supporting 3D content. The XAML pre-release version supported the use of 3D lines (*ScreenSpaceLines3D*) and text, however the released version did not support 3D lines and this made it unsuitable for UML visualisation. There are ways to extend XAML through user code to support 3D lines [77], however the complexity this introduced was not desirable for our work with 3D UML diagrams.

b) COLLADA (www.collada.org) – COLLADA is an XML based interchange format to support transfer of 3D content and assets in a standard way. COLLADA is less mature than X3D/VRML and not designed to be a complete visualisation solution with a ‘specific run-time model that enables picking, viewing, navigation, and scripting, and an API to manipulate the scene graph at run-time [7]’.

c) SVG Scalable Vector Graphics (www.w3.org/Graphics/SVG) - Scalable Vector Graphics is a standard 2D XML format which is suitable for standard UML visualisation. The process for visualising UML through SVG is outlined in the UML Diagram Interchange specification [68]. SVG does not natively support 3D and is therefore considered high risk when considering the development of a 3D UML visualisation. A number of UML tools export diagrams as SVG, however we could not find a suitable converter of SVG to X3D (or other common 3D formats) to leverage SVG as a means of extracting diagram information.
3D Engines

There are a large number of 3D “Engines” available which provide a framework of features to support particular applications, such as games. DevMaster’s Game and Graphics Engines Database (www.devmaster.net/engines/) provides a good source for locating a number of these 3D Engines (not just for games), with over 290 engines listed. The database can be queried based on desired attributes, such as language, operating systems supported, however attributes such the quantity of text and lines viewable are not catered for. This section reviews some of the promising technologies that were considered for our research:

d) **Quest3D (quest3d.com)** – This is a commercial 3D engine and development environment which is able to quickly create 3D scenes. The development is limited to Quest3D’s visual Object-Oriented development environment, with scenes and functionality created by linking function blocks. This was not adopted due to licensing cost and because displaying large quantities text was not supported without optimization techniques.

e) **XNA (www.microsoft.com)** – XNA is Microsoft’s .Net Framework based game development environment. XNA supersedes Managed DirectX. Both these technologies are aimed at typical 3D graphics and optimized text is not directly supported.

f) **Irrlicht (www.irrlicht.sourceforge.net)** – Open Source 3D graphics engines, with an example of 3D UML [47]. The Irrlicht engine does not have optimized text and the example given are generated by overlaying texture images of complete classes on 3D objects.

g) **Unreal (www.unrealtechnology.com)** – Commercial product with older versions being released to public. This has been used for software visualisation in the past [52] however is limited to the “first person shooter” metaphor.
3D Libraries

3D libraries provide a higher level of abstraction to the 3D graphics problem by providing generic 3D features required by most 3D applications. These libraries are not as easy to use as graphics engines because they do not provide high level features, however they are much more flexible in their use. In this research OpenSceneGraph has been chosen as an implementation technology, this and similar technologies are reviewed below:

h) **OpenSceneGraph (www.openscenegraph.org)** – This OpenGL 3D scene graph is an open source C++ toolkit under active development. Evaluation tests have shown that it is able to render large quantities of text and this can be attributed to its use of the FreeType font engine (www.freetype.org). Details of the X3D-UML OpenSceneGraph implementation are given in Chapter 3. OpenSceneGraph has “quickstart” documentation [59] and an active mailing list community. Some low level knowledge of OpenGL is required and more complex features are only documented through C++ code examples.

i) **OpenVRML (www.openvrml.org)** – OpenVRML is a small open source X3D/VRML library, which uses OpenGL for rendering. Source code and project files for building with Microsoft Visual C++ Express are provided, however due to other 3rd party library dependencies this proved to be too complex a task to undertake for an evaluation. We assumed that due to the difficulty in building OpenVRML for Microsoft Windows it would not be a well used and well tested graphics engine in that environment. OpenVRML is used in an augmented reality toolkit [8], and it was observed that lines and text are not correctly displayed (pictured later in Figure 6-1).

j) **NVIDIA Scene Graph (www.nvidia.com)** – NVSG (NVIDIA Scene Graph) is a free commercial scene graph SDK similar to OpenSceneGraph. This technology, like OpenSceneGraph, lists the FreeType font engine (www.freetype.org) in its feature list which suggests that it would be capable of rendering UML scenes. A test to determine this capability failed due to an incomplete implementation of NVSG’s VRML import capability, where text was ignored. Another issue is that this scene graph is restricted to the “NVIDIA FX family of graphics cards” [66]
which would only make this scene graph suitable in researcher controlled environments.

3D API’s

3D API’s provide the lowest level access to 3D graphics creation. These provide 3D software programming capability where the software engineer is required to build up all the functionality required. The use of a low level 3D API does provide the best means to optimize the visualisation but is also the most labour intensive. Due to this effort and the relative capabilities of higher level libraries and engines, direct access to 3D API’s to achieve visualisations is becoming less needed. Due to effort 3D API’s have not been chosen, the main 3D API’s are listed below:

k) **Java3D** ([java3d.dev.java.net](http://java3d.dev.java.net)) – An open source 3D API for Java.

l) **Microsoft DirectX** ([www.microsoft.com](http://www.microsoft.com)) – A proprietary 3D API for Microsoft Windows platforms.

m) **OpenGL** ([www.opengl.org](http://www.opengl.org)) – Open Graphics Library is an open cross platform 3D API.

n) **Macromedia Flash** ([www.adobe.com](http://www.adobe.com)) – Is a proprietary web API. It is possible to create 3D like functionality using low level features.

The following table summarises the issues highlighted above against the desired attributes for 3D UML visualisation. The table highlights only the tested blocking issues and absence of the stated blocking issue.
Table 2-2 – Summary of 3D Technology Issues. For convenience purposes, this table provides a summary of the blocking issues stated above to using the technology for this research.

X = the tested blocking issues for this technology.

<table>
<thead>
<tr>
<th></th>
<th>a) Ease of Use</th>
<th>b) Specialised Text Rendering</th>
<th>c) 3D Lines</th>
<th>d) User Interaction</th>
<th>e) Extendable</th>
<th>f) User Interaction Metaphor</th>
<th>g) Object Oriented</th>
<th>h) Maturity and User Base</th>
<th>i) Import/Export Formats</th>
<th>j) Cross-Platform</th>
<th>k) Cost</th>
<th>l) Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) XAML</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b) COLLADA</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c) SVG</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d) Quest3D</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e) XNA</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f) Irrlicht</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g) Unreal</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h) OpenSceneGraph</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i) OpenVRML</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>j) NVIDIA Scene Graph</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k) Java3D</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l) Microsoft DirectX</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m) OpenGL</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n) Macromedia Flash</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.6 Summary

In this chapter we have presented our research in the context of prior research and the available technologies. We have discussed prior 3D UML research and the extent to which evaluation of 3D UML has been undertaken. We have seen that despite the popularity of UML, very little research has been undertaken into extending UML with 3D. Of the research that does exist, only limited evaluation has been undertaken.

We have then discussed the issues with evaluating 3D user interfaces from both an evaluation perspective and a technology perspective. In the next chapter we look at implementing X3D-UML.
Chapter 3 - Implementing X3D-UML

3.1 Overview

In this chapter we present our implementation of X3D-UML as means of creating advanced 3D UML visualisations. As part of our research we have developed and tested a number of implementations, with the final implementation consisting of a combination of techniques most suited to presenting UML in 3D. In Section 3.2 we present the X3D-UML library, which is the final implementation available through open source. We also present early implementations which are suited to particular tasks. Section 3.3 describes creating X3D visualisations that can be treated as source code and directly compiled as such. Section 3.4 describes translating XML representations of source code into X3D-UML representations. Finally we summarise these implementations in Section 3.5.
3.2 X3D-UML Library

This section describes the implementation used for X3D-UML user studies and which is undergoing continued development. In Section 3.2.1 we describe the UML Diagram Interchange specification [68] and how it has been leveraged to create standards-based 3D UML visualisations. The X3D-UML library implementation is in two parts, one part supporting X3D and the other part supporting complex interaction techniques. In Section 3.2.2 we present the C# implementation designed for generation of X3D from UML, with content capable of being viewed through a 3rd party browser. In Section 3.2.3 we present the OpenSceneGraph C++ implementation which provides a highly customisable interactive standalone viewer for 3D UML.

The aim of the study is to represent UML in a 3D space. We then employ a user-centred design approach to find and address issues experienced by software engineers in industry. The results of our research have required technology changes in the implementation to enable the visualisation to meet the user requirements. These changes have resulted in two main separate implementations to meet user needs: one based on C# X3D generation; and the other on OpenSceneGraph C++. Both implementations are derived from, and extend the UML Diagram Interchange specification.

The X3D-UML C# implementation captures UML diagram information through the UML toolset API and then generates X3D for viewing in an X3D browser or plug-in. Therefore this implementation is highly reliant on 3rd party X3D browser implementations and the X3D specification implementation. Due to this browser dependency, the X3D issues described in Section 2.5.1 have limited what can practically be done within the framework of this research. For example it was not possible to test a UNIX user base, simply because the X3D browser plug-in chosen did not have a UNIX version and no UNIX browser alternative existed that was capable of rendering large UML scenes.

To overcome the browser limitations a new implementation was created using OpenSceneGraph which is an open source OpenGL C++ 3D library. The advantage is that OpenSceneGraph supports cross-platform use and allows its implementation to be extended if required. Our intention was to create a cut-down specialised X3D-UML
viewer under our control. OpenSceneGraph does not directly support X3D. However, as our OpenSceneGraph implementation is a self contained viewer there is no need to implement X3D import/export to gain the research results required\(^{11}\).

The UML Diagram Interchange specification and both X3D-UML implementations are described in detail in the following sections. In Section 3.2.1 we first describe how we have modelled the relationship between the UML Diagram Interchange specification and 3D. This model is then used for creating the library for generating X3D through C# described in Section 3.2.2 and the complete X3D-UML OpenSceneGraph implementation is described in Section 3.2.3.

### 3.2.1 UML Diagram Interchange and 3D Extensions

The stated goal of the Diagram Interchange specification “is to enable a smooth and seamless exchange of documents compliant to the UML standard … between different software tools.” [68]. Unfortunately this specification has not been widely adopted, so the reality is that UML diagrams cannot be easily transferred between tools. Despite this, we have employed it in our research because it is a standard and it provides a means of defining UML diagrams and further defining any extensions required to represent X3D visualisations.

With the Diagram Interchange, the OMG (Object Management Group) provides the XMI (XML Metadata Interchange) model data for the specification [69]. This enables the specification to be imported into a tool (in this instance Sparx Systems Enterprise Architect[84]) and code generated directly from the model. This forms the basis of a UML diagram library, which can be adapted further for use with X3D. As an example, simply adding a “z” value to the existing Diagram Interchange “point” class enables any graph element to be given a depth position (as well as “x” and “y”). This “z” value can then be related to an aspect of the UML model not previously captured in the visualisation (such as hierarchy).

\(^{11}\) X3D capability has not been included purely due to programming effort. The OpenSceneGraph implementation uses the same Diagram Interchange structure as the C# implementation, so the programming effort would be in integrating the C# X3D export capability into our viewer.
Interestingly the original Diagram Interchange proposal suggested an optional Z value. However this did not appear in the final specification with the reason stated as ‘not included but desired in further development.’ [67]. Our implementation of X3D-UML not only provides this “desired” feature to UML but adds the possibility of extending the concept much further than simply layered diagrams. The text for this proposal is given below:

‘6.6.2 Layering of Diagrams. The proposal may address the presentation of graphical elements on different layers of the same diagram. This is distinct from 3D representation, in that it would be possible to show, or emphasize, sub-diagrams by displaying one or more layers of the total number of layers. In graphics parlance this is the concept of viewplane (or cells in the context of animation). It might be allowed that a 3D presentation could display connections between entities between two or more layers. In general, it would be expected that the layers represent logical collections within one diagram (or model). Layers are primarily a way to filter the display and simplify complexity.[67]’

The Diagram Interchange [DI] requires all implementations (including “2D” ones) to define the graphical representation of each graph element as these are not explicitly captured by the Diagram Interchange specification, ‘For example, the knowledge of drawing a class as a rectangle is not stored in the DI metamodel [68]’. To address this, we have extended and specialised the GraphEdge, GraphNode and Diagram classes to aid in managing the appropriate graphical representations. An example of this structure is given in Figure 3-1, where TransitionView is a specialisation of GraphEdge, and PseudoStateView, StateView and TextViews are specialised GraphNodes. This enables us to easily save and visualise elements based on their type, while still conforming to the DI specification.
The extended DI structure provides the basis for all the implementations detailed in the following sections. The structures for the C# and C++ implementations were both created with the aid of the Sparx Systems Enterprise Architect model.

### 3.2.2 X3D-UML C# Library Implementation

This section explains the C# implementation of the X3D-UML library, which is the first X3D-UML implementation used for usability tests. The first user group we studied was Microsoft Windows based and C# proved to be a good choice due to good XML support and ease of integration to the tool through the COM interface.

The C# implementation consists of two parts the “X3DUMLLib”, a library which contains the generic DI data structure with X3D-UML extensions as described in Section 3.2.1 above, and a C# executable specific to the UML tool being used for the research utilising the tools API via COM. This architecture separates the 3D visualisation data and logic from the tool specific logic, allowing the same library to be used across multiple tools.
The following code demonstrates a simple example of the X3DUMLLib. This code creates a single diagram with a single element and then generates the X3D file that can be passed to an X3D browser.

```csharp
using X3DUMLLib;

// Create an Instance of X3D-UML
X3DUMLLib.X3DUMLLib x3duml = new X3DUMLLib.X3DUMLLib();

// Set the main diagram
x3duml.theDiagram = new X3DUMLLib.DiagramView();

// Add element to the diagram
x3duml.theDiagram.contained.Add(
    new X3DUMLLib.PseudoStateView(
        X3DUMLLib.PseudoStateView.Stereotype.Initial, Width, Height, XPosition, YPosition, ZPosition));

// Generate an X3D file
x3duml.genX3DUrl("diagram.x3d");
```

For each view element type, we have internally defined the X3D representation. In the above example an “Initial PseudoState” is a black sphere scaled to represent a 3D dot. The `genX3DUrl` internally iterates through the diagram structure and adds each X3D representation to the scene. A fully functional program contains recursive functions to iterate through elements of a UML model via the tool, selecting the elements of interest and adding them to the diagram structure. The exact program logic and code structure is dependent on how the UML tool has been implemented and how it exposes its API.

This implementation approach has the following advantages and disadvantages:

**Advantages**

- X3D Browser implementation is used, so the only concern is the generation of X3D scene content.
- Visual Studio and C# greatly ease working with the UML tool API by making it easy to navigate the API functionality while coding through “intellisense”.

54
- The development complexity is divided into two separate problems: obtaining the UML data; and generating the X3D.

**Disadvantages**

- 3rd party X3D Browser implementations are not complete, and the lack of some features complicates the scene building.

- Tightly integrating the 3D UML view to the tool can be very difficult due to the view being a separate independent application.

Our implementation demonstrates that X3D is capable of describing and rendering UML notation effectively. The implementation then enabled user studies to be undertaken; however the results of those user studies showed that a more interactive diagram was required. As explained in the section on X3D Issues (2.5.1), X3D provides user interaction techniques, however combining techniques becomes very complex very quickly.

Due to the limitations imposed by 3rd party X3D browsers, it was decided to create a custom X3D UML implementation which would allow complete control over functionality. This new implementation is described in the next section.

### 3.2.3 X3D-UML OpenSceneGraph C++ Implementation

The results of our user research based on the C# implementation described above revealed that a highly interactive diagram was required. In addition to this, late in the research, a software engineering team of Linux-based users became available. Using the existing implementation became difficult due to limited X3D browser support on Linux and user interaction issues (as outlined in Section 2.5.1 X3D Issues).

As the previous X3D-UML implementation provided sufficient results around the capabilities and possibilities of the X3D standard, it was decided to create a specialised cross-platform viewer for X3D-UML. The new architecture is presented in Figure 3-2 where OpenSceneGraph provides a specialised X3D-UML viewer, coined the “Interactive Diagram”. The architecture is implemented using the UML Diagram Interchange and 3D Extensions (Section 3.2.1) by internally retrieving and storing UML
in a Diagram Interchange structure. Although not implemented, a fully integrated solution would export limited interaction X3D views and these could be generated as per the C# solution. Although not investigated an ideal solution would allow edit actions in the 3D diagram to feed back into the original model repository, as would be the case in fully integrated 2D diagrams. With the viewer under our control, this now becomes a reference implementation for X3D-UML. The open source project page can be found at http://www.x3d-uml.org/X3D-UML_Source_Code.

Figure 3-2 – X3D-UML OpenSceneGraph implementation. This implementation consists of a specialised viewer, which utilises a Diagram Interchange structure for storing and viewing UML.

One advantage of OpenSceneGraph is that because it is open source, it can be extended to have new capabilities specific to UML and 3D diagrams. To take advantage of this we structured our implementation to enable a flexible approach and provide benefits beyond those targeted at UML. The aspects of our implementation are described below:

**osgEdge:** OpenSceneGraph does not directly support diagram edges, and it only has a tree scene graph structure. We created new extended “edge” types to allow OpenSceneGraph geometric nodes, referred to as “geodes”, to be linked in 3D space
based on their geometric properties. The extended classes allow the creation of 3D graphs of any type and are described below:

**Edge** – This is a smart line which joins two nodes from centre to centre.

**EdgeTriangle** – This is a smart transparent triangle which joins two nodes. The tip of the triangle attached to centre of the first node and the base of the triangle to the perimeter of the last node.

**EdgeCallBack** – This is a callback method used by both edges to allow real-time animation. Geodes linked by the edges can be moved around a scene and the edges are automatically updated to stay attached.

**osgX3DUML:** As with the C# “X3DUMLLib” library, *osgX3DUML* is a C++ library which contains the generic Diagram Interchange data structure with 3D extensions as described in Section 3.2.1. Each “view” class type contains a geometric description, which can be displayed in an OpenSceneGraph viewer implementation (described next). As this implementation is designed to be highly interactive, a “DiagramInterface” has been provided to encapsulate common tasks undertaken with the Diagram Interchange data structure. The *DiagramInterface* also hides the internal implementation and allows elements to be called by name rather than iterating through the internal structure.

**OpenSceneGraph viewers:** OpenSceneGraph provides a viewer class which can load a scene and provide interaction techniques to the user. We have extended this concept by creating custom viewers for X3D-UML. For the Windows-based users we created an ActiveX viewer (x3dumlActiveX) and this enables us to fully integrate the X3D-UML view into the UML toolset. As *osgX3DUML* contains the platform independent interaction logic it is possible to create viewers for other platforms-based on other windowing mechanisms.
As an example of usage we present some typical code snippets for creating and managing an interactive diagram.

//to create a diagram, include the interface + create an instance
#include "DiagramInterface.h"

X3D::DiagramInterface* interactiveDiagram;
interactiveDiagram = new X3D::DiagramInterface();

//to add a diagram (this now becomes the "current diagram")
interactiveDiagram->AddDiagram("Top");

// to add elements to the "current diagram"
interactiveDiagram->AddStateView(Width,Height,X,Y,Z);

// to set the current diagram by name
interactiveDiagram->SetCurrentDiagram("Root.Top.");

// to create triangle edge between element and diagram
interactiveDiagram->LinkElementToDiagram(
  "Root.Top.2.StateView","Root.Top.Subdiagram1.");

// to render the scene
interactiveDiagram->RenderDiagram();
interactiveDiagram->RenderWorldEdges();

In Figure 3-3 we present the conceptual view of the internal diagram structure. Every element is contained within a single “IDiagram”. Each diagram can itself contain sub diagrams and diagrams can contain elements. Every element can be referred to by its fully qualified name which can be constructed from its position in the “IDiagram”. For example “Root.Top.Subdiagram1.0.StateView” refers to the StateView element of “Subdiagram1”, which in turn is a DiagramView element of the “Top” diagram, and so on.
The resulting OpenSceneGraph X3D-UML viewer that we have created has comparable performance to the best 3rd party X3D Browser and this is possible because we are able to purely focus on the performance needs of 3D UML users. Another crucial advantage of our X3D-UML viewer is further research is not hindered my licensing costs of high-end X3D browsers.

### 3.3 “Compilable” Visualisations

This section outlines the investigation path taken where X3D is used as both the UML visualise description of a software artefact and the container for the software artefact. This concept enables a single x3d file to be viewed as a visual object in an X3D browser and at the same time be compiled as a source code object. The motivation for this is that having the source code and visualisation as a single entity ensures a tight coupling between them. “Compilable” visualisations proved possible and did provide a tight coupling however they were not perceived to be scalable due to questions around how to visualise relationships between visual artefacts and the difficulty in automatically generating the visualisation. The concept and problems are described in detail below.

To create “compilable” visualisations the java source code to be visualised is embedded in an X3D script node. A script node is standard within X3D and is designed to carry source code information as CDATA (i.e. XML general character data). The purpose of the script node is normally reserved for embedded scripts executed at runtime by events.
routed to the node. However if no events are routed then the script node effectively becomes a source code storage node.

An example of such storage is demonstrated below, where the “HelloWorld” java code is embedded in the scene of a “Helloworld.x3d” X3D file. The other elements of the XML description are used in the standard way to describe visual aspects and these visualisations can be combined in a single system view by including a number of x3d files.

```xml
<Scene>
  <ExternProtoDeclare name="X3DUMLVisualisation" url="x3duml.x3d"/>
  <ProtoInstance name="X3DUMLVisualisation"/>
  <Script url="x3duml_java.js"><![CDATA[
    class HelloWorld {
      public static void main(String arguments) {
        System.out.println("Hello, world!");
      }
    }
  ]]>></Script>
</Scene>
```

To compile a “Helloworld.x3d” a java compiler wrapper called “x3dumlc” was created based on XML processing libraries. This wrapper pre-processes the *.x3d file to create a *.java file and then invokes the javac compiler. So the command “x3dumlc javac -verbose HelloWorld.x3d”, is translated to “javac -verbose HelloWorld.java”, where HelloWorld.java contains only the contents of the script node.

The “compilable” visualisations enabled the same file to be used in different contexts (i.e. as a visualisation or as source code), however coupling the actual visualisation to the source code proved difficult. The two main difficulties are described below and are the reasons this approach was not taken further:

1) In principle the visualisation should be generated from the contained source code as this maintains the tight coupling between the visualisation and the code i.e. if the code changed so would the visualisation. One possible way of achieving this would be to generate the visualisation at runtime by parsing the contained source code, however this proved impossible. Due to X3D security it was not possible for a browser to self-parse a
file from the operating system. X3D allows scripting nodes to access specific fields but not entire CDATA sections.

To counter this limitation we believe X3D needs to provide a specific “raw text” CDATA node where any arbitrary text can be stored and retrieved through appropriate fields. This would allow not only automated visualisations of source code but other data visualisations such as 3D spreadsheets.

2) An important part of the visualisation of source code is the visualisation of the relationships between source code artefacts. Although “compilable” visualisations treat classes as visual objects that can be inserted in a scene, additional processing of the source code scene is required to visualise the relationships. This requirement is non-trivial and is compounded by not being able to access source code information at runtime.

Many attempts to work around these problems have been investigated and possibility that some combination of web techniques could successfully bypass the standard security measures. For example one possibility might be to have all the parsing done on a remote web server, where browsers do have permission to retrieve files and execute scripts. Such a workaround may be fine for an open source web-based project; however for our research it created an undesirable dependency on external servers and complicated our implementation.

3.4 XSLT from JavaML to X3D-UML

This section outlines the investigation path where XSLT is used to translate XML source code representation into an X3D-UML visualisation. The motivation is that such visualisations could be integrated into the current software development workflow because the visualisation is derived directly from the source code. Source code is designed for explicit program specification, and by creating such a close coupling between the source code and the visualisation, it would ensure that as the source code changes, so does the visualisation. The difference between this implementation and the “compilable” visualisations (Section 3.3) is that we do not attempt to merge the code and visualisation into a single file; instead the visualisation is generated from the source code. Our implementation was successful and was used to verify the capabilities of X3D
for UML visualisation. However a more flexible approach was required to adapt to emerging visualisation requirements and the X3D-UML library was created as a replacement. This section therefore describes the method of generating static X3D-UML visualisations through XSLT and X3D prototypes.

The mechanism for extracting the java source code information to visualise is explained in JavaML (Java Markup Language)[9]. JavaML uses the Java compiler itself to generate an XML version of the code. By converting the code to JavaML it becomes easy to extract information regarding class, attribute and operation names. Badros [9] has previously demonstrated this by transforming JavaML into HTML through XSLT.

In translating XML data into X3D, Polys [78] demonstrated that XSLT can also transform one form of XML data, CML (Chemical Markup Language), into an X3D visualisation. Hetherington and Scott [40] also demonstrated that XSLT can be used to extend X3D by adding the dimension of time to a scene. For X3D-UML, XSLT is used in a similar way for the translation and JavaML is used as the source XML.

In this implementation X3D-UML can be consider as a user defined library which makes use of the X3D prototyping mechanism. The initial "X3D-UML.x3d" definition file contains a standard UML class view "X3DUML2DClassview" whose prototype has fields for name, attributes and operations. Based on these, the prototype generates the UML visualisation, with bordered compartments that display the elements.

Finally the software system can be represented by an X3D scene which includes, through the X3D inline feature, all the class X3D files. Through these steps a complete software system can be transformed from a textual representation into standard UML, existing in a fully immersive Web3D environment.
The steps taken in transforming Java code into X3D-UML are summarised below and graphically depicted in Figure 3-4.

1. Java source code is transformed into JavaML

2. XSLT is used to extract information from JavaML and to generate an X3D file, which contains information pertaining to the UML visualisation.

3. SoftwareSystem.x3d contains the classes of a system inlined (included via the X3D referencing mechanism) into an X3D world. Once loaded the class visualisations are displayed as defined within X3D-UML.x3d.

At this point it is important to note that each step is independent and open. The transformation process is designed to allow evolution of visualisation outside the control of any one tool. Java could be replaced with another programming language or another abstract representation, as long as the appropriate information can be extracted. Different aspects of information can be visualised through changing the XSLT file, and different visualisations can be applied to the prototype. At the SoftwareSystem.x3d level, users are open to changing layout and introducing additional functionality.

![Diagram](image)

**Figure 3-4 – Transforming Java to JavaML to X3D-UML**
Through this process source code can be visualised in a standard way through X3D-UML. For example a single class (as show in Figure 3-5) can be instantly recognisable as UML. The only visual clue to its new environment is the perspective view given when rotated. The class displayed has all the benefits of any other object represented in X3D. It can be viewed via the web; actions can be associated with it when it is touched; it can be displayed in a multi-user environment; different users can view it from different perspectives and it can be animated.

![Image of FirstApplet.x3d](image)

*Figure 3-5 – FirstApplet.x3d displayed through an X3D browser demonstrating the display quality of class visualisation*.

The major benefit of UML is when it comes to comprehending large software systems. Even with no form of UML representation, it would not be difficult to understand the function of a single class, with only one operation. Our X3D-UML class can be combined with other nodes in a scene to create a more complex system. One benefit of this new environment can be to render entire software systems, but for this to be usable we need to render systems as large as those found in reality.

Our research found that Bitmanagement’s *BS Contact* (6.2 or greater), with the performance option set to “Use textures for text” was capable of rendering such scenes. An example of a large existing software system is presented in Figure 3-6. This provides a visualisation of the complete set of Java 3D classes, from the Sun’s Java 3D Core API code base [48]. Using the open source project it was possible to generate this visualisation directly from the source code and demonstrate that, without the aid of

12 A video example is available at [www.x3d-uml.org/YouTube_Examples](http://www.x3d-uml.org/YouTube_Examples)
scene optimisation techniques, over 700 UML classes can be rendered and navigated effectively using a Web3D environment. Prior to version 6.2 of *BS Contact*, our experiments showed that only at most 200 UML classes could be rendered and navigated effectively. Further external research into this area, using a derived technique, shows that more than 2000 classes can be visualised [4].

![Figure 3-6 – Classes from the Java 3D packages shown as UML through an X3D browser demonstrating the display quality of class visualisation. Classes are grouped in planes based on the package (directory structure) in which they are found. GVector (front) is from the vecmath package, the other planes show j3d-core-utils](image)

Although only simple UML has been used (as yet no associations have been visualised) and only simple X3D features, we can see that web-based projects such as Java 3D could benefit from UML, through X3D-UML visualisations. The benefit is in “web

---

13 A video example is available at [www.x3d-uml.org/YouTube_Examples](http://www.x3d-uml.org/YouTube_Examples)
browsing” a complete project as UML, without needing to download the source code and import it into a specialised tool. Through user-centred design, this visualisation could be improved and adapted to meet user and usability needs for software engineering tasks.

Furthermore, this demonstrates that X3D is able to effectively render large software systems of the sizes found in real systems, as UML. We are able to visualise a software system of many hundreds of classes without any optimisation techniques, such as partial scene loading. Although in this instance class diagrams are visualised, the results are applicable to all notations requiring large amounts of unique text and lines.

Not only is X3D capable of presenting the data, it is able to do so in a way that will facilitate exploration into advanced UML visualisation. This implementation has demonstrated that XSLT can be used to transform one form of XML source code information into X3D-UML. XSLT can therefore be used in a similar fashion with other XML data such as XMI. As the X3D-UML visualisation is merely a user defined X3D type, there are no restrictions on redefining the visualisation to explore new areas.

The reason this implementation has not been pursued further was because, although it achieved the task of creating a visualisation in a structured way, it was difficult as a means of creating prototypes. The research required a way of quickly generating different types of visualisation based on user needs. The XSLT approach made use of more advanced X3D features, which by the nature of X3D were more defect prone. The decoupled nature of the process also made debugging harder, for example script nodes were used for layout of class borders and these could only be debugged by trial and error.

Due to the difficulty described above and the fact that the first test user base was using a UML tool with an API to access the model data directly. It was decided to create a specialised tool that read the model data and generated the preformatted X3D directly, without the use of secondary prototyping or scripts. The XSLT approach is still valid, however it is more suited to areas where the generated visualisation requirements are already known and clearly defined.
3.5 Summary

In this chapter we have presented a number of implementations which enable the realisation of 3D UML visualisations. These implementations enable researchers to focus on evaluating the use of 3D UML rather spending unnecessary effort on areas of 3D computer graphics which may not work\(^\text{14}\). As evidence of this, aspects of this work have already be employed by Anslow [4] to generate 3D UML visualisations. The X3D-UML library presented will undergo continued refinement as an open source project (http://code.google.com/p/x3d-uml/). It is intended that X3D-UML will provide the foundation for future research where high performance 3D rendering of UML is required.

\(^{14}\) We have not reported on the many failed implementations. As an example of the seriousness of the issues encountered in this phase of the research, in 2004 our conclusion was X3D could not be used for UML and we were spending research effort quantifying the issues as to why. It was only performance improvements in X3D browsers and researching how to make use of those improvements that we could return our focus on research into the usability of 3D UML.
3D Evaluation of Human Development in the Womb.
Image of Max McIntosh at week 27 derived from ultrasound data.

4.1 Overview

This chapter describes the research methodology applied to evaluating 3D UML visualisations. The methodology is one of the contributions of this research as it describes a refined methodology specifically for evaluating 3D UML interfaces. The methodology has been refined through application to our research and through a workshop session at the 2nd Workshop on the Layout of (Software) Engineering Diagrams [60].

The intent of this methodology is to gain empirical evidence as to the benefit of 3D UML visualisations within the context of actual users and user tasks. This contribution is significant in that empirical studies involving actual users and user tasks in software engineering are rare. Combined with the complexity of creating 3D visualisations, research in this area requires a specialised methodology to optimise results.
In this chapter we describe the initial refinement of *Sequential Evaluation* as first applied to our research in Section 4.2. We then detail the results of practical application and further refinements to the methodology in Section 4.3. The *Sequential Evaluation* approach is a multi-method approach, using a combination of four usability techniques. Each method refined in this approach is covered separately in sections 4.4 to 4.8.

### 4.2 3D UML Sequential Evaluation in Theory

This section presents the *3D UML Sequential Evaluation* approach and how we have applied it to the problem of evaluating 3D UML diagrams. This information presents the methodology as first refined from the literature and represents the initial application of the methodology. The subsequent Section 4.3 presents the final refined evaluation approach based on experiences throughout this research.

The *Sequential Evaluation* is a methodology first defined by Gabbard et al. [32] and the varied slightly by Bowman et al. [14]. The methodology aim is to create a ‘*useable and useful interface prototype*’ utilising a combination of qualitative and quantitative techniques, ordered in a sequence designed to enhance the quantity and quality of results produced from limited time and budget. The concept is to use low cost and generic techniques first and follow these up with more costly but precise usability techniques. The methodology therefore uncovers obvious problems early and the design can easily be refined before more extensive testing is carried out.

In this section we present our refinement to *Sequential Evaluation* and cover in-depth each step as it pertains to applying a 3D extension to an existing UML diagram. An overview of our variation of *Sequential Evaluation* is shown in Figure 4-1. Although one goal is to create a ‘*useable and useful interface prototype*’, our approach differs in that the main goal is to answer a given research question and to capture qualitative and quantitative results at all stages. The areas of 3D UML studied are untested, so we cannot make the assumption that a ‘*useable and useful interface prototype*’ is possible. Therefore we require a methodology which supports capturing evidence as to the reasons a 3D UML visualisation may not be “*useable and useful*”.

69
Within this methodology we also adhere to our X3D-UML constraints stated in Table 2-1 of *Actual Data, Actual User Tasks, UML, X3D, Standard PC Hardware and Development Environment Imitation*. The intent is to undertake the research in the context of existing environments, work practices and standards.

![Figure 4-1 — Sequential Evaluation methodology as we have refined it for researching advanced 3D UML visualisation through X3D.](image)

Firstly we present an overview using a textual non-3D extension called “chromacoding”, so that examples of how the methodology is applied can be given against a familiar software visualisation extension.
4.2.1 Sequential Evaluation Chromacoding Example

In this section the analogy of “chromacoding”, a non-3D visualisation technique, is used as a presentation aid for how we have executed the research methodology. Chromacoding is a technique of applying colours to source code text to help the software developer more quickly interpret the code. An example is shown in Figure 4-2 where comments appear in green text, operators in black, and keywords in blue. In this instance chromacoding can be thought of as a visualisation technique which extends the current software engineer’s textual working environment. As is the case when basing 3D software visualisation extensions on UML, with chromacoding the user is not required to learn a new language or a completely new way of working, because they are simply presented with an evolution to what they are already familiar with. With the benefits of hindsight, there is no doubt that chromacoding is a valuable aid to software engineers and has now become a feature of almost all present day software integrated development environments (IDE’s).

```csharp
using MindTouch.Deki.Logic;

namespace MindTouch.Deki
{
    using File = System.IO.File;
    using RFC_File = MindTouch.Deki.File;
    using HttpPostedFile = MindTouch.Dream.Http.HttpPostedFile;

    public class DekiContext {
        //--- Class Fields ---
        //--- Class Properties ---
        public static DekiContext Current {
            get {
                DekiContext dc = CurrentOrNull;
                if (dc == null) {
                    throw new InvalidOperationException("DekiContext.Current is not set");
                }
                return dc;
            }
        }
    }
```

*Figure 4-2 - Example of chromacoding of C# source code in Visual Studio. In this example comments appear in green text, strings in red, operators and expressions in black and keywords in dark blue.*

Making use of the chromacoding analogy, the following section walks through the Sequential Evaluation process as applied to X3D-UML:
The process starts with the research question; for example: “Is there benefit in 3D state machine diagrams for software engineering tasks?” For the chromacoding example analogy, one question could have been “Is there benefit in colourising source code for software development?”

(1) In the User Task Analysis stage, we first capture, through user sessions, the goals of the users and the tasks they do to achieve those goals in their current environment. Goals may be to “make change to existing software system” and tasks may be to “change code; check code by compiling; fix errors and warnings; repeat until changes are complete”. Quantitative results might show “Users undertake 5 compilations an hour on average across 10 users”, Qualitative results might show “software engineers like to compile often to flush out errors early but avoid this if compilation times are too long”.

(2) In the Heuristic Evaluation stage, with the information from User Task Analysis the initial extension is created based on the research question. Experts are called on to review the implementation before involving users, where possible guidelines are to be followed in the review. Quantitative results might show “The implementation meets all guidelines for colour combinations in text as per the graphic design handbook X”. Qualitative results might show “Evaluator’s suggested the actual source code text should be made more prominent to the user than the source code comments text”.

(3) In the Formative Evaluation stage, the initial extension has been refined with the help of information from Heuristic Evaluation and users are called on to review the implementation. Quantitative results might show “Users undertake 3 compilations an hour on average across 10 users”. Qualitative results might show “User stated that comments looked cleaner a lighter colour and did not clutter the code”.

(4) In the Summative Evaluation stage the extension has been refined with the help of information from the Formative Evaluation, and users are called on to test the implementation against the existing process. The users are asked to complete a set of tasks associated with a goal defined from User Task Analysis. Quantitative results might show “Users with chromacoding completed the task 20% quicker and with 10% more source code comments” and “90% of polled users preferred chromacoding”. Qualitative results might show “Users expressed it was quicker to understand the code they were
reviewing and they felt more confident with not compiling to flush out errors” or “users would like to choose their own colour scheme”.

The above example shows that by systematically applying the Sequential Evaluation process we can demonstrate that there is a significant benefit in the chromacoding extension. Also the measured benefit is not left “dangling” by being measured against a task of unknown importance and we have produced further quantitative and qualitative evidence which validates the answer. Time has not been lost by undertaking expensive user testing on a design that might have failed basic guidelines and new benefits and guidelines can be discovered and explained along the way.

We also note that if problems occur, then it is possible to reiterate through steps. If users found the implemented colour scheme distracting, we are able run through another evaluation pass where users may be able to choose their own colour scheme.

Above is an example of the overall Sequential Evaluation process, and in the next sections, we present the detail of each stage in relation to our 3D UML diagrams.

### 4.2.2 User Task Analysis

The process within the User Task Analysis stage has been derived from Hackos and Redish [36], who outline the steps to gather effective user data and measure the effectiveness of a design against that data. Before we can proceed to test for benefit we need to know what goals the user wishes to achieve. Does the user even use the diagram we wish to extend? If they do, why do the use it? When they use it how do they use it? Through this analysis we can determine the performance metrics that we will finally test against. We can also find common and critical tasks that users perform and focus on those.

As the context of the user task analysis is confined to a particular diagram and users are from broad backgrounds, “Usability Evaluation” sessions have been chosen to be the most appropriate means of gathering user information. This is due to limited availability of participants and complications with negotiating direct access to participants in their work environment. User Evaluation sessions are sessions undertaken with the researcher and the user in a location away from the work/study environment. In addition to this, user profiles are captured with a questionnaire prior to the user session.
Hackos and Redish [36] suggest that as part of planning for user testing, one should define the issues and objectives. For UML, the general issues we are trying to resolve and final objectives of user sessions are listed below:

Issues:

- What goals do users have that trigger them to use the UML diagram?
- What percentage of their work is related to those goals?
- What tasks do users do with the diagram as a result of those goals?
- How similar are the goals and tasks across teams of users?
- Are the users creating another mental model when using the diagram?
- Do users get frustrated with the current diagram, if so, what are the issues?

Objectives:

- List of user goals across a range of users pertaining to a particular diagram
- A task list and scenarios for each goal
- Weighting on the importance of each task and goal
- Indicative performance metrics for individual tasks and/or goals
- Representative User Task Scenarios (the most important goals and tasks)
- An understanding of the user’s mental model vs. the conceptual model.

To support User Task Analysis, metrics are gathered directly from existing UML models to provide quantitative evidence as to the extent of a particular problem or provide evidence to support further investigation. For example, if a particular aspect of a diagram is frustrating a number of users, the model metrics can uncover how many such diagrams exist containing that aspect.
4.2.3 Heuristic Evaluation

Heuristic Evaluation is a “rule of thumb” evaluation by expert evaluators to provide an initial “sanity check” of the visualisation for use with the representative user task scenarios captured in the User Task Analysis stage. The X3D-UML visualisation is presented for feedback, where possible, using documented guidelines and heuristics, to expert evaluators defined below. Due to the novelty of 3D UML, it is not expected that the expert evaluators will be able to provide specific feedback (i.e. expert comparison against 3D UML guidelines, of which none exist), however it is expected that either UML or 3D expertise will be applied to give valuable first round feedback.

Expert Evaluators

The following groups of people have been identified and categorised as Expert Evaluators:

- Tool Specialists (Evaluators with direct experience with the 2D UML tool being used for the 3D comparison)
  - Consultants
  - Developers
  - Technical Support

- Expert Users (Evaluators with years of experience using the 2D UML tool being used for the 3D comparison)

- General UML Experts
  - Consultants
  - Expert Users of other tools
  - Teachers

- Researchers and Developers in the area of 3D User Interface Design

- X3D Experts
• General User Interface Experts

Guidelines and Heuristics

There are no documented guidelines for how to present UML diagrams using 3D. There are however a number of other sources of guidelines and heuristics for UML and 3D which could be utilised (listed below). While soliciting feedback from Expert Evaluators we will request the source of any guidelines they may have used. These guidelines can also be used to invalidate the use of an unusual, but possible 2D diagram. For example a very large 2D diagram may present similar information to a 3D diagram, but existing guidelines may explicitly discourage this because such diagrams are known to be less useable.

Here is a list of possible guideline resources:

• UML Style Guidelines, such as Ambler [3]
• Toolset Guidelines provided in training and documentation
• 3D UI Guidelines
• Emergent Guidelines (specific 3D UML guidelines evolving from this research)
• X3D Guidelines
• Personal Expert Evaluator “rule of thumb”

4.2.4 Formative Evaluation

Formative Evaluation involves getting feedback from users as the visualisation evolves prior to it undergoing direct comparison tests (in Summative Evaluation). The Researcher will work with users to walk through representative user task scenarios captured in User Task Analysis. This step is to gain first hand understanding of issues pertaining to the visualisation and its use. This step also uncovers exact areas where more testing should be focused and in itself can provide initial test results (i.e. simple timed tests could be undertaken).
4.2.5 Summative Evaluation

*Summative Evaluation* will directly compare the 2D and 3D extension by testing performance of the final visualisation against representative user task scenarios captured in *User Task Analysis*. Manual and automated metric gathering will be put in place and statistical analysis used to determine real world benefit in terms of productivity measures. *User Task Analysis* will determine the specific metrics to be measured based on the actual variables needed to complete tasks. Although the variables to be measured cannot be determined without *User Task Analysis*, examples of possible metrics are given below:

- **Time** – Time to complete tasks defined from *User Task Analysis*
- **Accuracy** – The number of errors in completed tasks
- **Efficiency** – Formula based on time and mistakes
- **Comprehension** – Tests which require information that is available in both a single 2D and a single 3D diagrams to be answered.
- **Emergent Knowledge** – Tests which require information that is available in a single 3D diagram but requires combinations of 2D diagrams to be answered.
- **User Perceived Comparison** – Post hoc questionnaires where users rate ease of use of each visualisation

4.3 3D UML Sequential Evaluation in Practice

In the previous section we presented our initial refinement of *3D UML Sequential Evaluation* for application to our research. The following sections describe the results of applying the *3D UML Sequential Evaluation* technique to actual 3D UML diagrams and users, and further refinements. Our results are derived mostly from applying the evaluation technique to 3D UML state machine diagrams (the most extensively investigated diagram).

We present each phase of *3D UML Sequential Evaluation* in its own section. For each phase we describe the general outcome, the issues observed and the refinement to
address those issues. In the area of *Heuristic Evaluation*, extensive refinement was undertaken in a workshop session at the 2nd Workshop on the Layout of (Software) Engineering Diagrams [60] and this is covered in *Heuristic Evaluation Workshop* (Section 4.6).

### 4.4 User Task Analysis

We applied the *User Task Analysis* for 3D UML as described in Section 4.2.2. The results gathered through user task analysis were valuable and met the research objectives of understanding current processes. Our addition of gathering of model metrics through scripts was found to be a particularly powerful tool for gathering accurate quantitative data. Through scripts and email, anonymous data could be quickly gathered from many companies. Recipients simply ran the script and returned the email with the data, with the results representing years of UML use from large teams of developers on diverse products.

#### 4.4.1 Issues

This section lists the issues that arose from applying *User Task Analysis* as theoretically defined in Section 4.2.2.

**Off-Site “Usability Evaluation” Incorrect Method**

The original intent was to undertake “Usability Evaluation” sessions away from the workplace. It was thought that participants were more likely partake in the research if it was disassociated from their current work (and therefore avoiding confidentiality issues). In practice though, participants were more forthcoming when they did not have to leave their desk. Also company managers were open to on-site research if the impact was low and they saw some company benefit from the research.

Once on-site it became apparent that off-site session would not provide the complete information required for effective user task analysis of UML usage. Although the UML usage should be generic to “any UML” on “any machine” in “any environment”, the user’s current environment provided vital evidence regarding the problems with the status quo. One example of this was the use of pen and paper diagrams or other means of overcoming problems with UML, which are scattered around the desk of the participant.
If the research was undertaken in another environment, these artefacts and the discussions they generated would have been missed.

Another issue was the participants’ perception of their work and the reality. Obtaining a true indication of work practices could only be derived from direct observation. One example of this is a user claiming they did no debugging using the diagram, however direct observation showed that although they did not use the diagram debugging feature provided with the tool, they did use the diagram for navigating code while debugging.

**Low participation rate**

Low participation rate is a common problem, though not easily be solved, it was reduced by undertaking the research on-site. In an open office environment, once others saw that the impact was low, they more readily offered to participate. Also opportunistic participation happened, an example being one participant volunteering because they were blocked from their normal work and had spare time at that moment.

**Closed Vs Open Questions**

In such an early stage of understanding the problems, it proved difficult to capture quantitative measures of participants’ views through closed questionnaires due to the variability of the areas. For example the question “*Please rate from 1 to 5 how you feel UML helps you in your role (1 = Not at all, 5 = Essential)*”, users may consider UML is imposed on them through a particular development tool and therefore the tool dictates that UML is “Essential”. The intent of the question was to determine if the user had a bias towards UML, however the answer given was more related to their views on the tool. Changing the question to be an open discussion revealed more interesting results, such as diagrams being thought “Essential”, UML “Good” as a diagram and the tools for UML implementation were generally considered as “Average”.

A similar problem was found in trying to quantify the users’ thoughts on 3D through closed questions. Experience varied greatly and results were not clear cut. The intent of the questions was to determine if users had bias towards 3D and if they were already comfortable with navigation. However indirect closed questions related to 3D applications to ascertain this were not successful. For example the question, ‘*Please list any notable 3D software applications that you have had experience with in the past or*
3D experience in the last 2 years?’, was too vague, hard to answer and missed essential information about 3D experience prior to the period. One participant spent a large amount of effort developing a 3D interface prior to the 2 year time frame.

4.4.2 Refinement

This section lists further refinements to User Task Analysis (Section 4.2.2). These can be summarised as: restricting research to work environments and to relaxing the use of closed questionnaires at the start of the research process. These points are detailed further below:

It is essential to undertake User Task Analysis in the work environment of the UML user and observe the user as they work (as per Hackos and Redish [36] recommendations). The UML tool only makes up part of the users environment and it is evidence found through watching the participant work and the other parts of their environment that provide the most accurate and valuable information. We were looking for limitations in the UML notation which 3D may address, therefore the tool usage alone will not show what the UML can not do. It is the workarounds existing outside of the tool, such as pen and paper drawings or unusual user habits, which capture specific UML problems that need to be addressed and provide insight to possible solutions.

Based on our results, we extend the issues/objectives listed in Section 4.2.2 to also consider:

- What “out of diagram” effort is done in areas associated with the diagram?
- Does the participant have a log book with sketches?
- How have these sketches changed over time from first learning the tool?
- What do the sketches enable the participant to do that could not be achieved by the UML tool?
- Are there periods of time where the participant is thinking about a diagram?
- What are they thinking and is it for long periods of time?
What are the things that frustrate the participant about the whole process (not simply the diagrammatic representation)?

We found that a list rather than a formal questionnaire was more useful for gathering information, and the issues and objectives (Section 4.2.2) themselves can be used as discussion points. For the User Task Analysis stage, the number of participants is likely to be small and so will not provide an opportunity to obtain statistical conclusions from a questionnaire. However, open questions provide information to enable a more accurate questionnaire to be developed later, should it be possible to undergo a survey or user testing of a larger population later in the research.

4.5 Heuristic Evaluation

We applied Heuristic Evaluation as described in Section 4.2.3 providing a 3D UML prototype to experts to evaluate. In addition to the solution evaluation, evaluators were asked to comment on our user task analysis results. In general, Heuristic Evaluation proved a valuable stage and highlighted many usability issues with the prototype 3D UML solution. These issues were able to be uncovered with relatively little effort and rectified before involving users. The expert evaluator opinions on the validity of user tasks also were proven to be very valuable, with expert evaluators being able to comment on direct experiences with hundreds of users from different companies and countries.

4.5.1 Issues

Expert evaluators were senior software engineers and scientists and were found to be very busy and had little time to contribute towards an evaluation. For the state machine diagram evaluation, 9 expert evaluators volunteered to participate but only 2 completed the evaluation. Two of the non-participating volunteers, spoken to at a later date, cited lack of time as the reason for not contributing.

Expert evaluators were not specifically usability experts and therefore are drawing conclusions mostly from their personal experiences rather than methodically applying usability guidelines. Experts were asked to suggest any guidelines they did use but none were forthcoming. It would be ideal to have usability experts, with in-depth knowledge of
UML and the tool being extended, however the likelihood of locating a pool of such experts is not promising.

4.5.2 Refinement

The refinement to this heuristic evaluation has been significant. For this refinement, heuristics guidelines were tested in a session at the Layout in Software Engineering Diagrams Workshop [55] using two 3D UML approaches. The details of the workshop session and refinement are covered in the next section.

4.6 Heuristic Evaluation Workshop

From the experiences of applying Heuristic Evaluation (Section 4.5) it was apparent that a more structured approach to applying this evaluation technique was required. The need for this structured approach is to aid “time poor” evaluators to review visualisations quickly, while thinking more about the problem from a usability guidelines perspective.

The initial direction in defining a structured approach was to gather together researchers that had experiences with 3D UML visualisation and collaborate on a heuristic guidelines paper for the Layout in Software Engineering Diagrams Workshop (LED’08) [55]. This workshop was associated with the 2008 IEEE Symposium on Visual Languages and Human Centric Computing [92] as part of Visual Week 2008. There are no guidelines for 3D UML, so this was an opportunity for a number of researchers to define heuristics based on user experiences with their tools.

Although there have been a number of 3D UML visualisations (covered in Section 2.3.1) many have not been continued. These researchers were contacted, however most did not respond or could not contribute due to work commitments. This left only one GEF3D [33] (researcher Jens von Pilgrim) and X3D-UML, which did not provide a sufficient number of alternative views and experiences to review guidelines for general 3D UML.

LED’08 did however provide an excellent opportunity to present heuristic guidelines to researchers in the layout of diagrams to evaluate. LED’08 traditionally have a workshop challenge for which we created the 3D UML Heuristic Challenge [60]. The goal of the 3D UML Heuristic Challenge was to workshop 10 well known heuristics using UML diagrams and 3D layout techniques. The challenge was to identify the best performing
heuristics and the best performing diagram layouts. The result of the workshop provided information in the form of reviewed guidelines that aid both 3D UML evaluators and implementers to assess the usability aspects of diagrams\textsuperscript{15}.

In the next Section 4.6.1 we will discuss the ten heuristics and how the were chosen. Then in sections 4.6.2-4.6.4 we present the structure of the workshop, the layout problems given to the participants and the challenge tasks. In sections 4.6.5-4.6.8 we review the results of each of the layout problems and the heuristics evaluated, including test workshops. Finally we present a summary of results in Section 4.6.9 with the resulting refinement in Section 4.6.10.

4.6.1 Ten Heuristics

For the workshop, 10 heuristic guidelines were chosen from a literature review. The body of work related to heuristics is extremely large, which made a comprehensive literature review impractical. For example Nielson’s [65] online heuristic evaluation resource states the number of web pages related to heuristic evaluation was 58,000 in 2005 and growing quickly.

In addition to the large quantity of heuristics available, the heuristics were found to be very generic and could be interpreted in many ways. For example, if we were to evaluate a UML diagram of a Java based Swiss banking system against the heuristic “Speak the User’s Language” [62]. The “User’s language” could be interpreted as “Swiss”, “Financial Symbols”, “UML notation” or “Java” depending on the evaluators interpretation of the user and the diagrams intent. Furthermore, heuristics give no strict criteria for measurement of compliance and due to the quantity of heuristics and the “openness to interpretation”, many heuristics overlap.

With the above in mind the heuristics were chosen from well recognised sources and selected based on our experiences with 3D UML, using our understanding of the user and the intent of the diagram. For each visualisation (GEF3D and X3D-UML), the most important heuristics were chosen by the researcher and then a combination of these heuristics was selected that were thought to be appropriate for general 3D UML.

\textsuperscript{15} This workshop is the result of collaboration with Jens von Pilgrim, Lehrgebiet Software Engineering, FernUniversität in Hagen, Germany. (Jens.vonPilgrim@FernUni-Hagen.de). This thesis documents the X3D-UML perspective of the workshop results.
For X3D-UML, heuristics were based on experiences with user studies relating to 3D UML state machine diagrams. The most appropriate heuristics were found in Shneiderman’s [83] “Visual Information Seeking Mantra”, as these reflected aspects of user needs and general areas in which problems were reported with the 3D UML state machine diagrams. Shneiderman’s mantra related to seven tasks is: Overview, Zoom, Filter, Details-on-demand, Relate, History and Extract.

The mantra is an ordered list however for the purposes of the workshop the guidelines were presented as individual items to be evaluated. In addition similar “related” guidelines where the interpretation overlapped were also included. The following are the combination of guidelines [2, 62, 83, 87] as presented to the workshop:

1. **Speak the User’s Language**

   ‘The dialogue should be expressed clearly in words, phrases, and concepts familiar to the user rather than in system-oriented terms.[62]’

2. **Overview**

   ‘Gain an overview of the entire collection. Overview strategies include zoomed out views of each data type to see the entire collection plus an adjoining detail view [83].’

3. **Be Consistent**

   ‘Users should not have to wonder whether different words, situations, or actions mean the same thing. [62]’

   Related: ‘Show Data Variation, Not Design Variation [87]’

4. **Zoom**

   ‘Zoom in on items of interest. [...] Smooth zooming helps users preserve their sense of position and context. [83]’

5. **Relate**

   ‘View relationships among items. [83]’
Related: ‘Rationale-Based Tasks’ (‘Concretize Relationships’ and ‘Formulate Cause And Effect’) [2]

‘Users need to be able to relate data sets to the realms in which decisions are being made.[2]’

‘[A] system can help bridge the Rationale Gap by clearly presenting what comprises the representation of a relationship, and present concrete outcomes where appropriate.[2]’

6. Minimize the User’s Memory Load

‘The user should not have to remember information from one part of the dialogue to another.[62]’

7. Filter

‘Filter out uninteresting items. Dynamic queries applied to the items in the collection is one of the key ideas in information visualization.[83]’

8. Data-Ink Maximization, Data Density

‘Graphical excellence is that which gives to the viewer the greatest number of ideas in the shortest time with the least ink in the smallest space. [87]’

Related: ‘Simple and Natural Dialogue [62]’

‘Dialogues should not contain irrelevant or rarely needed information. Every extraneous unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.[62]’

9. Details on Demand

‘Select an item or group and get details when needed.[83]’

10. History

‘Keep a history of actions to support undo, replay, and progressive refinement. It is rare that a single user action produces the desired outcome.[83]’
The above 10 heuristics are the basis for workshop session described in detail in the next sections.

4.6.2 Workshop Structure

This section details the LED’08 workshop 3D UML Heuristics Challenge agenda and how it was presented to participants.

Workshop Agenda

The agenda for the workshop session starts with participants forming 4 groups, consisting of 2 groups undertaking layout tasks and 2 groups undertaking heuristic tasks. The workshop is presented with background for the challenge and then groups select a number of heuristics to test in different ways against two different UML diagram types. An overview of each section of the challenge and the time is given below:

Introduction, 15 minutes: The workshop forms groups of layout or heuristic teams. The workshop participants are presented with an overview of Heuristic Evaluation and the list of 10 heuristics used for the challenge.

Layout Problem, 10 minutes: Two types of traditional complex UML diagrams, state machine and translated class diagrams, are presented. Also presented is the concept that sets of diagrams, within each type, are related and that 3D can be used to layout a set of diagrams to improve usability.

Group Challenge, 45 minutes: Each group is given their “evaluation pack” described below. Layout teams select 2-3 unique guidelines to create diagram layouts against. Heuristics teams explore all guidelines applied to existing diagram layouts. The groups prioritise each heuristic and have a fixed time to complete the challenge.

Results, 20 minutes: Each group then talks through their diagram layout and how the heuristics performed. The workshop as a whole, discuss the best heuristics and best performing diagrams.
Evaluation Pack

The evaluation pack contains material to enable groups to create or evaluate 3D UML diagram layout. A 3D UML environment does not exist to test diagram layouts, so groups will use pen, paper etc. to create representative layouts that could exist in a 3D environment. The evaluation pack contains the following:

- Formatted Worksheets and cards (to write down heuristics and results):
  - Heuristics questionnaire
  - Heuristic evaluation worksheet (for heuristics team)
  - Task descriptions (for layout teams with state machine diagram and transformation chain)
- Pens, Paper and Post-it notes

Heuristic Evaluation Presentation

The presentation of material to the workshop participants was done in a structured way to reduce any influence on the results. The workshop is first presented with the agenda and goals of the challenge. Then all 10 heuristics were presented in full, with the aim to have all participants fully aware of all heuristics and to not need to study each item at the start of the challenge.

Next the participants are presented with the challenge of applying the heuristics. What do the guidelines really mean? Can this description be refined for UML? What criteria can they be measured against? Is the guideline relevant? What is the motivation or user task?

Finally the participants are presented with the UML problems that could be solved with 3D layout techniques. At no point prior or during the presentation are participants shown the X3D-UML or GEF3D solutions. These solutions are reserved for the evaluation groups, once they have completed their guideline rating task. The following section presents the 3D UML layout problems.
4.6.3 3D UML Layout Problems

This section presents two examples of traditional complicated UML state machine and translated class diagrams. These examples are exactly as presented to participants and subsequently tested in the 3D UML heuristic challenge [60]. To enable participants to focus on a particular user and problem associated with each diagram problem, the “Motivation and User Tasks” are defined at the end of this section.

The two types of UML diagrams represent examples of sets of strongly related diagrams, where the combination of these diagrams in a single view provides valuable information to the user. However to layout multiple diagrams, to show their relationships, present a problem because of the limitations of traditional 2D representation. To remove the 2D limitations 3D can be used, which enables new and novel diagram layout techniques.

State Machine Diagram Example

In this example we present the UML state machine diagrams that are to be tested using 3D layout techniques against chosen heuristics. The state machine being tested consists of a hierarchal structure with a top state diagram and separate substate diagrams.

These diagrams are from RoseRT, which is a tool that generates code from state machines and is targeted at the real time embedded market. The “TrafficLights” model\textsuperscript{16} is an example model provided with the toolset for training purposes. The model demonstrates the use of state machine inheritance to produce different variations of behaviour for different countries (e.g. North America and Austria). This model is the most complex provided in terms of hierarchy, with the Austrian variant containing 6 related state machine diagrams. Although it is considered complex in terms of the examples, actual models surveyed in industry are much more complex and are shown to have as many as 19 diagrams in a single state machine.

In Figure 4-3 we see how the structure is currently presented to the user. RoseRT presents these diagrams in a tabbed window approach, where substates are navigated to by double clicking on the diagram. With the use of 3D layout techniques, these

\textsuperscript{16} The “TrafficLights” model is a UML model of a traffic light system. This is a model provided with RoseRT to demonstrate state machine inheritance through the example of traffic lights designed to work in different countries.
diagrams can be presented in a more intuitive way and score better results in a heuristic evaluation. A view, called the navigator, shows the hierarchy of individual states (not state machine diagrams) and gives some clue of the layout of the state machines diagrams.

Visualization hints:

- In the evaluation pack, the name at the top of a state machine diagram is its substate name. This indicates how it relates to another diagram element, on another diagram. The name "Top" indicates that it is the top level state machine diagram.

- Junction points (black dots on the state edge) which appear on the substate element (i.e. within a diagram), also appear on the edge of the diagram representing the complete substate. Note: These junction points generally appear in the same location at both levels of diagram, but not always.

- When considering the behaviour of a state machine, how events trigger transitions is very important. Events that are not handled at a substate level (i.e. do not trigger a transition) are handled by the superstate (or its superstate) level or not by any state.

- When an event does trigger a transition, that transition could form part of a transition chain that spans multiple diagrams. To follow the behaviour of one event may require the user to trace a transition to a junction point, find the related junction point in another diagram, then follow the transition connected to that.

- When considering a state machine, superstates and substates are closely related to each other. However only substates within the same diagram are closely related due to the direct transition links between them. Therefore state machines can be thought of has have a tree structure, where branches have no direct relationship to other branches.
Figure 4-3 – RoseRT “Traffic Lights” model. The tree on the left shows the “navigator” hierarchy of states within the state machine. The state machine diagram on the right shows the top level. Both “initializing” and “working” are composite states containing substates diagrams, as can be seen by the icon bottom right and the child states in the “navigator”.

Transformation Chain Example

This example presents typical artefacts produced by a model-driven development process. An initial UML class model is transformed by several model transformations, resulting in a so called transformation chain. The following artefacts are to be visualized:

1) Diagram 1: Initial UML class diagram representing a simple library model (Figure 4-4)
2) Transformation 1: The first transformation adds an interface to the model and an attribute ID to each class. The transformation consists of the following rules:

   a) For each package, a new package with the same name is created and an interface “IUnique” is created.

   b) For each class in the source model, a class in the target model in created with the same name. The newly created class implements the interface “IUnique” and an attribute “Id” is added.

   c) For each attribute/operation in the source class, an attribute/operation in the target class is created.

   d) For each association in the source model, an association in the target model is created.

3) Diagram 2: The second UML class diagram is the result of the first transformation applied on the first model (Figure 4-5).
4) Transformation 2: The second transformation adds getter and setter methods for each attribute and association. The transformation consists of the following rules:

a) For each package, a new package with the same name is created.

b) For each classifier in the source model, an appropriate classifier in the target model in created with the same name.

c) For each attribute in the source classifier, an attribute in the target classifier is created. The visibility of the attribute is changed to “private” and appropriate getter and setter operations are created.

d) For each operation in the source classifier, an operation in the target classifier is created.

e) For each association in the source model, an association in the target model is created. The properties of the classifier referencing the associations are made “private” and appropriate getter and setter operations are added.

5) Diagram 3: The third UML class diagram is the result of the second transformation applied on the second model (Figure 4-6).
Figure 4-6 – Third diagram, UML class model with getters and setters

6) Transformation 3: The third transformation creates an Entity-Relationships (ER) model. For each class, an entity is created. Associations are mapped to relationships, attributes are omitted here for simplicity. That is:

a) For each package, a new ER model.

b) For each class in the source model, an entity in the target model in created with the same name.

c) For each association in the source model, a relationship in the target model is created.

7) Diagram 4: The last diagram is an ER diagram. It is the result of the third transformation applied on the third model. Note that it might have been possible to apply this transformation to the preceding models as well, but it was applied to the
third model here. In a more complex situation, the preceding transformation may have added other attributes or classes which have to be mapped, too (Figure 4-7).

![Diagram](image)

**Figure 4-7 – Last diagram, ER model**

Visualization hints:

- Transformations are to be visualized. They may be visualized by drawing traces for each applied rule. A trace connects all source elements with their target elements. Optionally a label is added indicating the name of the rule or other comments. Note that traces may connect m source elements with n target elements.

- Note that traces may be ordered hierarchically: A top level rule transforms the package, on the second level, classifiers are transformed, on the next level attributes and operations, and so forth.

- The layout of the diagrams was created automatically here by some tool without taking preceding diagrams into account. This may be optimized, that is the layout of the 2D diagrams may be changed (if 2D diagrams are used).
The transformations are rather simple here. This is usually not the case. That is: Do not rely on special properties of the transformations here. In MDD, a transformation may transform $m$ source models into $n$ target models, but usually, a single source model is transformed into a single target model (maybe using some additional information which are not visualized). For the visualizing, you can assume simple 1:1 transformations.

**Motivation and User Tasks**

To be able to apply heuristics effectively we need to consider the user and what they are trying to achieve. For example, to evaluate compliance to the heuristic “Speak the User’s Language”, we need to know who the "User" is to determine what their language might be. We also need to consider what information needs to be communicated to this user before we can evaluate if the way it has been communicated is appropriate. The following gives information on the user, their motivation and the task for each diagram type:

**State Machine Diagrams**

*User (of the diagram):* Embedded Software Engineer familiar with UML and RoseRT

*Motivation:* Employed to develop and maintain a variety of controllers for different products.

*User Task:* Develop and maintain controller firmware for traffic light behaviour.

**Transformation Chain**

*User (of the diagram):* UML user who created the very first “domain model”

*Motivation:* Understand the transformation chain, check for errors or bugs, and understand the derived models.

*User Task:* Use MDD techniques to speed up development, use prepared transformations (maybe third party transformations)
4.6.4 3D UML Heuristic Challenge

The Layout Teams were asked to select 2-3 heuristics from the 10 defined; they then completed the question cards about the heuristics (as per the example below). With the heuristics defined, the team then demonstrated diagram layouts which represent the heuristics.

The Heuristics Team were asked to answer all question cards about the heuristics in Section 4.6.1. Once complete they use the full evaluation sheet to test their heuristic definitions against 3D UML diagram layouts.

Question cards about the Heuristic

Before the diagrams are evaluated, the participants are asked about the heuristics first. As the heuristics are generic they can be interpreted in many ways, we need to determine how the groups interpret them for 3D UML. For example, what does "language" actually mean, in the heuristic below and having determined this, how do we quantify compliance?

The questions are presented on individual cards so that participants can easily treat heuristics one by one. A sample question looks like the following (ratings are from 1 – 5, where 1 = "Not applicable" and 5 = “Essential”):

1. Speak the User’s Language [MN90]
   “The dialogue should be expressed clearly in words, phrases, and concepts familiar to the user rather than in system-oriented terms.” [MN90]
   How do you think this heuristic description can be better stated for UML Diagrams?
   What criteria would you expect to use in rating a diagram against this heuristic?
   How important do you think this guideline is? ① ② ③ ④ ⑤

*Figure 4-8 – Question card for heuristic evaluation. Workshop participants were asked to answer questions like this for each guideline.*
Full Evaluation

For the Heuristics group, full evaluation of existing 3D UML diagram task, the following questionnaire is used with the heuristics listed above (ratings are from 1 – 5, where 1 = “fails to comply” and 5 = “fully complies”):

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Rating</th>
<th>Comments/Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speak the User’s Language</td>
<td>1 2 3 4 5</td>
<td>___________________</td>
</tr>
<tr>
<td></td>
<td></td>
<td>___________________</td>
</tr>
<tr>
<td>Overview</td>
<td>1 2 3 4 5</td>
<td>___________________</td>
</tr>
<tr>
<td></td>
<td></td>
<td>___________________</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4-9 – Heuristic Evaluation sheet. Workshop participants were asked to evaluate 3D UML diagrams using this form.

4.6.5 State Machine Diagram Layout Results

Group 1 had the challenge of selecting 3 important heuristics, refining those heuristics and then creating a 3D UML state machine diagram layout to demonstrate a diagram adhering to their chosen criteria. The group was the smallest with only 3 members and was the least motivated out of the four groups. One team member stated they were sceptical about the use of 3D for diagrams, which may have contributed to the lack of motivation.

The team also had trouble refining the heuristics, this was partly due to not considering the heuristics individually (as happened in the Melbourne test workshop Section 4.6.8). For example they considered two heuristics ‘2. Overview’ and ‘4. Zoom’ [83] to be the same, because ‘you’d naturally want to zoom into something from an overview’. It was asked of them to consider if it was possible to implement a diagram that had an overview but no zoom. They considered this and said ‘yes’ it was possible but stated ‘it wouldn’t
be logically done because it wouldn’t make sense’. They were then asked to consider why it didn’t make sense and suggested that the answer would help define the heuristic for “zoom” as a separate entity.

After 30 minutes they were still discussing the heuristics. The team seemed to get frustrated and towards the end of the session were chatting about other things. There were no heuristics refined on the cards. The heuristics instead were presented verbally as the diagram was presented.

Group 1’s Top 3 Heuristics were:

5. Relate

9. Details on Demand

7. Filter

Group 1’s diagram layout is shown in Figure 4-10 and is very similar to the X3D-UML 3D state machines diagrams, with state machine diagrams layered in space. It is known that at least one of the participants had seen an X3D-UML example prior to the workshop and this may have influence on the result. The features of the diagram, in relation to the heuristics are as follows:
5. **Relate** – Through a user event, the entire diagram rotates to a side view which shows the hierarchal view of the state machine and shows how each diagram relates to the one above and below.

9. **Details on Demand** – Through a user event such as clicking on a junction point, transition pathways through the diagram are highlighted.

7. **Filter** – Through a user action such as clicking on a diagram, top layer diagrams flip away (like the windowing features in Apple OS X and Microsoft Vista)

### 4.6.6 State Machine Diagram Evaluation Results

Group 3 (with 4 members) had the challenge of reviewing 10 heuristics from the perspective of undertaking a 3D UML state machine diagram evaluation. The group initially had trouble understanding the motivation for the task, so it was put to them to imagine that they are a company of heuristic evaluators whose job is to review 3D UML
diagrams. The heuristics will be their list to have ready when customers come to them and say “what are the problems with my diagram?”

Once the group had a common understanding of the goal, they worked well together completing all 10 heuristics. They also added a new heuristic for navigation as they felt that navigation was important and a known problem with 3D.

Group 3’s Heuristics Priority was:

Equal First (Rating 5)

4. Zoom

6. Minimize the User’s Memory Load

9. Details on Demand

7. Filter

2. Overview

11. Navigation

(Rating 4)

5. Relate

8. Data-Ink Maximization, Data Density

(Rating 3)

1. Speak the User’s Language

3. Be Consistent

10. History

Group 3 then reviewed the heuristics in order of the above priority and refined them to suit 3D UML state machine diagrams. The refinement asked for a refined heuristic
description, the criteria to measure the heuristic against and the overall opinion of the importance the heuristic. The refined heuristics are presented in Table 4-1.

Table 4-1 – Refined Heuristics for State Machine Diagrams

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Refined UML Description</th>
<th>Measurement Criteria</th>
<th>Heur. Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Speak the User’s Language</td>
<td>help/tooltip</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>2. Overview</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>3. Be Consistent</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4. Zoom</td>
<td>Restrict movement</td>
<td>Support for graphical zoom + navigation help. Support for semantic zoom.</td>
<td>5</td>
</tr>
<tr>
<td>5. Relate</td>
<td>Keep it June</td>
<td>Navigate along connections. Highlight related objects (deal with obstacles). Visualise perspective (direction of line)</td>
<td>4</td>
</tr>
<tr>
<td>6. Minimize the User’s Memory Load</td>
<td>Way to find items you are looking for. Keep mental map.</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>7. Filter</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>8. Data-Ink Maximization, Data Density</td>
<td>Collapse items</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>9. Details on Demand</td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>10. History</td>
<td>Keep this heuristic</td>
<td>Support for history (undo/redo). Navigate backwards (navigation history)</td>
<td>3</td>
</tr>
<tr>
<td>11. Navigation (added)</td>
<td>Restrict movement</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

With the above refined heuristics Group 3 then evaluated the 3D UML state machine diagram implementation. They rated the diagrams compliance to the heuristic and added comments about the compliance. The results are shown in Table 4-2.

Table 4-2 – Evaluators Assessment of the X3D-UML 3D state machine diagrams

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Eval. Rating</th>
<th>Comments</th>
</tr>
</thead>
</table>
| 1. Speak the User’s Language       | 4            | • Common UML  
• 3D Mental model  
• Missing tooltip |
| 2. Overview                        | 2            | • No overview layout  
• Semantic zoom out for overview  
• Overlapping rectangles in initial view |
| 3. Be Consistent                   | 5            | • Nothing inconsistent |
| 4. Zoom                            | 3            | • No semantic zoom only graphical zoom  
• No navigation help |
| 5. Relate                          | 4            | • Less transparency  
• or interactive highlight. |
| 6. Minimize the User’s Memory Load | 4            | • No search  
• Mental map is kept |
| 7. Filter                          | 1            | • No Filter |
| 8. Data-Ink Maximization, Data Density | 5           | • Collapsing Possible |
| 9. Details on Demand               | 4            | • It’s available  
• No skipping of labels (upside down) |
| 10. History                        | 2            | • No navigation history |
| 11. Navigation (added)             | Not stated   | • Rotation |

101
Group 3 then presented their evaluation results with explanation of some of the main points and extra things to consider: The main things highlighted from the presentation were:

- Navigation is an important consideration for 3D and therefore required a new heuristic.
- The diagram had the problem with no restriction in navigation, causing upside down views.
- They considered the overview too complex and the tool needed to provide automatic layout.
- The diagram should filter less important elements.
- The diagram should provide a search function.

### 4.6.7 GEF3D Translation Chains Results

This section gives the results of GEF3D translation chains focusing on heuristic ratings for general 3D UML heuristic evaluation. GEF3D specific results with regard to evaluation we have reported in von Pilgrim et al. [94].

#### Translation Chains Layout Results

Group 2 had the challenge of selecting 3 important heuristics, refining those heuristics and then creating a translation chain diagram layout to demonstrate a diagram adhering to their chosen criteria. In comparing the results to 3D UML state machines diagrams, it is interesting to note that the group also created a new navigation heuristic as did the 3D UML state machine evaluation group.

Furthermore, in a similar result to the 3D UML state machine layout group, the translation chains groups’ 3D UML diagram solution appeared similar to the GEF3D solution. This may suggest that applying heuristics for the design of 3D UML creates
consistent diagrams or it may suggest that the solutions proposed are the most natural mapping to the participants’ mental model.

Group 2’s Top 3 Heuristics were:

2. Overview [83]
4. Zoom [83]
11. Navigation

Translation Chains Evaluation Results

Group 4 (with 4 members) had the challenge of reviewing 10 heuristics from the perspective of undertaking a 3D UML translation chain evaluation. The refinement asked for a refined heuristic description, the criteria to measure the heuristic against and the overall opinion of the importance the heuristic. The refined heuristics are presented in Table 4-3

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Refined UML Description</th>
<th>Measurement Criteria</th>
<th>Heur. Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Speak the User’s Language</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>2. Overview</td>
<td>High level view of transformation chain</td>
<td>available; easily viewable (one screen); how much interaction required</td>
<td>4</td>
</tr>
<tr>
<td>3. Be Consistent</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>4. Zoom</td>
<td>Physical zooming</td>
<td>How easy to intake; info re: zoom How smooth; how much context provided; limit the level</td>
<td>4</td>
</tr>
<tr>
<td>5. Relate</td>
<td>Within diagram: adhere to UML standard, Transformation: clearly relate occurrences of the same entity across diagram</td>
<td>relation can be followed forward and backward; type of transformation is clear; direction of transformation is clear (input vs. output)</td>
<td>5</td>
</tr>
<tr>
<td>6. Minimize the User’s Memory Load</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>7. Filter</td>
<td>Show the level of detail for the user, task and stage of design.</td>
<td>How easy is it to filter out items</td>
<td>4</td>
</tr>
<tr>
<td>8. Data-Ink Maximization, Data Density</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>9. Details on Demand</td>
<td>Precise information about every change between 2 diagrams</td>
<td>Is details information available for every item of interest</td>
<td>4</td>
</tr>
<tr>
<td>10. History</td>
<td></td>
<td></td>
<td>1 (browsing)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4 (editing)</td>
</tr>
</tbody>
</table>
4.6.8 Test Workshop Results

Prior to the LED’08 workshop the 3D UML Heuristic Challenge was tested to refine the session to make the best use of time. The test runs were undertaken at FernUniversität in Hagen, Germany and RMIT Melbourne, Australia. Although the workshops were intended to be trial runs they produced some interesting results. As the direct aim of these test sessions were to streamline the evaluation process, these results are applicable the methodology refinement and are reported here.

Heuristics Challenge Test Run FernUniversität in Hagen

The first test run undertaken was in Germany, with the FernUniversität in Hagen “Focus Software Engineering” group, consisting of 8 researchers (3 professors, 5 research assistants). Our initial intent was to run the layout and evaluation phases in serial, with each team creating a layout and then reviewing another team’s solution. However the test run trialled only the layout section of the challenge. The results showed that the challenge as a whole was too long and could not be run in serial. The revised challenge was then executed in Melbourne.

Of note, for X3D-UML state machine diagrams the selected guidelines were: Overview, Zoom and Relate. Surprisingly the resulting diagram for state machines (Figure 4-11) closely resembled the X3D-UML solution (Figure 5-9).
Heuristics Challenge Test Run RMIT Melbourne

The second test run was undertaken at the fortnightly “Distributed Systems Engineering and Architecture” group, consisting of 8 researchers and lecturers. Due to time limitations the test run trialled only state machine diagrams, but tested both the layout and heuristics parts of the challenge. Furthermore the layout teams only picked 2 heuristics (instead of 3) and the heuristics team to only pick 5 (instead of 10).

Heuristic Group

The team were give the exact same 3D UML state machine diagram as provided to the real evaluators in main 3D UML state machine diagram study (i.e. it was not a refined state machine).

This test session highlighted that information about the user, their motivation and user task were required to focus the evaluators attention. For example one evaluator started considering the use of state machines by business people and what their language
might be. As the state machine tool is aimed at embedded engineers, this focus would have produced undesired results.

The session also highlighted that care needed to be taken to get evaluators to consider the heuristics as individual guidelines. Without doing this the intended purpose of the guidelines was altered. For example, one evaluator was trying to consider both Overview and Zoom together, which both caused confusion and failed to isolate problems to each specific area.

**Layout Group**

The layout group chose the heuristics Overview and Zoom. In a similar result to the German test run they produced a 3D state machine diagram similar to the X3D-UML solution. However the group went further to also suggest a 2D solution which involved “flattening” the overview through a “centre out” approach (Figure 4-12).

*Figure 4-12 – Layout Group “2D” solution to meeting the overview heuristic*
4.6.9 Conclusions

The LED’08 workshop and test runs provided a unique opportunity to observe experts from different areas in software engineering undertaking heuristic evaluation. The motivation for this study was that initial experiences with using heuristic evaluation found that UML experts, that were not necessarily usability experts, were time poor and not undertaking the evaluation in a structured way. The result of the workshop has therefore streamlined and structured the evaluation process to avoid common pitfalls and encourage more participation. The outcomes of this have lead to a refined process specifically for 3D UML. This section summarises the outcomes and the refined process is documented in the next Section 4.6.10.

A summary of the heuristics ratings in Table 4-4 shows that most heuristics reviewed for 3D UML where considered between average importance (3) and essential (5). One notable exception was the concept of “Speak the User’s Language” with an average rating of 2.5. We chose this guideline as a heuristic due to our belief that if the visualisation did not look like UML, it would not be familiar to engineers and fail this basic guideline. The low ranking however is due to evaluators not considering that the area of 3D software visualisation has many instances of non-UML like visualisations (this was directly observed in the German test where participants stated they did not consider that a visualisation might change the way UML looked).

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Eval. SM</th>
<th>Eval. TC</th>
<th>Layout SM</th>
<th>Layout TC</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Speak the User’s Language</td>
<td>3</td>
<td>2</td>
<td></td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>2. Overview</td>
<td>5</td>
<td>4</td>
<td>X</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>3. Be Consistent</td>
<td>3</td>
<td>4</td>
<td></td>
<td>3.5</td>
<td></td>
</tr>
<tr>
<td>4. Zoom</td>
<td>5</td>
<td>4</td>
<td>X</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>5. Relate</td>
<td>4</td>
<td>5</td>
<td>X</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>6. Minimize the User’s Memory Load</td>
<td>5</td>
<td>3</td>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7. Filter</td>
<td>5</td>
<td>4</td>
<td>X</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>8. Data-Ink Maximization, Data Density</td>
<td>4</td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>9. Details on Demand</td>
<td>5</td>
<td>4</td>
<td>X</td>
<td>4.5</td>
<td></td>
</tr>
<tr>
<td>10. History</td>
<td>3</td>
<td>1 (browsing)</td>
<td>4 (editing)</td>
<td>2</td>
<td>3.5</td>
</tr>
<tr>
<td>11. Navigation</td>
<td>5</td>
<td></td>
<td></td>
<td>X</td>
<td>5</td>
</tr>
</tbody>
</table>
The true test of the validity of a heuristic evaluation is if it indeed captures problems that will become apparent in usage. The main purpose is a low cost, first round, evaluation technique to flush out obvious problems. Table 4-5 compares the heuristic evaluation results for state machines with the response from actual users through a formative evaluation session. The results show that most issues raised in heuristic evaluation had an equivalent issue raised by users. There were very few areas of conflict and the conflict was minor. For example the issue of filter, one user did not consider it important if there was a way to navigate the view in a way that bought the diagram of interest into view quickly, in effect the navigation was a form a filter. Other issues were influenced by the presence of the researcher in the formative evaluation, for example, negating the need for a “tool tip” as the researcher could be asked a question about using the tool. There were more user issues that were not highlighted by heuristic evaluation, however these were highly domain and tool specific and beyond the expected knowledge of evaluators.
Table 4-5 – Comparison between Evaluators’ and Users’ Responses (detailed in Section 5.5.3 On-Site Formative Session with references to user feedback items)

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Evaluator Comments</th>
<th>User Formative Session</th>
</tr>
</thead>
</table>
| 1. Speak the User’s Language | • Common UML  
• 3D Mental model  
• Missing tooltip | • No Issues were raised with notation  
• No issues were raised with mental model  
• No issues were raised with “tooltip”, however formative evaluation was undertaken with the researcher present and questions were asked of the researcher |
| 2. Overview | • No overview layout  
• Semantic zoom out for overview  
• Overlapping rectangles in initial view | • User stated they needed a side tree view (ref 5.2.1)  
• User stated they wanted to “hot-key” between different view aspects (ref 3.3) |
| 3. Be Consistent | • Nothing inconsistent | • No issues were raised with inconsistency |
| 4. Zoom | • No semantic zoom only graphical zoom  
• No navigation help | • User requested to allow quick ‘snap to’ navigation to individual state machine diagrams (ref 3.2.1)  
• No issues were raised with “tooltip”, however formative evaluation was undertaken with the researcher present and questions were asked of the researcher |
| 5. Relate | • Less transparency  
• or interactive highlight. | • Transparency was found to be a user preference and the tool required Opacity configuration to suit user preferences and different graphics capabilities of machines (ref 5.1)  
• Dynamic highlighting requested by users (ref 1.1) |
| 6. Minimize the User’s Memory Load | • No search  
• Mental map is kept | • No issues were raised with searching, however the user that used the existing model search function to locate model elements was not part of the formative session  
• No issues were raised with mental model |
| 7. Filter | • No Filter | • In the formative session “collapsing branches” was presented as a filter mechanism, however one preferred better navigation than such filtering (ref 3.2.2) another suggested alternative filtering methods (ref 1.2.1 and 1.2.2) |
| 8. Data-Ink Maximization, Data Density | • Collapsing Possible | • See “filter” above. |
| 9. Details on Demand | • It’s available  
• No skipping of labels (upside down) | • Users requested more details for job specific tasks (ref 1.2 and 6)  
• Users requested upside down views be prevented (ref 3.1) |
| 10. History | • No navigation history | • No issues were raised with navigation history but users requested navigation improvements (ref 3.2) |
| 11. Navigation | • Rotation | • User requested Hot-keys to navigate to different view aspects (ref 3.3) |
| Presentation | • Problem with no restriction in navigation, causing upside down views.  
• Overview too complex, provide automatic layout  
• Filter less important elements  
• Provide a Search Function | • User requested restricted navigation options to prevent awkward views (ref 3.1)  
• User requested ways of improved hierarchal view layout (ref 5.2.1)  
• In the formative session “collapsing branches” was presented as a filter mechanism, however one preferred better navigation than such filtering (ref 3.2.2) another suggested alternative filtering methods (ref 1.2.1 and 1.2.2)  
• No issues were raised with searching, however the user that used the existing model search function to locate model elements was not part of the formative session |
The final point to make from the workshop was that the above information was gained from a 20 minute evaluation of the 3D UML tool. This demonstrates that the approach did streamline the evaluation process, although some speed improvements may be attributed to the heuristics being “fresh” in the minds of the evaluators from working on refinements. In practice the overall final evaluation would not be undertaken by a team of 4, but by 4 experts reviewing independently. The ratings and comments would therefore have more meaning and significance.

4.6.10 Refined Heuristic Evaluation Procedure

The refined heuristics evaluation procedure, involves providing a structured approach so that evaluators can quickly undertake an evaluation and provide a structured feedback. Ten heuristics that have been demonstrated to be effective for 3D UML are listed in Section 4.6.1. A refinement to this list is to replace the heuristic “1. Speak the User’s Language” with “Navigation” unless a new style of UML notation is being evaluated.

In this section we outline some important information for evaluators to consider when undertaking an evaluation:

**User, Motivation and Task**

To concentrate on usability issues specific to the user tasks that are being studied, the evaluator needs to consider the heuristics from a specific point of view, considering the user, the user’s motivation and user’s specific task. For example, the evaluator may consider state machine diagram evaluation from the point of a “software engineer”, being “employed to develop and maintain a particular class of software system” with the particular task of “developing new control logic”.

**Validating User Task Problems**

If the expertise of the expert evaluator is with people that use the diagram, then this can be used as an opportunity to gather other valuable data from their experiences. For example data such has the evaluator seen the problem manifest and with how many users.
**Considering Heuristics Individually**

The evaluator should be reminded to consider each heuristic individually. If the evaluator believes that two or more heuristics are the same, then ask the question is ‘*one possible without the other?*’ If the answer is “yes”, then the heuristic should be addressed individually.

**Provide the Evaluator with a Heuristic List (and blank spaces)**

Provide the evaluator with the list of heuristics to evaluate the 3D UML solution against. Provide additional blank spaces for the evaluator to add additional heuristics that they consider necessary.

**4.7 Formative Evaluation**

We have applied “Formative Evaluation” for 3D UML as described in Section 4.2.4 with feedback generated through user walk throughs of the visualisation. This stage also provided valuable information as a means of further defining the user requirements for a diagram. The approach is simple to apply, however the visualisation itself proved to be the biggest cause of issues as outlined below:

**4.7.1 Issues**

**Features to Include**

The main issue was the refinements to the visualisation based on the Heuristic Evaluation stage. Heuristic evaluation uncovered many issues with the visualisation that needed to be addressed; however the complexity in addressing the issues meant that implementing all features/fixes was not practical. For example one feature request was automated layout; layout algorithms are still a problem not yet completely solved for 2D UML, implementing a 3D UML automated layout feature would take significant time (and be a complete research topic in itself). Also due to the novelty of the area the benefit in the feature/fix was unknown, which made prioritising the features/fixes difficult. Due to these factors, there is a risk that features of low importance would be chosen and poorly implemented, with the result causing possibly more usability problems rather addressing existing ones.
**Variations in User Data and Environment**

As our research is constrained to actual user data and environments, this naturally causes issues with unpredictable data and environments. Although the intent in using actual user data and environments is to capture issues not possible with toy scenarios, the research issue is with quickly addressing the problems so that a complete formative session can be executed. In the previous stages the visualisation is derived from some example source, such as an example UML model and this visualisation generated on the researcher’s notebook computer. To work with the user’s data, the visualisation has to be implemented in a way that it can be generated from an unknown data source and unpredictable environment and therefore bugs occur as many variables are not tested. The following illustrates two such examples and provides an indication of the difficulty of researching a visualisation based on actual data, in actual user environments.

As one example of data problems, a classifier name within a UML model was used by X3D-UML to gather information about that classifier and then visualise it. In one participant’s UML model they had implementation and interfaces with the same classifier name, resulting in an empty visualisation due to the X3D-UML implementation only finding the interfaces classifier.

As one example of environmental problems, X3D-UML was used to gather visual information from a user’s model to generate an X3D view and had worked on many computers without problem. The view required floating point calculation of curves, to visualise “roundtangles” and curves were calculated from the height and width of the UML object. Issues were caused by one operating system’s regional preference for using a comma ‘,’ instead of the decimal point ‘.’ for floating point numbers. This caused very unusual behaviour (Figure 4-13) due to X3D’s use of the comma to separate vectors in line definitions.
4.7.2 Refinement

This section lists the refinements that should be applied to “Formative Evaluation” (4.2.4).

The Formative Evaluation stage can be extended to review feature/fixes suggested in Heuristic Evaluation without necessarily implementing those features. Rather than try an implement all feature/fixes or prioritise some based on the intuition of the researcher, feature/fixes can be prioritised based on difficulty in implementation. If a feature is too difficult to implement, Formative Evaluation can be used to determine the impact of the absence of the feature. As an example, an automated layout feature would take significant effort to implement and even what a “good layout” to aim for is not known. Although this was a feature highlighted in Heuristic Evaluation, only manual layout was implemented for the Formative Evaluation sessions. The absence of the feature lead users to discuss what they would like in such a feature should it exist, and this provided research results to what “a good layout” might be. It also revealed that the feature was not expected or of high priority to the user, a much higher priority was the ability to
define a layout themselves and save that layout. In reality the manual layout of a diagram could be done in less than one minute and was only required once per diagram. By not implementing an automated layout approach we were still able to gain equally valuable formative information from users without monopolising the researcher’s time implementing just one feature.

To address the issue of variations in user data and environments, the researcher needs to test the refined visualisation prior to going on site. Where possible the researcher should obtain representative user data to develop against, so problems related to this can be reduced, and test that in an environment similar to the user’s. However due to the sensitivity of user’s data, the above may not be possible, so the researcher should provide a test case that can easily be run by an on-site participant to enable prior communication of problems. The researcher must also be prepared to fix problems on-site quickly to avoid misuse of valuable user time and the subsequent missed opportunities for research results.

4.8 Summative Evaluation

In “Summative Evaluation” we intend to test the implementation against the existing process. Performance measurements are captured while users complete a set of tasks associated with a goal defined from User Task Analysis. The “million dollar question” for 3D User Interfaces is there benefit, and for science, disproving benefit is also of equal value. For this research the ideal outcome would be a performance graph, like the one in Figure 4-14, comparing 2D performance against 3D performance, with state machine diagrams being the most obvious candidate. The research to this point suggests the trends shown in the graph may be possible however it is only speculation until a suitable performance test has been undertaken. Our research has also highlighted issues that would impact the performance tests if they are not addressed prior to undertaking experiments. This section explains the issues and refinements required to achieve a result like in Figure 4-14 (regardless of the line trends).
4.8.1 Issues

In practice the research outcomes from previous phases show that a Summative Evaluation test is a non-trivial exercise. This has meant that the results for this test are ongoing with a time frame greater than what this PhD allows. Below we capture the areas of difficulty and the refinement section captures the structure for ongoing research.

Small Time-Constrained User Base

Finding willing participants in a user study of professional software engineers is difficult. Of the areas studied, 3D UML state machine diagrams contained the most participants with 4 volunteers from Australia. Even if it was possible to arrange for all the participants to undertake performance tests, this number is not sufficient to provide meaningful data (outside those 4 participants). Users from outside Australia have been located; however it is not possible to conduct controlled tests remotely. It is already known that the user environments vary greatly (having already started the process of adapting the X3D-UML tool).

Additionally if the performance test outcome is indeed as suggested in Figure 4-14, we would require a large amount of participants time to get enough data points and this time would not be pleasant. For example, it is expected that there will little benefit in 3D for trivial state machines, the important tests therefore will be on complex state machines.

![Figure 4-14 – Summative Results 2D Vs 3D](image)
The tests would therefore involve long and thought intensive repetitive tasks to get the 2D result.

Large Number of Variables

The research outcomes have shown that the visualisation needs complex interactivity. This interactivity is not yet implemented and the quality of implementation will have an effect on test outcomes. The tests are therefore not as simple as adding a 3D extension, they require specialise interaction techniques that are not yet proven to work.

Also training is an issue to be considered. A direct comparison is difficult when a participant has years of experience working with 2D diagrams and only days working with 3D.

Time to Complete

To enable an effective experiment to test for benefit to be undertaken requires many issues to be addressed which would require a large amount of time. Firstly we would be required to implement the interaction techniques highlighted by our research. This would not only take time in implementation but, because it would also require interaction with users to work through bugs, would require a careful managed of interaction over a period of time to reduce the researchers influence on results. For example, we must avoid participants being influenced by positive and responsive interactions with the researcher and the tool, as it may lead to the participant being influenced more by the researcher than the usefulness of the tool.

4.8.2 Refinement

To measure for benefit of 3D UML requires ongoing research. For our research the 3D UML State Machine Diagram is the most complete diagram studied and basing Summative Evaluation on that diagram we suggest that ongoing research is structured as follows.

To enable testing with a large diverse user base requires the release of an X3D-UML plug-in to augment the existing UML tool with 3D UML diagram capability. This would require a refinement period to fix bugs, followed by a period of no interaction to allow users to settle into usage of the tool without the researchers influence.
It is envisaged that the tool gathers the follow statistics about itself:

a) time the 3D UML diagram is used compared to the current tool usage
b) the types of interaction techniques used in a 3D UML session
c) the types of 3D UML visualisation features used
d) the complexity profile for state machines visualised through 3D UML, compared to those in the entire model

After a period of time (6 months) users can be contacted to gather the statistics. The statistics collected above will provide evidence to the following:

a) The extent of user perceived value in the 3D UML diagram. If the user is using the 3D UML diagram within the tool consistently for a period of time, shows the 3D UML diagram is valued beyond novelty or influence of the researcher.

b) The types of diagram interaction features the user finds of value.

c) The types of visualisation features the user finds of value.

d) The information required to analysis the profile of state machines to create test cases based on actual data.

If the above indeed provides evidence that 3D UML state machine diagrams are perceived as useful, we can proceed to test if they are useful. Because it is possible that the users’ perception of benefits do not match the reality, we must test actual performance through timed tasks. It is envisaged that a game be created that recreates typical industry situations, based on the complexity profiles, that can provide a repeatable experiment across diverse users.

### 4.9 Summary

In this chapter we have explained how the research methodology utilises the *Sequential Evaluation* [32] approach to test for benefit in 3D UML diagrams. This methodology is borrowed from pure 3D User Interface research and refined as *3D UML Sequential Evaluation* and documented in both theory and practice.
The methodology is significant in that it imposes rigor to establishing research results. Prior research described in Section 2.3.3 has evaluated created diagrams against assumed tasks at best. Our methodology first seeks to understand the actual tasks, through User Task Analysis, and if indeed there is an issue that needs resolving. It then seeks quantitative evidence through model metrics to quantify the extent of such issues in industry. These results are of benefit to all researchers interested in the usability of UML, regardless of whether the use of 3D is involved.

Our Heuristic Evaluation refinements provide an effective evaluation technique to allow researchers to gain qualitative evaluation results for 3D UML diagrams. As the evaluation techniques involving users may not be possible or limited, these refinements provide 3D UML guidelines that can be used to improve user evaluation quality or provide at least some external qualitative critique in the absence of user evaluation.

Our Formative Evaluation refinements take into account the complexity of creating 3D UML visualisations and tailor sessions to probe further for user need of labour intensive features.

The Summative Evaluation phase is in ongoing research and may take many years to collect enough data to draw meaningful conclusions; however this is only the case because the previous phases uncovered many issues that impact on the usability of the 3D UML diagrams tested. As the area of 3D UML is novel, issues are to be expected and the methodology has successfully identified and captured the issues and risks that prevent an effective quantitative user test being undertaken.

Due to the volatile nature of the software development industry, it is important not to rely on completion of all phases of an evaluation methodology. At any point in time a development project could be halted, contracts not renewed, companies sold or a global financial crisis takes hold. Despite not completing all phases, our methodology has provided qualitative and quantitative evidence with regard to 3D UML.

The next chapter details our research into 3D UML State Machine Diagram and covers the area that has most influenced our refinements to the Sequential Evaluation methodology. This also gives a concrete real-life example of our methodology in practice.
Chapter 5 - 3D UML State Machine Diagrams

5.1 Overview

As our first investigation into 3D extensions to UML diagrams we decided to extend the UML state machine diagram also known as “statecharts”. As discussed in Chapter 2 and presented in Table 2-1, our research is constrained to actual data and user tasks. Through industry contacts it was possible to undertake a study meeting these constraints through a UML tool called IBM Rational Rose RealTime (abbreviated to RoseRT).

In this chapter we first describe in Section 5.2 the background to the existing 2D state machine diagrams and the proposed 3D extension. We then present the evaluation of the 3D extension using 3D UML Sequential Evaluation, with sections 5.3 to 5.6 describing the results of each stage of this approach. Finally we summarise the results in Section 5.7.

---

Permission to use image for this thesis was kindly granted by Jacqueline Hansson, IEEE Intellectual Property Rights Office.
5.2 2D State Machine Diagrams

Our research into 3D UML State Machine Diagrams is based on RoseRT. RoseRT as a tool in itself is an excellent candidate for aiding the investigation into extending state machine diagrams. RoseRT makes extensive use of state machine diagrams for describing system behaviour and these descriptions generate code (i.e. “the model is the code”). The benefit is that the diagram is not simply a documentation object that can be read once, ignored or misinterpreted but it is instead the primary means of interacting with the code.

Working with RoseRT, the software engineer must fully understand the state machine diagram and must interact with it to do their day-to-day tasks. More details on RoseRT can be found, under its new name “Rational Rose Technical Developer” [43]. RoseRT is a mature\(^\text{18}\) and successful product. The fact that the UML is at version 1.4 and the tool has not been noticeably updated since the acquisition by IBM in 2003 provides a unique research opportunity. Due to the longevity and stability of the product, there is potential to research large projects with experienced users. This has enabled our research to focus on looking at usability issues directly related to UML diagrams themselves, rather than usability issues caused by immature tool implementations or immature use of UML.

RoseRT implements a hierarchal state machine diagram using a layered approach. The top-level state machine diagram is presented first as the entry point to the complete state machine diagram. Any state on the top level, that has substates, has an icon in the lower right hand corner (which can be seen on the state labelled “green” in Figure 5-1) to indicate this and that there is more information available. By clicking on the state, a new window tab is opened which contains the substate diagram, which itself may also have substates. This layered approach is good at abstracting away detail but may cause issues when trying to understand the full state diagram, due to detail being hidden.

An alternative approach to presenting state diagrams, used by some other UML tools, such as IBM Rational Software Modeller [44], is to present all states in the same diagram, where substates are drawn inside the superstates. This approach has the advantage of allowing the software engineer to view the full state machine diagram in a single view; however, the complexity of the state machine diagram is limited to the amount of information that can be drawn in a single viewable diagram. Also diagrams become difficult to manage as, for example, adding a new state lower down in the hierarchy, would require the whole state machine diagram layout to be changed to accommodate the new object. In Figure 5-1 above we see an example of a training UML model provided with RoseRT called the “TrafficLights” model, in Figure 5-2 we see the same state machine created in IBM Rational Software Modeller.
Figure 5-2 – An example of a non-layered approach to describing a state machine. This diagram represents the RoseRT “TrafficLights” state machine presented Figure 5-1 recreated with IBM Rational Software Modeller. Instead of each state machine layer being displayed in a separate tab, the entire state machine is presented in a single diagram.

Trying to include all state machine entities into a single diagram is problematic. The state machine in Figure 5-2 is fairly trivial, yet the researcher’s experience in creating the diagram was that the layout was driven mostly by “how to make things fit”, rather than “how to make it easily readable”. This problem is also highlighted by Ambler [3], who provides is a set of guidelines for creating UML diagrams. Guideline 206 states the following regarding the style of diagram depicted in Figure 5-2:

Although showing substates in this manner works well, the resulting diagrams can become quite complex – just imagine what would happen to Figure [of seminar state diagram] if the Being Taught state also had substates. An alternative approach would be to create a hierarchy of UML state machine diagrams… The advantage of this approach is that another detailed diagram could be developed to explore the Being Taught state as required.’

Our hypothesis is that engineers would benefit from a 3D state machine diagram giving them “the best of both worlds”, having advantages of separate substate diagrams but also the ability to view the state machine diagram as a whole. RoseRT implements separate substate diagrams by default. Figure 5-3 shows a prototype 3D UML state machine diagram we created from the RoseRT “TrafficLights” state machine Figure 5-1.

Figure 5-3 – Our prototype 3D State Machine Diagram displayed in X3D. With this diagram it is possible to rotate the entire diagram to different views and zoom in on any part of the diagram.¹⁹

The three diagrams above demonstrate the concepts on non-complex “toy” diagrams. The next sections give the results of testing our hypothesis on actual diagrams using our 3D UML Sequential Evaluation approach.

¹⁹ A video example is available at www.x3d-uml.org/YouTube_Examples
5.3 User Task Analysis

In this section we present our findings from User Task Analysis. User Task Analysis captures representative user task scenarios through one-to-one sessions with experienced users of RoseRT. The results are based on user sessions with 3 experienced RoseRT users from a team of 5. Additional results are gathered through models statistics from broader industry which analyse the usage of state machine diagrams in existing models.

The sections are presented in order of execution of our study. The first step, Section 5.3.1, was to determine if hierarchal state machines were commonly used in industry. The next step, Section 5.3.2, was to understand and document how users actually used state machine diagrams. With this knowledge the main issues related to this usage are the documented in Section 5.3.3 with the proposed 3D solution presented in Section 5.3.4. Further quantitative data is gathered in Section 5.3.5 to quantify the extent of issues in industry.

5.3.1 Online State Machine Diagram Survey

To first justify research into using 3D for hierarchal state machine diagrams, we must determine if such diagrams are common in industry. To answer this, a survey of 1004 state machines, from four independent companies, was undertaken using a RoseRT model metrics script we developed. The script counted the number of states at each level with the results in Table 5-1 showing all models making extensive use of hierarchal state machines, with between 33.58% and 64.66% of all states existing at substate levels. The significance of this result is that it provides quantitative evidence that any solution that improved the use of hierarchal state machines would be beneficial to industry and therefore worth pursuing. If the survey had returned evidence to the contrary i.e. very few hierarchal state machines, this evidence would be used to refute the use of 3D in this area based on the lack of need.
Table 5-1 – Substate survey of commercial RoseRT models, showing extensive substate usage. The results are from four industries and have been generalised to areas of “office equipment”, “robotics control”, “systems control” and “networking equipment”.

<table>
<thead>
<tr>
<th>Totals</th>
<th>Office</th>
<th>Robotics</th>
<th>Control</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Machines</td>
<td>216</td>
<td>172</td>
<td>583</td>
<td>33</td>
</tr>
<tr>
<td>States</td>
<td>1576</td>
<td>411</td>
<td>3727</td>
<td>116</td>
</tr>
<tr>
<td>Level 1 States</td>
<td>780</td>
<td>273</td>
<td>1501</td>
<td>41</td>
</tr>
<tr>
<td>Level 2 States</td>
<td>567</td>
<td>90</td>
<td>1295</td>
<td>22</td>
</tr>
<tr>
<td>Level 3 States</td>
<td>208</td>
<td>48</td>
<td>617</td>
<td>22</td>
</tr>
<tr>
<td>Level 4 States</td>
<td>21</td>
<td>0</td>
<td>274</td>
<td>31</td>
</tr>
<tr>
<td>Level 5 States</td>
<td>0</td>
<td>0</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>Percentage of Substates</td>
<td>50.51%</td>
<td>33.58%</td>
<td>59.73%</td>
<td>64.66%</td>
</tr>
</tbody>
</table>

These results also enabled us to compare the RoseRT example models, provided with the toolset for training purposes, against industry models and therefore determine if they may be suitable candidates as test data. The results of analysing 30 example models showed only 12 models used substates and only 1 (the TrafficLights example model) had substates at levels 3 and 4. This indicates that the example models did not suitably represent actual industry models and can be considered “toy data” which can not be reliably used for analysing the effectiveness of new visualisations for actual industry tasks.

5.3.2 User Sessions

In this section we present the results of user sessions. These involved a one-to-one questionnaire (Appendix A) and an open discussion related to working with RoseRT, followed by observation of the user working. The sessions comprised of 3 users from a team of 5 at a single company. A later session was also undertaken with an ex-team member and those results are presented separately.

“Evaluation Sessions” were initially thought to be the most appropriate means of generating information regarding user tasks. The sessions were to involve users from a single company talking about and walking through their every day tasks using RoseRT on example models (despite their known limitations), in a location away from their work place. This type of session was thought to be appropriate because we were trying to glean generic task information that could be applied to any UML model. More
importantly, in theory, it increased the possibility of user involvement because it was decoupled from any company confidential information.

In reality participants were reluctant to give up their spare time to participate in the research. One expressed they were happy to help but would rather do it at their desk. Based on this, the company was approached and it was possible to arrange on-site sessions. A separate arrangement was made where the researcher allocated some time to non-related consulting work to compensate for the engineering time taken away from development projects. The consulting was arranged so as to not influence participation in the research and of the 5 team members only 3 volunteered to participate.

The first session immediately showed that undertaking the sessions in the work environment was far superior and that off-site “Evaluation Sessions” were not appropriate. Although the sessions were not aimed at any particular model, or in some cases even with RoseRT running, completely surprising and valuable information arose out of talking to users at their desk. For example, when trying to determine the mental models that people might have with regard to state machines, one user thought for a while, discussed a few things and then as an afterthought turned to show me how he drew state machines in a notebook first before creating them in RoseRT. This piece of evidence would not have arisen outside the work environment.

The user sessions were intense, generating a lot of information very quickly and not all of it could be written down. It was useful to break up the sessions, reflect and review the results while they were still fresh in our minds. As an example, the first one hour session took over 4 hours to document and review the results. This time was very important because the users “don’t know what they don’t know” and it was the areas where users had trouble stating the problem which were the most interesting to our research because they are the areas that may be improved with 3D. For example one user stated that ‘Sometimes I wonder if diagrams are a distraction’ but could not concisely explain why. After we probed this statement further, we found that the diagram was not showing all that was required for the task, and the user was attempting to use another means to solve the problem. The user could not concisely state what was missing because the alternative was incomplete also. Breaking up the sessions had the added advantage of allowing us to return to things of interest to seek clarification.
User Questionnaire Summary

The process is to walk through the set questionnaire in Appendix A with each user at their desk, then leading those closed questions to an open discussion. In practice only particular questions were suited to closed questions, areas related to UML and 3D were not clearly defined and instead of enforcing the closed structure, these were used as discussion points. Table 5-2 lists the results of the closed questions that yielded results.

Closed questions related to 3D use were too restrictive and instead the questions were used as discussion points. The results of this showed that the users had very mixed prior experience with 3D.

User 1: ‘Played first person shooter games’ and had worked on a ‘visualisation tool that produced 3D visualisations from digitised (3D scanned) data’ and stated that they had a positive attitude towards 3D, however later in the study they revealed they had trouble with stereo perception.

User 2: stated they had no 3D experience and were not of any opinion about the use of 3D.

User 3: had commercial CAD (Computer Aid Design) experience and played first person shooter games ‘10 hours a week.’ User 3 was positive towards 3D however it depended on the application and based their opinion on the ‘single factor of how easy 3D navigation is to use; how easy is it to get to a view of what I want.’

In a similar way, some closed questions related to UML were too restrictive. For example UML was too tightly coupled to the tool to have a clear 1-5 rating. The discussion related to ‘how you feel UML helps you in your role’, lead to discovering that User 1 and 2 felt that UML was helpful, regardless of RoseRT and User 3 felt that ‘Diagrams have always been used and can be considered essential but UML per se isn’t.’
<table>
<thead>
<tr>
<th>Attribute</th>
<th>User 1</th>
<th>User 2</th>
<th>User 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currently Using RoseRT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Last Used RoseRT</td>
<td>Today</td>
<td>Today</td>
<td>Today</td>
</tr>
<tr>
<td>Approx. hours RoseRT usage last 6 months</td>
<td>480</td>
<td>840 (35 hours per week)</td>
<td>840 (35 hours per week)</td>
</tr>
<tr>
<td>Approx. hours total RoseRT usage</td>
<td>960 (last 12 months)</td>
<td>1820 per year x several years</td>
<td>1820 per year x 6.5 years</td>
</tr>
<tr>
<td>RoseRT Training</td>
<td>In-house</td>
<td>IBM Rational Training Course</td>
<td>IBM Rational Training Course</td>
</tr>
<tr>
<td>Primary Goal when using RoseRT</td>
<td>Creating the part of new firmware for a new product. Specifically creating the product configuration management part which allows external PC applications to control the product configuration.</td>
<td>Creating part of new firmware for a new product.</td>
<td>Creating part of new firmware for a new product.</td>
</tr>
<tr>
<td>High level tasks when using RoseRT</td>
<td>Class structure is generated through an XML document. Behaviour is then created manually.</td>
<td>Alarm processing and reporting logic</td>
<td>Develop the part that talks to the hardware</td>
</tr>
<tr>
<td>High level tasks from above that use state machine diagram</td>
<td>All behaviour is created manually through state machine diagrams.</td>
<td>All tasks, as alarm processing is state based</td>
<td>Most of what is done is done with state machine diagrams except for interrupt stuff</td>
</tr>
<tr>
<td>Tasks that use state machine diagram</td>
<td>Creating new state machine diagrams. (20% of time) Refactoring existing state machine diagrams. (80% of time) Reviewing own or someone else’s state machine diagram (rarely, maybe once a week)</td>
<td>Creating new state machine diagrams. (30% of time) Refactoring existing state machine diagrams. (40% of time) Testing and watching execution (30% of the time and feeds into Refactoring above)</td>
<td>Get a definition spec (basically datasheet) of hardware and use state machine diagrams to toggle lines</td>
</tr>
<tr>
<td>Job Title</td>
<td>Software Engineer</td>
<td>Firmware Engineer</td>
<td>Software Engineer</td>
</tr>
<tr>
<td>Experience in that Role</td>
<td>4 years firmware specifically, 15 years general software</td>
<td>18 years</td>
<td>20+ years</td>
</tr>
<tr>
<td>Other UML Tools Experience</td>
<td>Rose Visio Rational Software Modeller</td>
<td>No</td>
<td>No UML but other CASE tools ProMod from G&amp;E Systems Westmount I-Case Ward/Mellor Cadre</td>
</tr>
<tr>
<td>General UML Training</td>
<td>Books</td>
<td>Books</td>
<td>No</td>
</tr>
</tbody>
</table>
User Discussion Summary

Several points were common across the users when discussing topics around using RoseRT. It should be noted that the User 2 discussion was not as comprehensive as the other users. A paragraph is dedicated to each point below:

All users made use of pen and paper drawings before starting to create a state machine diagram in RoseRT. The level of drawing was related to the experience of the user. The more experienced the user the less drawing was used. User 2 could provide evidence of this through their log book, they showed how log books from 6 years ago contained many drawings daily and later log books only had the occasional structure diagrams. User 1, the user with the least experience, explained this process in detail with statements like it was ‘easy to translate drawings and thoughts of state machines from paper, straight into RoseRT’ and ‘Paper drawings can capture ideas more freely and then the tool enforces constraint. Constraint is good because it provides certainty of how the state machine is generated.’

All users were happy with the way that state machine diagrams presented the problem and did not try to translate the diagram representation into some other form of mental model (such as source code). User 1 stated that they ‘don’t imagine anything more below the state diagram representation and don’t try and translate it to code’; User 2 thought ‘at a high level the state machine abstraction is all you need, state machines work well’, however ‘at the low level, approaching the hardware, the abstraction is not used due to code overhead and more efficient code can be created by hand.’

All users considered diagram layout extremely important and used the aesthetics of the diagram to indicate the quality of the implementation. User 1 stated ‘Diagrams should be neatly laid out. Visual cues of symmetry a really important. Errors show up in badly drawn parts of a diagram, based on own working patterns’. User 3 had the opposite working pattern, where they spent a lot of time getting the diagram looking good first and then ‘adds flesh’. An observation from looking at the diagram layouts was that they could not easily be automated; User 1 had a top down flow, User 2 had a top diagram with some parts hidden off-screen and User 2 had a “star” layout. The layout was highly dependent on the system being defined rather than common layout rules. Also User 2
used colour to indicate critical states, which meant that we could not automatically assume that colour could be used to highlight areas of a 3D diagram. In sympathy with the indication of the importance of diagram layout above, all users complained bitterly about the diagram features RoseRT provided. User 1 stated the ‘drawing tool is dated compared to something like Visio’. Users complained about the inability to access some code features through the diagram, such as not being able to edit or copy+paste code. User 3 stated RoseRT was the ‘worst diagramming tool’. This feedback indicates that users were not simply using the diagram as a means of writing code, they were strongly motivated to manipulate the diagram layout after the state machine definition had been captured.

Users complained about issues with working with diagrams in the area of event tracking. User 1 stated ‘I have to think a lot (“staring into space”) about whether the state diagram captures all possible events. Trying to “test by analysis”. Have I captured all possible permutations of events coming in and have I handled them sensibly’ and this can be 10-15 minutes at a time. User 1 also stated that ‘Sometimes I wonder if diagrams are a distraction. Sometimes I find the tree view more useful and having states in a hierarchy helps me think about the events.’ This issue was followed up at a later session and it was discovered that the “tree view” presented a list of all individual states within a state machine; this was then used as check list for thinking about the state machine. User 3 did a lot of ‘in the head prototyping’ and had another method of solving the issue of event tracking by using ‘RoseRT search functionality to handle large state machines and get a list of which states handles what events in the search results.’

User Observation Summary

In another session User 1 and User 2 were observed while they worked, with additional discussion arising from the observation. This section lists the main observation points:

User 1 had initially stated that they did not use RoseRT for debugging and this task was not included in their estimated hourly usage of RoseRT. Direct observation showed that they indeed used RoseRT for debugging however they did not use the visual debugging

---

20 In presentations of our research audience members have suggested many 3D diagram features, such as colourising diagrams to highlight them. On the surface these features seems logical to include, however this point highlights the need for User Task Analysis and to completely understand the implications of any feature. In this case, implementing colour on intuition would have caused issues for this user.
feature provided by RoseRT. Instead User 1 used source code debugging, with breakpoints in Visual Studio and console *printf*s to output variable information. However the diagram was still used to navigate the state machine structure and code.

User 2 had stated they used RoseRT for debugging; however their usage pattern was the same as User 1. The diagram was used as a high level view of the system but actual debugging was with breakpoints and console *printf*s.

A discussion with the team of all 5 engineers arose regarding the way debugging was undertaken, and why the RoseRT visual debugging (which is real-time animated state machine diagram) was not used. There was a suggestion that following animations across many diagrams was difficult and that a single 3D state machine may make visual debugging more useable. However further discussion highlighted another issue which was limited connectivity between the target hardware and the development PC. Only having a remote serial connection meant animation requiring TCP/IP was not possible regardless of diagram usability. Although the system is also simulated directly on a PC, with TCP/IP available, the debugging method used for the target became the method across all parts of the system.

**Additional Off-Site User Session**

An opportunity arose to undertake a user session with a former team member (*User 4*). This session was undertaken outside of the work environment and was used to provide additional information regarding the user task analysis.

User’s relevant details were

- No longer using RoseRT
- Last used March 2007 (3 months before the session)
- Used for 4 years constantly at approximately 35 hours a week
- Had done IBM Rational Training Course

Being careful not to ask leading questions we explained the 4 tasks of Analysis and Design, Creation, Refactoring and Test + Debug (detailed in the next section). User 4 agreed they were the main tasks he undertook and stated the refactoring phase took
20%-40% of his time in a new product, but could take up to 80% if an entire system was being refactored.

We asked User 4 if he used hierachal state machines and to talk about the issues. User 4’s issues were the same as the others, where he had trouble comprehending large state machines. However his solution was different, User 4 instead drew the necessary parts of a diagram on paper or created a UML activity diagram to cover bits he needed to understand.

**Overall User Task Summary**

From this study it was determined that the goal of all members of the team were the same, the creation of a product subsystem. Each user owned a package within the model, which integrated to the other subsystem packages. Even though the subsystems were different, the high level tasks were the same and were as follows:

- Analysis + Design (pen + paper designs and “in the head” thinking)
- Creating Structure and Behaviour (translating design in to implementation)
- Refactoring (refining implementations, fixing bugs and adding new features)
- Test + Debugging (testing implementation)

In the refactoring task, users were required to refactor state machine diagrams over 30%-40% of their week. These were deeply nested state machines and it was observed that while doing this task, users appeared to be compensating for a limitation in the UML representation of such diagrams. Each user had a different method for overcoming that limitation that did not involve the use of the diagram itself. The next sections explain the issue in more detail and our solutions proposed.
5.3.3 Cognitive Load in Considering State Machine Behaviour

The symptoms of the limitation in single level state machine views manifested in users “thinking outside of the diagram”. One user claimed to be often “staring into space for 15 minutes” trying to determine the consequences of deferring one event. Another used the “find” functionality to find all items in the model with an event name so they could check them off individually. Yet, another stated that they took to drawing existing state machine diagrams on paper to be able to follow events through levels.

To explain this further, we consider a small hierarchical state machine diagram taken from the RoseRT help documentation on “Transition Selection Rules” (Figure 5-4). This diagram is unusual as it displays the substates \( S11 \) and \( S12 \) in the same diagram as their superstate \( S1 \), which is not how RoseRT presents such diagrams; this gives some clue of the need to display the complete state machine when considering events, to enable better understanding.

![Hierarchical state machine diagram](image)

*Figure 5-4 – Hierarchical state machine diagram from RoseRT documentation*

State machines are driven by events which trigger transitions; however the visual notation is aimed only at representing the behaviour through notation such as states and the transitions between states. The presence of events may be indicated with text notation, either implicitly through transition names or explicitly through text adornment. For small state machines, this may be sufficient, however as the complexity of a state machine increases, the complexity of considering events increases dramatically. The cognitive load on the software engineer quickly becomes very large when considering
the possible permutations of the state machine behaviour, especially for hierarchical state machines.

We now consider the following case study. Assume that this state machine (Figure 5-4) is driven by only two events. Let us imagine that we are a software engineer who must make a change to $S_{12}$. In order to implement this change we need to consider the impact of the possible incoming events on both $S_{12}$ and $S_1$. As $S_{12}$ is a substate of $S_1$, $S_1$ must handle events “let through” by $S_{12}$ (i.e. events not handled or deferred).

Now imagine that Figure 5-4 consists of two diagrams (Top and $S_1$), as it would presented in RoseRT and that we can only view one diagram at a time. In considering $S_{12}$ we now have one extra “off-screen” diagram that needs to be thought about outside of the current view. If we were to add another substate level in $S_{12}$ as part of that change (e.g. $S_{121}$), we would have two “off-screen” diagrams to consider when working on $S_{121}$.

As we see, the amount of “thinking outside of the diagram” is easily doubled, tripled or more depending on the hierarchical depth of the state machine being represented.

5.3.4 Cognitive Off-Loading Using 3D and Event Notation

This section explains how 3D state machine diagrams and event notation may help the software engineer in cognition of hierarchical state machine diagrams. 3D allows the complete state machine to be viewed in one single view. Event notation is suggested as providing an additional visual summary of where events are handled. Figure 5-5, Figure 5-6 and Figure 5-7 below summarise what information the UML presents to the software engineer and what they are required to imagine or think about to understand the effect of a single event on a state machine hierarchy. It must be stressed that at this stage of the evaluation process, neither the negative nor positive aspects of the visualisation have been tested. It is likely that aspects of the new visualisation may increase cognitive load in other areas of interpreting the information displayed. This section presents purely the proposed solution and how it has been derived from our user task analysis.
In the current situation the engineer is only able to view one state machine diagram at a time. In considering how a single event may be processed by the state machine, they must think about all superstates and how they may handle that event.

With a 3D state machine diagram the engineer is able to view all state machine diagrams in the hierarchy and only needs to think about how each may handle that event.

With combined 3D and event notation the engineer need only think about the events that have impact on specific states levels, irrelevant diagrams can be ignored.
5.3.5 Cognitive Load Metrics

Our initial state machine survey (Table 5-1) only looked at state depth (i.e. individual states within a machine) and included “empty” non-behavioural state machines, however, after our user study we now understand that a better measure of complexity is state machine diagram depth. The software engineer is not merely interested in the states themselves, but they are primarily interested in the behavioural aspects of the diagram as a whole (states, transitions, choice points etc) and the state machine diagrams higher in the hierarchy which participate in that behaviour.

To analyse the extent of diagram hierarchies (as opposed to state levels), a new RoseRT script was created to count used diagrams and depth. In addition to this, the script calculated an “off-screen” metric based on the average number of additional diagrams that had to be considered for any given diagram in the hierarchy. To gather metrics, companies from the previous survey were contacted and the results of 664 state machines from 4 models are shown in Table 5-3.

Our survey shows that hierarchical state machines are used a large amount of the time (37.2%-64.4%). This result is similar to our original substate usage survey; however with more detailed information it is now revealed that between 55.74%-89.85% of all state machine diagrams (either substate or superstate) form part of a hierarchical structure. This high number is due to hierarchical state machines containing more diagrams. For the most complex state machine found in the survey the software engineer had to consider, on average, 3.25 other diagrams for every diagram they worked on and that state machine contained 16 diagrams in total.
Table 5-3 – Metrics for 4 models from industries surveyed in Table 5-1 and the user task study subsystem. Off-screen diagrams is a measure of the average number of diagrams that need to be considered for any given diagram within a hierarchical state machine

<table>
<thead>
<tr>
<th>Totals</th>
<th>Robotics</th>
<th>Robotics</th>
<th>Subsystem</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Layer State Machines</td>
<td>27</td>
<td>67</td>
<td>n/a</td>
<td>177</td>
</tr>
<tr>
<td>Hierarchical State Machines</td>
<td>16</td>
<td>54</td>
<td>3</td>
<td>320</td>
</tr>
<tr>
<td>Percentage Hierarchical</td>
<td>37.2%</td>
<td>44.6%</td>
<td>n/a</td>
<td>64.4%</td>
</tr>
<tr>
<td>State/Substate Diagrams</td>
<td>61</td>
<td>223</td>
<td>n/a</td>
<td>1743</td>
</tr>
<tr>
<td>Percent Diagrams Hierarchical</td>
<td>55.74%</td>
<td>69.96%</td>
<td>n/a</td>
<td>89.85%</td>
</tr>
<tr>
<td>Average Off-screen &gt;= 2</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>Average Off-screen &gt;= 1</td>
<td>1</td>
<td>12</td>
<td>3</td>
<td>129</td>
</tr>
<tr>
<td>Max. Average Off-screen</td>
<td>0.75</td>
<td>2</td>
<td>3.25</td>
<td>3.16</td>
</tr>
<tr>
<td>Max. Diagram Size</td>
<td>4</td>
<td>10</td>
<td>16</td>
<td>19</td>
</tr>
</tbody>
</table>

Our survey in Table 5-3 clearly shows that any improvements in the area of understanding hierarchical state machine diagrams would be of great benefit to the software engineers work load. The high percentage of hierarchal state machine diagrams in each model shows they form the majority of work. The survey further shows that this work can be complex, involving sometimes as many as 19 diagrams in a single state machine. Complexity is also measured in the need to consider, on average, 1, 2 or more “off-screen” diagrams for each diagram worked on within each model.

If other aspects of 3D visualisation (such as navigation issues etc) were known to be insignificant, then this survey would be evidence that 3D UML state machine diagrams would be of benefit to engineers by virtue of intuitively presenting them missing information required to do their tasks. Other aspects of 3D visualisation though are not insignificant; the question is how much benefit has been gained compared to what may be lost to other aspects, such as navigation. The next phases of our research proceed to answer these questions.

5.4 Heuristic Evaluation

In this section we present the Heuristic Evaluation outline and results. These results summarise the outcome of directly applying the methodology to a specific visualisation. Due to the low response rate, the same visualisation was also evaluated using Formative Evaluation with the results reported later in Section 5.5.1.
5.4.1 Heuristic Evaluation Outline

*Heuristic Evaluation* is intended to obtain feedback from expert evaluators so that the initial visualisation can be refined before more user intensive testing is undertaken as part of the *Formative Evaluation* phase. In addition to this, to mitigate the risk that the user tasks defined may have been unusual or company specific, feedback was also sought on how the tasks might compare with usage patterns in other companies.

Based on our user task analysis results, examples of 3D UML state machine diagrams were created with event notation, generated from the “Traffic Light” example model provided with RoseRT (Figure 5-8). Event notation took the form of “event summaries” indicated at the top left of each diagram with a point and associated event name. If the same event was processed at different layers, the points were linked as a visual cue to which diagrams need to be considered for each event.

*Figure 5-8 – 3D UML state machine diagram derived from user task analysis. The state machine diagram presents all diagrams in a hierarchy in a single 3D dynamic view and provides additional information about the events that drive the state machine behaviour.*

[The associated x3d file is available for viewing at the following URL: www.x3d-uml.org/Publications/2008_X3D-UML:_3D_UML_State_Machine_Diagrams]
Nine experts in the area of RoseRT and UML were contacted and provided with information on the defined user tasks, an evaluation report form to complete and the proposed 3D solution in the form of an X3D file which could be viewed with a browser. Evaluators were asked to “evaluate the 3D state machine diagram examples given, against the task of refactoring existing state machine diagrams.”

The complete Heuristic Evaluation Sheet is given in Appendix B. In addition to the evaluation questions, this describes the background given to evaluators, the user tasks described, the proposed 3D UML state machine diagram solution and the event notation extension.

5.4.2 Heuristic Evaluation Results

In the time frame given to evaluators, two evaluation reports were received. Evaluator 1, was a former IBM Senior Software Engineering Specialist with over 6 years experience ‘with many (hundreds)’ of RoseRT customers and Evaluator 2, a Senior Software Engineering Consultant in real-time model driven development.

The overall opinion of the 3D state machine diagram from Evaluator 1 was that ‘Gestures for navigation and manipulation will be difficult, but the overall development experience would be served well I think.’ Evaluator 2 was already of the opinion that ‘a 3D view would be helpful’ and suggested the idea be taken further.

The results of the evaluation are summarised in the following sections.

User Task Question Responses

With the Refactoring task defined as, ‘refining implementations, fixing bugs and adding new features’, the responses indicate that our user task analysis was indeed representative of industry and may also be applicable to the UML tool “Rhapsody”.

Evaluator 1 User Task Responses Summary:

‘Refactoring a state machine is indeed a problem and large hierarchies can quickly become difficult to manage’ and ‘This is mainly a problem with deep hierarchies or hierarchies that have been developed by someone else, e.g. a previous other, or teams’.
The estimate for weekly task percentages was ‘30% structure 30% behavior (state machines), 30% design review’.

Evaluator 2 User Task Responses Summary:

For refactoring state machine diagrams the ‘experience is similar’. For task percentages the estimate was that ‘approx 30-45% of “normal/productive” modelling involves defining and refining behaviour and 30-40% defining and refining structure…’ however actual ‘redesigning … should not be >=20% of the effort’.

3D State Machine Diagrams and Event Notation Question Responses

The following summarises common and specific responses given in relation to 3D usability issues and event notation questions from both evaluators:

**Common Responses** (In order of Evaluator 1 and Evaluator 2):

- **Layout** – ‘The lay-out needs to be applied in such a way that the 3D version of the diagram is usable’ and ‘a new style of state diagram layout algorithm’ is required.

- **Navigation** – ‘The 3D browser doesn’t really make navigation easy’ and ‘an effective means to allow users to adjust to the optimal perspective’ is required.

- **Filtering** – ‘The 3D diagram contains a lot of information. State machines work well partially because the amount of information in a diagram is strongly reduced’ and ‘eliding all of the states not “enclosed” by the chosen state... can hide a lot of unnecessary detail’

**Evaluator 1 Specific Responses:**

- **Visual Connection between Layers** - An enhancement would be a ‘connection between a state at level N and the fold-out of it at level N+1’

**Evaluator 2 Specific Responses:**

- **Print Layout** – ‘printing 3D will be almost as important as on-screen visualization.’
Transition paths between State Layers – ‘show the transitions between layers’

Behaviour specific “Slices” – ‘it would be nice to view “slices” of a hierarchical state diagram improve understanding the design of the state machine’s behaviour.’

On the questions related to the benefit of the addition of event notation, Evaluator 1 did not ‘have a strong opinion’ either way. Evaluator 2 stated ‘yes’ it would be of benefit.

5.5 Formative Evaluation

In this section we present the Formative Evaluation phase and results. The goal of this phase is to walk through the visualisation with the user to gain feedback required to refine the visualisation to a point where empirical testing can be undertaken in the Summative Evaluation phase. In this phase, two types of sessions were undertaken, an “off-site” session which was used to support the Heuristic Evaluation phase (due to the poor response rate) and “on-site” sessions with the refined visualisation.

5.5.1 Off-Site Formative Session

Prior to revisiting the site where the initial user task analysis was undertaken, a Formative session was undertaken with a former team member using the “Traffic Light” example used in Heuristic Evaluation (Figure 5-8). The session was designed to provide additional information prior to refining the visualisation for on-site sessions. The user expressed that, as it was, the visualisation would be of benefit and suggested the following improvements:

- There needed to be a stronger visual link between substates and superstates.
- The event notation lines suggested incorrect relationships between states.
- Visually linking junction points between substates and superstates (as needed) would help when following transition paths.
- Showing (as needed) which transitions were triggered by each event would be useful (i.e. linking event notation points to their associated triggers).
• It would be useful to be able to exclude non-relevant state branches.

• There is even more potential benefit in 3D UML structure diagrams\(^{22}\).

### 5.5.2 3D UML State Machine Diagram Refinements

From the *Heuristic Evaluation* results and the Off-Site *Formative Evaluation* session results the following conclusions have been made regarding the visualisation and the refinements needed to better suit users’ tasks.

• There is definite and consistent perceived benefit in the 3D UML state machine diagram, prior to refinements.

• More precise user requirements are uncovered from walking through the 3D visualisation (i.e. user tasks become better defined).

• Layout of the diagram needs to be addressed.

• Navigation within the diagram needs to be addressed.

• Connections between substates and superstates need to be clearer.

• Dynamic user controlled filtering of irrelevant data is required.

• Dynamic user controlled presentation of more data is required.

• Event notation links were not strongly desired and had negative aspects.

---

\(^{22}\) In RoseRT, state machines are contained in *Capsules* (also known as *Active Classes*). A system is made up of a number of Capsules communicating and system structure is formed by connected *Capsules* within *Capsules*. 3D UML *Structure Diagrams* would achieve a similar effect as 3D UML *State Machine Diagrams* by allowing an engineer to see a complete system rather than a series of structure diagrams representing each layer of the system. Although we recognise this as an issue, we considered this area of research as too RoseRT specific compared to the more common state machines.
Based on these conclusions the visualisation (Figure 5-9) was refined in the following ways:

- Event notation lines, junction point lines and other data enhancements were not implemented to determine through user feedback if these were indeed required for tasks (i.e. gain more precise user requirements in this area).
- Event notation summaries were listed at the bottom of diagrams.
- The user was provided with the ability to create and save a diagram layout.
- The ability to reposition the whole diagram in the view was introduced.
- Transparent connection “cones” visually linked superstates and substates.
- The ability to “shrink” branches to filter data was introduced.

*Figure 5-9 – Refined 3D UML state machine diagram example. The “working” branch is laid out for easy viewing and the “initializing” branch, which is not of interest, is “shrunk” (top right). Event notation is shown as a summary at the bottom of applicable state machine.*

---

23 The associated x3d file is available for viewing at the following URL: www.x3d-uml.org/Publications/2008_X3D-UML:_3D_UML_State_Machine_Diagrams
5.5.3 On-Site Formative Session

With the refined 3D diagram, an on-site visit with the original RoseRT team was arranged. Unfortunately one of the three original users was not available so the session was undertaken with only 2 users. In this session the users were individually walked through a 10 minute training session with the new visualisation based on the “Traffic Light” example model (Figure 5-9) and were taught basic X3D browser navigation techniques; such as how to centre the diagram, rotate and examine, layout the diagram and save that layout.

Users were then asked to pick an actual state machine diagram from a model they were working on and, from the chosen diagram, a 3D state machine diagram was generated. Users were then asked to walk through the diagram and comment on the features they need to do their job, as well as usability aspects. An example of the most complex state machine diagram generated is given in Figure 5-10, with this diagram having an “off-screen” average of 3.25 diagrams (see Section 5.3.5 Cognitive Load Metrics).

All users stated that they saw benefit in the 3D UML state machine diagram and the event notation. The users stated this in strong terms with one user stating ‘of course’ (it would be useful) as though it was such an obvious question it should not be asked, and another stating it would be ‘definitely useful’.
The users each spent 30 minutes walking through the diagram and commenting on improvements. As part of this process a new low-level user task was uncovered related to tracing transitions through multiple diagram levels for a specific purpose. Users needed to logically highlight transition “chains” (i.e. groups of transitions) to analyse particular aspects of a state machine and these transition groups often spanned multiple levels of diagram. This user task was a logical extension to linking junction points between diagrams, as they form part of a transition chain.

The improvements suggested by the users are grouped together below in their respective categories:

Note: The points are numbered for reference in Section 4.6.9, where actual user responses are compared to refined heuristic evaluation.

---

24 A video example is available at www.x3d-uml.org/YouTube_Examples
1. **Dynamic Features for Multi-Level Transition Tracing**

1.1. Highlight transition chains of interest.

1.1.1. User 1 needed to see junction linkages between state levels, such as selecting a parent state and seeing all junction links.

1.1.2. User 1 stated strongly that junction point linking between state levels would be the ‘killer app’.

1.1.3. User 2 stated they needed to create path highlighting i.e. selecting paths of interest and highlighting them to help someone follow a path through the system.

1.1.4. User 2 stated the ideal would be able to ‘flatten the problem’. 3D allows a multilevel view and then you should be able to isolate aspects of the view. ‘You should be able to take the view to team mates and talk to a highlighted problem.’

1.2. Linking transition relationships at the same level in the hierarchy.

1.2.1. User 2 suggested that temporary single layer diagrams could be made in situ by linking transition paths at the same branch level i.e. the 3D view allowed substate diagrams to be merged to understand particular problems.

1.2.2. User 2 needed this to be dynamic stating ‘The point of nesting is removing transition information for clarity’

2. **Essential Data**

2.1. Dynamic junction point linking between substate and superstates.

2.1.1. User 2 required this linking and further suggested combining this with *Linking transition relationships at the same level in the hierarchy* (above)

3. **Navigation Improvements**

3.1. Restrict navigation options to prevent awkward views
3.1.1. User 1 stated the navigation was much too free and should be restricted to avoid the “batman” view of diagram on an awkward angle (“Batman” refers to the 1960’s TV series where tilted camera’s were used)

3.1.2. User 1 suggested restrict movement to up/down, left/right

3.2. Allow quick navigation to individual state machine diagrams.

3.2.1. User 1 would like to ‘snap’ to a state diagram (i.e. semantic zoom)

3.2.2. User 1 was not concerned about shrinking sections of the diagram away, they would much rather leave the diagram as a whole and have rapid navigation (i.e. snap to).

3.3. Hot-keys to navigate to different view aspects.

3.3.1. User 1 (a gamer) suggested navigation with hotkeys

4. Layout Improvements

4.1. Locking lower level state diagram positions

4.1.1. User 1 stated the manual layout method should have an option to not move children (i.e. not move the entire tree, so the diagram could be positioned by itself).

5. User Preferences

5.1. Opacity configuration to suit user preferences and different graphics capabilities of machines.

5.1.1. User 1 ‘preferred the diagrams to be completely opaque’

5.1.2. User 1 found that they could not see the transparent connection cones on a VMware image; however on another PC the cones needed to be more transparent.
5.1.3. User 1 found cone annoying when concentrating on single diagram, and suggested it should perhaps disappear when focusing on a single diagram.

5.1.4. User 2 ‘didn’t care either way’ about the transparency of diagrams.

5.2. Ability to adjust layout diagram depth to suit diagram size.

5.2.1. User 1 stated they would ‘like to stretch the diagram level depth to suit the diagram, so they could get a tree view from the side.’

6. Recreating 2D Features

6.1. Recreate features that exist in the RoseRT diagrams

6.1.1. User 2 would like to ‘hover over junction points to see event info’

6.1.2. User 2 would also like to hover over transitions and see the internal source code.

The above list has captured many issues that are needed to be addressed before empirical tests can be undertaken. Although the user perception of the 3D UML state machine diagram is positive, there needs to be many interaction techniques included to effectively enable the users to complete the tasks that we plan to measure benefit against. Although all the issues have possible solutions, the implications of implementing them have significant impact on undertaking Summative Evaluation, explained in the next section.

5.6 Summative Evaluation

Summative Evaluation is based on testing the performance of users completing user task scenarios. The performance is measured through users completing the same tasks in the 2D diagram compared to the 3D diagram. For Summative Evaluation to be effective, issues raised in the Formative Evaluation stage need to be addressed. The outcome of Formative Evaluation and the number of issues however has significant impact on the ability to undertake empirical testing.
The issues and solutions with applying *Summative Evaluation* with respect to state machine diagrams have already been presented in Section 4.8. These are summarised here:

a) To increase the test population from 3 users, requires the 3D UML diagram to be implemented as an add-in that can be sent to many companies (such as those surveyed). This add-in would be intended to gather performance data while users work on their everyday tasks.

b) The number of issues to be addressed would require significant effort and close collaboration with users to ensure implementation is bug free. A complication to this is that we have participants volunteering from both Windows and Linux environments, meaning a cross-platform solution is also required.

c) To prevent collaboration with the researcher influencing research results communication with users needs to be carefully controlled. A sufficient period of time also needs to be given to allow users fall into a normal routine without communicating with the researcher at all.

The above issues mean that an effective *Summative Evaluation* is not possible in the timeframe of this thesis. The results of *User Task Analysis*, *Heuristic Evaluation* and *Formative Evaluation* stages give consistent evidence that there is benefit in 3D UML state machine diagrams. The stages also highlight the issues that need to be addressed to measure the extent of that benefit.
5.7 Summary

The results of our research present consistent evidence that there is benefit in 3D UML state machine diagrams. The analysis highlights a common and critical software engineering task that benefits from an alternative UML visualisation provided by 3D. Both user and expert evaluation sessions provide qualitative evidence that benefit already exists in our 3D UML state machine diagram, however further refinements have been suggested to capitalise on 3D visualisation possibilities and improve navigation, to increase that benefit further.

The benefit of 3D UML state machine diagrams is in the area of understanding hierarchical state machine diagrams. Both user and expert responses show that this is an area of known concern for those in industry and our analysis captures the extent. For RoseRT engineers, defining state machine behaviour is a task which occupies 30%-40% of their effort and our model metrics survey shows that between 56%-90% of that effort is likely to be related to hierarchical state machines. We further quantify the issue through a metric of “off-screen” information missing from the engineers’ view which indicates the level of “out of diagram thinking” currently required.

Our approach however highlights that for effective utilisation of 3D UML state machine diagrams many complex interaction techniques need to be considered and implemented. Without these interaction techniques it would be premature to undertake empirical testing on a visualisation with known limitations.

We see that benefit in 3D UML state machine diagrams is in the ability to have a single diagram view that can then been manipulated to suit the specific user task while maintaining a consistent mental model.
Chapter 6 - 3D UML Mechatronic and Holistic Diagrams

6.1 Overview

In this chapter we present our 3D extensions to UML in the areas of 3D UML Mechatronic Diagrams and 3D UML Holistic Diagrams. For 3D UML Mechatronic Diagrams we consider the integration of 3D UML diagrams and physical views of a running system to aid in debugging. This investigation, which is described in detail in Section 6.2 leverages existing 3D UML State Machine Diagrams and ARToolKit [8], to allow real-time debugging of a Lego robot. In Section 6.3 we present the user study into the issues in using the open source project OpenSceneGraph with the view of implementing and evaluating 3D UML Holistic Diagrams.

6.2 3D UML Mechatronic Diagrams

This section outlines the results of applying 3D UML Sequential Evaluation to 3D UML Mechatronic Diagrams. To undertake the research using the methodology, the first issue was finding participants who were actual users working with UML on mechatronic systems. Several avenues were tried for locating users, with the final result being exploratory user task analysis from a number of areas. The main study was undertaken as student project, where an RMIT post-graduate student developed an integrated 3D
UML solution with our aid and also adapted an existing RoseRT UML model to execute on a Lego NXT robot [56].

First we present an overview of the concept of using 3D to aid in visual debugging of integrated hardware and software systems. Next in Section 6.2.2 we present three projects targeted by our study, with the resulting user task analyses in Section 6.2.3. In Section 6.2.4 we present how the 3D UML view was implemented, with Section 6.2.5 covering the experiences in using the view. Finally we summarise 3D UML Mechatronic Diagrams in Section 6.2.6.

6.2.1 Overview

The intent of our investigation was to integrate a UML software model and a mechanical model (such as a CAD model) to create our 3D UML Mechatronic Diagram, a 3D view of a system which presents both software and mechanical attributes of the system. The motivation was to test the use of this combined view during the design of a system.

Rather than relying on CAD model data however, we extended the concept through Augmented Reality. Although CAD model data for a system under development may not be available to a software developer, the physical devices are and because our 3D UML state machine diagrams are complete 3D models, they could be incorporated into scenes with the use of ARToolKit [8]. ARToolkit is a software library for overlaying 3D models of objects onto real-time images of real objects. In Figure 6-1 we see our prototype of the concept with a RoseRT state machine diagram projected on a development board. The state machine is generated from RoseRT, saved as VRML and overlaid using ARToolkit in a position relative to the CPU running the software. This view allows the software developer to see both the hardware and software at the same time.
In considering actual user tasks, if we refer to the RoseRT user task analysis results for state machines in Section 5.3.2, we recall that state machine behaviour was tightly coupled to hardware. User 3’s task had been to take the hardware datasheet and ‘use state machine diagrams to toggle lines’. Considering this, we decided that a view which visualises the state machine and the associated lines should aid such a developer in debugging a system.

One issue however is that 2D visual debugging of state machines is already a feature supplied by RoseRT. However this feature was rarely used and instead the preference is to debug with `printf` statements to a console. Through observation and our discussions with the RoseRT team it was thought that this is due to part of the system being limited to serial communications. RoseRT visual debugging requires TCP/IP connectivity, so visual debugging of the embedded device was not possible and therefore `printf` statement debugging became the default means across the entire system.

---

25 A video example is available at www.x3d-uml.org/YouTube_Examples
In considering the above, our approach aims to enable 3D visual debugging in a manner that addresses connectivity issues with the target hardware. By using ARToolkit we aim to animate the state machine as a visual debugging tool. This animation is to be possible at the lowest level of connectivity to the target hardware under development. In theory, even if only the most primitive debug tool of a “flashing a LED” were available because we are using a camera to view the scene we can use image recognition to translate flashes into visualised states.

6.2.2 Projects

To locate actual users for participation in our study, we investigated a number of possibilities. The following projects were investigated as potential case studies for our research:

Project 1 - University of Paderborn RailCab Project

A potential candidate for researching 3D UML Mechatronic Diagrams was the University of Paderborn RailCab Project[88] research initiative. This project is an autonomous rail system which enables specially designed rail carriages to behave as driverless taxi cabs on the existing rail network. The software engineering portion was designed using UML and has lead the University of Paderborn to development a UML tool called Fujaba [90] and the Mechatronic UML model-driven development approach [81]. In the area of Lego NXT robot development, the Fujaba tool has the ability to generate code through a Lego NXT Library [51].

Project 2 - RMIT Manufacturing Cell Student Project

Another potential candidate for a project was RMIT student projects, which are projects undertaken for 2 months over the Australian summer break (at the end on one year and the beginning of the next). One such project was the creation of a mechatronic model of a manufacturing cell [58] using Lego NXT robots [56]. The specification for the mechatronic model was given as a synchronous state machine, where each component of the system has to pass through specified states to complete a given task. Each component also had to synchronise at particular points in their process to collaborate with another component to complete a given task. The specific manufacturing cell had to
pass parts between processes and this interaction between robot processes became the synchronisation point.

**Project 3 - RMIT 3D UML Robot Debugging Student Project**

To allow us to leverage the manufacturing cell project above, we instigated our own student project to develop an animated 3D UML state machine to aid in the debugging of Lego NXT robots. The aim was to extend existing 3D UML state machine diagram code to enable simple real-time highlighting of individual states and provide debugging support to the manufacturing cell project.

**6.2.3 User Task Analysis**

In this section we present our findings from *User Task Analysis*. *User Task Analysis* captures representative user task scenarios through one-to-one sessions with users debugging mechatronic systems i.e. systems that use a combination of hardware and software. Participants in this area were not readily available so a comprehensive study, involving teams of experienced users and the collection of model metrics, was not possible. We present our analysis from investigations with a number of participants from the different projects described in the previous section.

**Project 1 - University of Paderborn RailCab Project**

On the invitation from Prof. Dr. Wilhelm Schäfer, the RailCab project was visited with a view to collaborating in this area and extending Fujaba with 3D UML. From discussions with researchers at the University of Paderborn it was considered that there would be benefit in a visual validation tool, to complement their model based validation approach. Their mechatronic system involved not only complex interactions between control software systems and hardware, but complex interactions between systems, where each RailCab (a system in itself) formed convoys with up to four other RailCabs.

Evidence for the need of visual validation was observed with the RailCab project already making use of augmented reality for observing mechanical behaviour [89]. Though 3D state machine diagrams were not directly used, we were shown that aspects of state machine data were presented in at least one simulation user interface to indicate system state.
However, in the area of using Fujaba for Lego NXT development we found that existing models did not make use of state machine generated code.

The conclusions from our investigation is that there is evidence that 3D UML Mechatronic Diagrams may be of benefit to the RailCab project, this is seen in the directions that project has already taken. However in the area of Lego NXT development, the particular 3D UML state machines diagrams we are proposing would not be of benefit because state machines diagrams are not found in current Lego NXT projects.

**Project 2 - RMIT Manufacturing Cell Student Project**

For the Manufacturing Cell Project, two students collaborated to produce a manufacturing production cell consisting of two Lego NXT robots. To not influence the students, they were left to undertake the project with whatever means they felt were appropriate. As the original specification was given as a state machine, it was hoped that the students would make use of a state machine based tool to develop the system. Although the students did investigate such tools, the final development was undertaken with “hand coded” Java.

For this project we could not investigate state machine usage; however the development methods that the project team did utilise could be used as input to our study. For example they made use of Lejos [57] and Bluetooth for debugging the system remotely. This revealed that the Lejos debug capability is limited to a console application retrieving system information explicitly sent by the developer (similar to printf statements). This is needed because there is no direct debug target runtime visibility, such as stepping through code, once the application has been downloaded to the target.

**Project 3 - RMIT 3D UML Robot Debugging Student Project**

For the student project under our control, we instructed the student to develop a prototype using an existing Lego model which had been developed in RoseRT. Although we realised we could not get effective user task analysis data based on one user completing work under instruction, this case study did enable the use of 3D for debugging using actual data. The remainder of this section outlines the case study undertaken.
The RoseRT model used for the study was called the Lego “Rover”. This model demonstrates the use of RoseRT to generate small footprint C code from passive\textsuperscript{26} state machines. The RoseRT model was originally created in 2003 for demonstration purposes at Rational conferences and it targeted a C runtime and an early version of Lego “RCX” robot. The current Lego NXT target uses a Lejos [57] Java runtime, of which RoseRT does not support code generation for passive classes. As part of the project the student was required to hand code the state machine implementation to meet the RoseRT model. We also adapted the model to cater for differences in Lego hardware.

In Figure 6-2 we present the original state machine for the main behaviour of the Lego “Rover” robot. The Lego hardware, for which this was originally designed, had the ability to detect left and right presses of a sensor. The robot logic made use of this data to alter its direction based on which sensor was triggered, enabling the robot to “rove” an enclosure. For our implementation the Lego robot only had a single sensor, so the state machine was adapted (Figure 6-3) to go backwards and forwards with no attempt to change direction. This behaviour we named “HeadBanger” because the robot was only designed to repeatedly bang into a wall.

\begin{center}
\begin{tikzpicture}
  % State machine diagram code here
\end{tikzpicture}
\end{center}

\textit{Figure 6-2 – The original RCX Rover main robot state machine diagram. This was designed for a robot which could sense left and right presses of a sensor.}

\textsuperscript{26} Passive state machines are state machines that do not use the supporting RoseRT runtime library. The state machine generates framework code and the developer implements all behaviour aspects themselves.
Figure 6-3 – New NXT “HeadBanger” state machines based on the original RCX “Rover”. This is the redesigned logic of the state machine to work with a robot with only a single sensor. The top state machine represents the new single sensor state machine.

For this project we see that the task is to create robot behaviour hand coded to meet the RoseRT state machine specification. This task follows a typical Lejos development process as per the Manufacturing Cell project. The only difference with the 3D UML project is that the student will drive a 3D UML visualisation rather than send text debug messages to a remote console.

In the next section we outline the implementation of the 3D UML Mechatronic Diagram with experiences of using the diagram given in the subsequent Section 6.2.5.
6.2.4 X3D-UML Implementation

In this section we describe how we implemented the 3D UML Mechatronic Diagram. The system architecture is shown in Figure 6-4. On the left we have a notebook computer as the development environment, this has osgPython 2.2.0 [75] installed which has an augmented reality viewer included. Through a web camera on the notebook we can then view the robot functioning on the right. The augmented reality viewer uses the “Hiro” position marker to project the state machine diagrams into the view at the correct size and location. On the right we present the Lego robot components the position marker, the Lego NXT controller and the touch sensor. The Lego NXT main controller reports the system state back to the development environment via Bluetooth and the development environment updates the state machine view.

![System Diagram]

*Figure 6-4 – The system for animated 3D UML state machine diagrams*
Figure 6-5 shows the 3D UML Mechatronic Diagram debug view as seen by the developer. The steps for animating the 3D UML state machine diagrams and gathering the debug information are given below:

1. Creating the 3D UML Mechatronic Diagram
   a. RoseRT state machine diagrams are exported as 3D models using X3D-UML OpenSceneGraph implementation.
   b. Using the exported diagram, a copy is made for each possible state and the state and associated transition are manually highlighted.

2. Animation Loop
   a. On every change of state the Lego robot writes state information to its Bluetooth communication link.
   b. A Java process on the development notebook reads data from the Lego robot and updates a local “states.txt” file with the robot state.
   c. osgPython is used to view the augmented reality scene through the camera
   d. The osgPython viewer loop reads the “states.txt” file and based on the information contained projects a specific state machine diagram into the scene.
6.2.5 Formative Evaluation Results

In this section, we present the experiences of exploratory use of 3D UML Mechatronic Diagrams. Although the prototype has not been refined through Heuristic Evaluation, we can informally report on experiences of walking through visual debugging with the student. The walkthrough was aimed at looking at the features of visual debugging rather than specifically aimed at addressing a user task. In Figure 6-6 we see an example of visually debugging the Lego robot, with the diagrams from Figure 6-3 projected into the scene. Even with the low resolution pictures provided, a developer familiar with the state machine diagrams can see the current state of each component as the robot moves back and forth.
Figure 6-6 – Example of animated 3D UML Mechatronic Diagrams in action.\textsuperscript{27} The current state in each diagram is highlighted with red. The left frame shows the states of the robot in “forward” state and the sensor in “free” state. The right frame shows the robot as it touches the wall, with a state transition to “WaitingForReverse” state in reaction to the sensors state transition to “Pressed” state.

Although the robot model used is of a basic design, with the sole intent of repeatedly bumping into a wall, the observations revealed benefit even for such a simple scenario. Despite the simplicity we observed complex interactions between hardware and software that are not normally revealed. The ability to view similar interactions would have even more benefit in a more complex system. From the walkthrough the following observations of benefit were made:

a) The visualisation showed directly how variability in hardware behaviour can influence system behaviour. For example, rather than the robot continuing to go back and forth as designed, variations in “straight” line travel were introduced by the swivel caster-style rear wheel and slight differences in front wheel sizes. Through the visual debugging we could see the impact this had with the robot slowly changing angle and we could observer and identify the exact point when the angle of impact of the robot hitting the wall failed to trigger the touch sensor.

b) State machine views gave a clear indication of hidden problems within a working system. This was observed during the student’s development of the robot behaviour, when we noticed through visual debugging that the touch sensor did

\textsuperscript{27} A video example is available at www.x3d-uml.org/YouTube_Examples
not change state when the robot reversed. Although the system was performing its designed function, the visualisation made it obvious the internal system state did not match what was expected with the observed physical behaviour; on inspection it revealed that the implementation of the touch sensor state machine had a logic error.

c) Diagrams gave visual cues which are able to be interpreted in situations where the text can not be seen clearly. If we refer to Figure 6-6, although the text of the state machines could not be seen, the layout is clearly visible. This provides a visual cue that is more readily interpreted from distances and angles compared to using text output on the LCD screen provided for the robot.

Although we observed positive benefits in using visual debugging there were issues with the concept. These issues are outlined below:

a) The ARToolkit uses the camera input for image recognition and is very reliant on specific lighting. If other dark objects are present in the scene this reduced performance significantly. For visual debugging to work effectively the user would need to control the debugging situation to suit the ARToolKit.

b) To allow the visualisation to work on a small robot, we reduced the size of the “hiro” position marker. We found using low quality web cameras further complicated image recognition and we needed higher quality cameras\(^\text{28}\) to work effectively.

c) The ARToolkit does not provide a recording functionality which made it difficult to capture and replay the debugging events.

d) The visualisation needed to be integrated into the IDE as the set-up of a debug session required a number of steps within different tools. The difficulty in this process could deter the use of visual debugging.

\(^{28}\) “Quality” in this situation refers to clarity of image. All 3 cameras we tried stated 640x480 picture resolutions, however there was noticeable difference in the clarity of image from the more expensive camera, especially in internal lighting situations as would be expected in debugging situations.
6.2.6 Conclusions

In this section we have presented applying *3D UML Mechatronic Diagrams* to state machine visual debugging of a number of different Lego NXT projects. Our case study reveals some evidence as to the benefit of our *3D UML Mechatronic Diagrams* however we also reveal that this benefit needs to be tested on projects other than those involving Lego NXT. Although tools exist to develop code for Lego robots through state machines, these are not commonly used and therefore the task of debugging state machines is not a common or critical task in this context.

Our user task analysis has shown that there is a desire for visual validation, such as in the RailCab project. We have shown that in practice *3D UML Mechatronic Diagrams* can be used for visual validation and they do uncover problems that are not evident in typical debugging scenarios. This evidence suggests that further research in this area is warranted and that applying the same technology to state machine based mechatronic system should provide measurable benefit. This benefit is most likely in the areas of detecting incorrect state machine behaviour in a functioning system and in more effectively isolating issues related to the runtime behaviour of integrated software and hardware.

We should also note that *3D UML Mechatronic Diagrams* further strengthens the argument for our approach to 3D UML visualisation. We have previously shown benefit in generating a complete 3D description of state machine diagrams for the task of understanding complex hierarchies. It could be argued that the benefits of *3D UML State Machine Diagrams* described in Chapter 5 could be achieved in other ways, such as layered planes or some other 2D technique. However, due to our underlying 3D model, we are able to reapply these diagrams to solve a different problem in a different context. In this example as we have modelled the UML in 3D we are able to leverage other 3D tools, such as ARToolkit, to create an augmented reality debugging tool.

6.3 3D UML Holistic Diagrams

This section outlines the results of applying *3D UML Sequential Evaluation* to *3D UML Holistic Diagrams*. For this study we focus on developers using an object oriented 3D open source library called *OpenSceneGraph* [73]. The reasoning for this was to look at the question of benefit from a different perspective, looking at UML for 3D development.
The OpenSceneGraph library is used by developers of 3D applications; by studying the user tasks of 3D developers in relation to UML we gain insight from users with experience in 3D User Interfaces (UI’s). Another consideration was the possibility to “bootstrap” 3D UML development through OpenSceneGraph and to leverage 3D UML benefits in our own development of 3D UML.

First we present an overview of the concept of using 3D to aid in visualising complete system views. Next in Section 6.3.2 we present the resulting user task analysis of our study with additional 3D UI results presented in Section 6.3.3. Finally we summarise 3D UML Holistic Diagrams in Section 6.3.4.

6.3.1 Overview

The concept of 3D UML Holistic Diagrams is that 3D views can enable a more complete class diagram view of a system. The purpose of this 3D diagram is to address three main issues we see with the traditional 2D diagram approach. The issues and how 3D UML Holistic Diagrams address them are described below:

a) Current 2D UML diagrams provide only “snapshot” views into a model\(^{29}\). To understand a system a software engineer has to create a mental model of the system derived from these multiple diagram views. The holistic diagram attempts to remove this burden from the engineer and present a 3D diagram that represents the whole system, the engineer can then tailor the view to suit their current engineering task.

b) Due to the restrictions imposed by 2D diagram sizes, 2D diagrams are normally manually created with the diagram author choosing the elements contained in each diagram based on the information they wish to convey. The implications of this manual process are that as the model evolves the diagrams become out of date. The holistic diagram reverses the situation by giving a view that contains all current information, with features that allow the viewer to filter information.

\(^{29}\) It is possible to create a 2D UML holistic diagram by creating one large diagram containing everything in the model, printing that diagram out and taping the sections together. This was witnessed on one RMIT industry project where the diagram created was approximately 2m x 2m. This is evidence to the need to enable a single system view within a typical computer screen to prevent the need for such workarounds.
c) Another implication of the manual diagram approach is that it is reliant on the creator of the model to take a disciplined approach in creating diagrams to allow new users of a model to build up knowledge of a model. When a new user opens a model there should be some “start” point to understanding a model and then progressively detailed views of aspects of the system. The holistic diagram presents this start point with the intent of automatically generating the view based on aspects of model usage. The aim is to present the viewer with complete system views that use 3D visualisation techniques to highlight important elements with respect to other proportionately hidden aspects of the system.

As part of our research we had gained some experience using OpenSceneGraph for implementing X3D-UML. From this experience OpenSceneGraph appeared to be a good candidate for our investigation as it has a large user base numbering over 2000 and 372 active contributors [74]. Through probing the OpenSceneGraph mailing list we found that UML is not specifically used however there are UML-like diagrams used to present class structure in OpenSceneGraph reference documentation [53].

For our study OpenSceneGraph developers presented an interesting case study due to their familiarity with 3D. The area also lends itself to heuristic evaluation due to ready access to 3D UI experts familiar with both the subject mater and 3D UI techniques. The results of our study are intended to gather information from 3D expert users rather than a UML user. We also saw the possibility that OpenSceneGraph users could eventually contribute to an OpenSceneGraph based X3D-UML implementation should the results of the study lead to 3D UML Holistic Diagrams that aided the developers in their task.

6.3.2 User Task Analysis

In this section we outline the results of our user task analysis. We first present the results of the questionnaire undertaken; we then present the current user tasks and finally the issues that need to be addressed with the current tasks.

Despite the large number of users of OpenSceneGraph we were only able to locate one Australian participant through the OpenSceneGraph mailing list. The results of our study presented in this section are for a single user session undertaken with a developer that produced 3D traffic situation simulators. This was the sole developer responsible for all
simulation software for a company specialising in driver education. Although the results are from a single user, the participant allowed us access for a whole day to undertake the study and gather detailed information.

**Questionnaire Results**

Applying our revised *3D UML Sequential Evaluation* approach, we gathered questionnaire results through discussion with the user. As the OpenSceneGraph user did not specifically use UML, the questionnaire has been undertaken with the term “UML Tool” replaced by “Text Editor + Diagram Tool Combination”. We present a summary of the questionnaire responses in Table 6-1. It should be noted that the Text Editor is the developer’s primary tool, so the answers are specifically for the text editor unless otherwise stated.

**Table 6-1 – Results of closed questions for Text Editor + Diagram Tool Combination Users.**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>User 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currently Using Tool</td>
<td>Yes – Uses Text Editors for code and “Dia” <a href="http://www.gnome.org/projects/dia/">http://www.gnome.org/projects/dia/</a> for creating software diagrams (Dia does support UML but was UML not used).</td>
</tr>
<tr>
<td>Last Used Tool</td>
<td>That day</td>
</tr>
<tr>
<td>Approx. hours Tool usage last 6 months</td>
<td>40 hours a week for 6 months</td>
</tr>
<tr>
<td>Approx. hours total Tool usage</td>
<td>17 years – fulltime</td>
</tr>
<tr>
<td>Tool Training</td>
<td>Self taught</td>
</tr>
<tr>
<td>Primary Goal when using Tool</td>
<td>Developing software for a simulator project.</td>
</tr>
<tr>
<td>High level tasks when using Tool</td>
<td>High level task is adding feature by feature (2 weeks to 4 months – average about 1 month). Tasks for a 1 month feature 1 week understanding, planning and documenting the new feature to be added 1 week “getting into the zone” of coding 2 weeks testing and debugging</td>
</tr>
<tr>
<td>High level tasks from above that use state machine diagram</td>
<td>Diagrams are used as part of the understanding, planning and documenting phase.</td>
</tr>
<tr>
<td>Tasks that use state machine diagram</td>
<td>Diagrams are used to help understand how a system is to be implemented (e.g. data flow diagram).</td>
</tr>
<tr>
<td>Job Title</td>
<td>Simulations Systems Software Engineer (Sole Developer)</td>
</tr>
<tr>
<td>Experience in that Role</td>
<td>17 years</td>
</tr>
<tr>
<td>Other UML Tools</td>
<td>No</td>
</tr>
<tr>
<td>Experience</td>
<td>UML Training</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
</tbody>
</table>
Further discussions in the area of UML and 3D experience revealed the following points:

The user did not use UML, even though the diagram tool Dia supported UML diagrams, instead the user preferred to create diagrams of his own notation. The user rated the use of diagrams as *Essential*.

The user did not favour 3D. He cited that as he developed 3D for a job he did not consider it as a fun passtime and gave evidence of this through his favourite game being a purely text based game called *Nethack* [64].

**Current User Tasks**

In the user session we looked at how current development was undertaken and how the use of diagrams and 3D fitted into the software development process.

In the area of diagramming, although the user did not know UML (only know of UML), he was very positive towards diagramming. He made use of a tool called Dia [22] and had a ‘cabinet full of papers with hand drawn diagrams’. These diagrams were created at the start of implementing a feature but not updated. He consider them a ‘thinking tool’ and, due to not working in a team, they were left “as-is” once the work was complete, unless updating it was a necessary part of the user documentation.

He also made use of diagrams generated by Doxygen [25] for areas of the code he had not visited for several months. Doxygen is a document generator tool that parses source code information and produces high level html summaries and diagrams. His development environment was set up so that Doxygen generated html documentation and diagrams directly from his code. However he did not continue to use the Doxygen diagrams or documentation once he had become familiar with an area.

Due to the spatial nature of the simulator being developed, it was also asked how much diagramming was spatial and how much software. He stated that some diagrams are spatial but most are for analysing software problems. He gave evidence of this through data flow diagrams for his current task.

In the area of the software development process, he worked on a single simulator project and incrementally added new features. Generally he tried to develop one feature a month, but depending on demand and need, features could take between 2 weeks and
4 months. For a new 1 month feature, the work breakdown was estimated to be as follows:

1 week understanding, planning and documenting the new feature to be added

The effort was estimated as 5% diagramming with the remaining effort reading/documenting and thinking. Thinking involved ‘staring into space’ and also ‘thinking a lot on paper’. The goal of this first week is to ‘get a thorough understanding of what needed to be done’. He estimated that typically only one diagram (in electronic form) was created at the start of each feature. For this he would use Dia and create block diagrams rather than UML stating ‘simply because he was not aware how to use UML’.

1 week ‘getting into the zone’ of coding

The second week would be ‘getting in the zone’ and coding the feature. He would ‘put on music and just code’.

2 weeks testing and debugging

The third/fourth weeks effort would be in testing and debugging to refine the feature.

In the area of using 3D for software development, the user had some custom made tools for “visual” debugging of the traffic simulation system and he was also able to view cars in action. His debugging implementation allowed him to select a road and observe/manipulate traffic “splines” which directed cars. However he noted that the printf statements (to a command window) were the most useful for debugging car behaviour.

**Development Issues**

The area of interest for 3D UML Holistic Diagrams was the effort in the first week, in the area of understanding, planning and documenting the new feature to be added. To investigate the tasks related to this we picked the topic of “shadows” as this was a feature the user had recently investigated.

---

30 “Shadows” is a 3D computer graphics feature that allows light shadows from objects to be included in a scene.
The users stated that the biggest issue ‘and time killer’ was the lack of documentation with OpenSceneGraph. The following is a typical process based on what he would do for “shadows”, to get around this issue:

- Look at the example projects for Shadows (OpenSceneGraph supplies many examples)
- Look at the header files for the Shadows code
- If time permits do learning bit by bit (trying concepts out)
- Read the OpenSceneGraph wiki
- Read the OpenSceneGraph user group and post if necessary

When asked what he was looking for in the source and what would help, he responded with ‘something that shows the relationships and how to use them (the classes)’.

6.3.3 3D UI Expert Experiences

One of the drivers for choosing to study OpenSceneGraph users was the possibility of additional information for 3D UML arising from the users’ experiences in developing 3D UI’s. Additional information was forthcoming and this section captures the issues raised.

The user spoke about the impact on age in immersive environments. Although users of immersive environments eventually adapt to the simulated reality, older users take longer and care must be taken to introduce an environment slowly to prevent simulation sickness. For 3D UML this has implications in the need to consider demographics of users when testing for benefit. In some situations testing for benefit on students is likely to have significantly different results than testing on experienced software engineers.

Care must also be taken in the content of an environment to prevent a large amount of “stuff” passing the viewers eyes when they need to turn within that environment. A large amount of visual movement contributes to simulation sickness by giving conflicting signals, with the user getting the impression the world is spinning as well as the intended feeling of being in a turning car. For 3D UML this has implications because software systems have a large amount of visual data. In an immersive environment an interaction
metaphor that has the user static and in control of a moving view (e.g. a microscope) may be better than a “flying” interaction metaphor\textsuperscript{31}. The microscope interaction metaphor does not attempt to simulate user movement and therefore the user is receiving only one signal.

6.3.4 Conclusions

The results of this study suggest that 3D UML Holistic Diagrams may provide benefit in aiding the initial understanding of a system. Our proposed solution was aimed at providing a view that aided a developer to quickly understand a system through analysing relationships. The user study revealed that this is indeed a task that this developer undertakes and that he has a need for ‘something that shows the relationships and how to use them (the classes)’. Based on this result and how the developer currently addresses the issue we see that a 3D UML Holistic Diagram that leverages code examples to highlight important classes and relationships would provide the most benefit. The solution would initially follow the developer’s train of thought by displaying a diagram that firstly highlights classes based on a given example, and then secondly highlights classes which are directly linked to those classes and then faded links into the code base as a whole.

An issue preventing further investigation in this area however is the lack of available participants. A study based on a single user would not produce results of any significance. Also creating a prototype proved difficult, when we attempted to generate a UML model from the OpenSceneGraph code base we found that the effort required was large due to the extensive use of pre-processor macros. Our user study does show however that the problem of learning something “new” or “unfamiliar” in a computer software system seems to be the same for an experienced engineer as for a student learning a new topic. We suggest that a new study involving students learning a new API may provide better research results into the use of 3D UML Holistic Diagrams. Using students gives us the advantage in that participants are more accessible, the “User Task Analysis” exists in the form of learning material and “Summative Evaluation” already exists in the form of examinations.

\textsuperscript{31} We had witnessed this problem ourselves, in an early RMIT presentation of X3D-UML. We had presented a large class diagram and we flew through the diagram displayed. The presentation was not in an immersive
Basing our study on OpenSceneGraph developers did reveal important outcomes. For example the results of a student study, suggested above, would also need to be validated on older users due to differences in ability to adapt to 3D environments. Another outcome was an appreciation of the impact of particular types of interaction metaphors on the usability of a 3D view.

6.4 Summary

The results of our investigations into 3D UML Mechatronic and Holistic Diagrams provide supporting evidence as to the benefit of 3D UML documented in previous chapters.

In Chapter 5 we investigated 3D UML State Machine Diagrams and our user task analysis documented issues with engineers comprehending hierarchal state machines. We saw that due to limitations in the visual information the engineer was required to take steps to create their own mental model of how aspects of the system worked together. In our 3D UML Mechatronic and Holistic Diagrams we see different domains which display similar issues.

In Lego NXT development the software engineer is required to think about the complex interaction between software and hardware domains. Although this activity is core to developing a mechatronic system, it is again left up to the engineer to take steps to create their own mental model of how these aspects of the system work together. Our 3D UML Mechatronic Diagrams demonstrate that this burden can be reduced by integrating the software and hardware in a single visualisation so that the developer can directly view the complex interactions. The benefit in this view is that it aids the developer in understanding the impact of hardware variability and validating that internal programmed behaviour matches the observed behaviour.

For OpenSceneGraph development we also see the engineer spending dedicated effort in creating a mental model of the relationships between aspects of the system. This is an area that can be addressed with a structured implementation of 3D UML Holistic Diagrams, presenting the user with the relationships they are currently piecing together manually.

---

game environment but on meeting room LCD projector. One of the attendees viewing the scene stated that it made them “feel sick”.

173
In considering the benefit from the perspective of 3D modelling of UML we have demonstrated that our 3D representation of UML can be reused in different contexts due to the underlying 3D geometric description. We support the argument that each aspect of benefit we have presented could in some way be addressed by a combination of “2D” techniques; however by using 3D techniques we provide more possibility of reuse. For example if **3D UML State Machine Diagrams** were implemented as “pseudo-3D” layered planes designed specifically to the problem of representing hierarchies they could not be reused as 3D objects in **3D UML Mechatronic Diagrams**. Our approach lets us model the software in 3D and then use the view to provide the interaction techniques required for each task.

Due to the low participation rate in this area of research, further research is required to reapply these concepts to similar areas with larger user bases. This is required before the concepts can be fully refined through **3D UML Sequential Evaluation** and before benefit can be measured.
Chapter 7 – Conclusions, Validity and Future Research

Graphics Processing Unit Analysing Human Brain Processing.
Images from “GPU-based Multi-Volume Rendering for the Visualization of Functional Brain Images” [80]. © 2006. SCS-Publishing House e.V.

7.1 Overview

In the introduction to this thesis we highlighted that soon CPU processing power, on the average desktop pc, would arguably overtake human brain processing power. It is interesting to note that in the time span of this thesis GPU (graphics processing unit) power has overtaken CPU processing power in areas of parallel computing and this is now being used to assist the CPU in non-graphics tasks. It is also interesting to note that the Eclipse Project has recently adopted the GEF3D extension as an incubation project. The continued rapid increase in 3D graphics capability holds much promise for further exploration into effectively leveraging 3D for software engineering tasks. With GEF3D enabling 3D UML editors to be created within Eclipse we are likely to see implementations which leverage 3D for UML. These developments have meant that our

---

32 Permission to use image for this thesis was kindly granted by the author Friedemann Roessler and Rainer Rimane, SCS-Publishing House e.V.
research outcomes have become more relevant to evaluating and ensuring 3D UML extensions are designed to be useable in industry and address actual user issues.

3D user interfaces are often referred to as a ‘solution looking for a problem’. By applying user-centred evaluation we have approached the research question with a problem-centric focus and first seek to understand and document the problems engineers face. Our outcomes provide detailed information about these problems which lead to targeted research and solutions. The value in this extends beyond the area of researching 3D as different approaches can be applied to address the same issue. However by approaching the problem from a 3D perspective it has allowed us to use 3D as a thinking tool, to allow us to explore solutions unrestricted by traditional user interfaces.

In our research we have demonstrated that by using a 3D geometric model we can visualise UML for different tasks. For example the state machine diagrams can be used as an aid to understanding complex hierarchies or used in augmented reality. From a modelling perspective there is a strong synergy between UML and 3D. For both we have the need to present the viewer with an intuitive representation of an underlying concept. We use the power of computer data processing to first model the concepts because we can formalise our ideas in a way that a computer can process and manipulate. For UML the representation the user sees is a view into an aspect of a complex software system model, with the aim to help the user understand a part of a system. For 3D, the representation the user sees is a 2D projection derived from a multi-dimensional 3D geometric model, with the aim to facilitate presenting the user with a realistic scene. From this point of view, 3D is a natural way of modelling and visualising multi-dimensional UML relationships with the intent of creating a visualisation on a 2D screen.

The area of 3D computer graphics is complex and this complexity has contributed to the lack of research and development of usable UML visualisations. Our X3D-UML implementations have reduced this complexity and facilitated the modelling of 3D UML. Through X3D-UML we have enabled the design, implementation and evaluation of 3D UML.

In this chapter we present in Section 7.2 the conclusions drawn from our research as a whole. In Section 7.3 we present the threats to validity and finally in Section 7.3 we discuss future research opportunities.
7.2 Conclusions

Although the research has not definitively answered the question ‘is there benefit in 3D UML visualisation?’ The research outcomes have changed the question. The argument is no longer “to 3D or not 3D” based purely on 3D being a single concept. We have shown that the area is much more complex. The research shows that UML does map to 3D and users find this mapping natural. The research shows that although the mapping is natural there are a number of issues to address before undertaking Summative Evaluation. The argument against 3D is now based on issues around lack of specialised user interaction support for working with UML diagrams. Far from being unsolvable problems, the results of our research have documented the specialised user interaction support required.

Previously in Section 4.8 for Summative Evaluation we expressed that to definitively answer the “million dollar question” of benefit in 3D UML the ideal outcome would be a controlled experiment producing a performance graph, like the one in Figure 7-1, measuring 2D performance against 3D performance. Although the graph still contains a question mark, our research has contributed significantly towards quantifying benefit in this context. In the next sections we review each area researched, discuss them in this context and explain how we have addressed various aspects of the graph. The importance of the user tasks investigated represents the area under the graph and the 2D and 3D complexity trends represent the difference between the approaches.

![Figure 7-1 - Summative Evaluation of 2D Vs 3D](image)
7.2.1 3D UML Sequential Evaluation

The first aim of our research has been to refine a methodology and to apply the methodology to test for benefit in the use of 3D for UML. The research outcomes reflect the success of the refined methodology, with detailed research results in areas with documented user need. Professional users willing to participate in research are a rare research resource and therefore it is critical to make efficient use of this resource when they are available.

In User Task Analysis, the refined methodology allows us to evaluate current user tasks more efficiently. This efficiency is first through structuring the task analysis to get the best results and the clues to look for when studying UML usage. From a small user base, the methodology extends the results through model metric surveys which provide quantitative evidence supporting user issues. These results are extended through the addition of user task questions to expert evaluators. In this phase we quantify the importance of user tasks and the complexity issues with existing “2D” practices.

In Heuristic Evaluation, the refined methodology presents a reviewed and highly structured approach to obtaining an effective evaluation of a 3D UML implementation from experts. This refinement improves the ability to capture common user issues prior to user involvement and therefore further reduces the wasting of professional participants’ time.

In Formative Evaluation, the refined methodology guides against some pitfalls in developing 3D UML implementations and testing in the users’ environment. This refinement allows the researcher to clarify user need first through absence of a feature, rather than labour intensive implementation of features that may not be necessary or worse, detrimental.

Summative Evaluation is the most user intensive phase, the refined methodology allows us to enter this phase with all due diligence complete. The previous phases reduce the complexity of “3D” to the user and testing can be done with confidence that the resulting outcome will accurately reflect the performance measures of 3D UML.
7.2.2 3D UML State Machine Diagrams

Using the refined methodology, the concept of using the third dimension to aid the software engineers navigating substate and superstate layers in hierarchical state machine diagrams was tested. This area represents the main focus of our research.

The importance of the user task is quantified. The analysis highlights a common and critical software engineering task that benefits from an alternative UML visualisation provided by 3D. For RoseRT engineers, defining state machine behaviour is a task which occupies 30%-40% of their effort and our model metrics survey shows that between 56%-90% of that effort is likely to be related to hierarchical state machines.

The trend of complexity of the existing “2D” practices is also quantified. We quantify the issue through a metric of “off-screen” information missing from the engineers' view which indicates the level of “out of diagram thinking” currently required. Both user and expert responses show that this is an area of known concern and our analysis captures the extent.

Evidence is given of the reduced complexity of our “3D” approach. The benefit of 3D UML state machine diagrams is in the area of understanding hierarchical state machine diagrams and intuitively presenting the user with additional information needed to complete their tasks. The results of our research present consistent evidence that there is benefit in 3D UML state machine diagrams. Both user and expert evaluation sessions provide qualitative evidence that benefit already exists in our 3D UML state machine diagram, however further refinements have been suggested to capitalise on 3D visualisation possibilities. Our research highlights that specialised interaction techniques are required for 3D UML to reduce complexity, such as semantic zoom, restricted navigation and filtering.

7.2.3 3D UML Mechatronic Diagrams

The concept of a 3D view of a system which presents both the software and mechanical runtime attributes was tested. A fully comprehensive study could not be undertaken in this area due to the lack of availability of users for this study. User task analysis and anecdotal evidence through implementation shows potential benefit in the integrated mechanical and software visual models of a system. We tested the concept of UML
diagrams in a 3D space, integrated with a Lego NXT robot, where the view of the software and the robot are combined into a single view. The importance of the user task is quantified for RoseRT users with *Test and Debugging* being listed as 1 of 4 main tasks.

In the task of reducing defects, we found “3D” visual debugging had advantages over the traditional “2D” debugging approach. For example the robot was designed to go back and forth against the wall and from observing the robot in the non-augmented view; the developer had met the specification quickly. However the “3D” visual debugging showed problems with things outside the specification, for example the “buttonReleased” state was not triggered when the robot reversed. Also variability in “hardware execution” was more apparent, for example difference in wheel rotations and the touch sensor angular performance. Even for such a simple model we found the 3D helped make a more robust system compared to the traditional approach.

Another important aspect of this investigation is the effective demonstration of the significance of using 3D computer graphics with UML. As our 3D UML is defined as a complete 3D geometric description we are able to reuse that description in a new context. In this instance we are able to use the same 3D UML used previously for understanding complex state hierarchies in combination with ARToolkit to facilitate a new task of debugging. If our implementation has been pseudo-3D we would not have been able reuse the UML in this way.

### 7.2.4 3D UML Holistic Diagrams

The concept that 3D views can provide a more complete view of a system was investigated. Current 2D UML diagrams only provides snapshot views into a model, which means the software engineer is presented with multiple diagram views into a system and it is left up to engineer to form a more complete model view in their heads. The holistic diagram attempts to remove this burden from the engineer and present a 3D diagram with which the engineer can tailor to suit the engineering task. Due to the approach taken and the lack of available users with the required expertise, the results are limited to reporting user tasks and 3D expert feedback.
In the area of understanding a system the user task is quantified. The engineer is required to dedicate approximately 25% of their time to familiarising themselves with a particular area of a system. In the area of 2D complexity, the user study revealed that they have a need for ‘something that shows the relationships and how to use them (the classes)’, with the users process suggesting a possible automated approach.

The other aspect of this area of investigation was leveraging 3D expertise. The results provide valuable information to 3D UML implementers in reducing 3D complexity and in measuring the results, especially in the area of fully immersive virtual reality environments. These results highlight that consideration is needed for the users’ ability to adapt to 3D views, and this needs to be consider in both the implementation and measurement. The most significant outcome for our research is in the area of a participant’s age impacting on their ability to quickly adapt to 3D environments. This further supports our conclusion in 3D UML State Machine Diagrams that for valid empirical measurement of benefit, a long-term study is required with professionals, rather than the more typical short student based experiments. Using students would likely skew results in favour of 3D due to their ability to quickly adapt. There is an opportunity however to leverage student based experiments in the area of learning complex systems for educational purposes.

### 7.3 Threats to Validity

In this section we discuss the issues that impact the validity of our research. We first look at the extent that our X3D-UML design constraints from Table 2-1 have produced valid results. We then look at a number of areas that have influence on the validity of our results. The issues discussed are in UML contributing to 2D complexity, bias caused by 3D perceptions, demographic influences and questioning validity from a higher level.

#### 7.3.1 X3D-UML Design Constraints

In Section 2.3.4 we discussed how X3D-UML was to be applied to researching 3D UML and placed this research in the context of design constraints given in Table 2-1. The purpose of these constraints is to focus the research results so they are broadly applicable to industry and have external validity. We now revisit those constraints and
discuss how they have contributed to the validity of our research and suggest areas where they could be extended.

a) Actual Data
   a. Validity
      i. 3D UML visualisations have all been based on actual data from industry and therefore represent non-toy data sets.
      ii. Model metrics have shown that our visualisations are valid for the size of data found in industry, for example we have demonstrated usable visualisation of the largest 3D UML State Machine Diagrams typically found in industry.
      iii. The problems the visualisations address have been demonstrated to exist in industry and the extent of their existence is quantified through model metrics.
   b. Extending Validity
      i. In the area of state machine diagrams, data has been based on RoseRT models. Surveys of model metrics from other tools, such as Rhapsody [45] are needed to further validate our approach. Although we may assume some validity with tools that use the same approach of layering of diagrams, other influences such as the relationship between code and diagram; or useability aspects may lead to different use of diagrams. The extreme of this would be most obvious in UML tools that do not have a tight coupling between diagram and code; such diagrams may avoid complexity by the engineer creating incomplete diagrams\(^{33}\).

b) Actual Tasks
   a. Validity
      i. The majority of the user task analysis was undertaken with experienced professional engineers.

---

\(^{33}\) As an example, the author has had industry experience with validating a state machine diagram given in a system specification document. Through validating a state machine diagram using a modelling tool it was found to be incorrect due to the specification author over simplifying the diagram. The final product however had met the specification author’s unstated intent due to developer relying more on local knowledge of how products should work rather than following the diagram as a strict specification.
ii. The user task analysis quantifies the extent that the user tasks are applicable to industry. The focus of this research has been on high impact tasks which occupy 25% or more of an engineer’s effort.

iii. All aspects of our 3D UML implementation are solutions in response to problems captured through user task analysis (i.e. non-toy tasks).

iv. As further evidence to the validity of the state machine diagram issues we have described, a recent independent study has also highlighted problems in comprehension of layered state machine diagrams [39]

b. Extending Validity
i. Empirical experiments are desired to extend the validity of the qualitative results.

c) UML
a. Validity
i. Our 3D UML visualisations have not required the engineer to change existing software visualisation practices.

ii. Our results have shown that visualisation of UML in 3D has been challenging due to the lack of support for text and lines in 3D engines, however we have demonstrated implementations which successfully visualise UML.

b. Extending Validity
i. Although UML is a standard it has many different implementations. Differences in how the UML is used will impact how successfully 3D UML can be applied.

d) X3D
a. Validity
i. We have demonstrated that X3D can be applied to 3D UML visualisation and also presented the limitations of X3D when applied this way.

b. Extending Validity
i. Although X3D is a standard it has many different implementations. Differences in how X3D is implemented, such as browser performance and user interaction techniques, will impact on the usability of the final 3D UML.

e) Standard PC Hardware
a. Validity
   i. Visualisations presented in this thesis are not restricted to specialised graphics machines.
   ii. Formative Evaluation was undertaken directly on the development machines that engineers’ were using for their current development.
   iii. Our research has been demonstrated to be applicable to Windows XP and Windows Vista machines.

b. Extending Validity
   i. Cross platform issues will impact the ease of extending the research to different operating systems. With X3D there will be issues with browser performance on non-Windows platforms due to lack of X3D maturity on these OS’s. With our OpenSceneGraph implementation there will be no performance impact however effort is required to integrate the X3D-UML viewer into an appropriate window system.

f) Development Environment Imitation
   a. Validity
      i. The visualisations presented have imitated the developers’ environment as closely as possible. All visual queues, such as colour, shape and size, have derived from the existing environment.
   b. Extending Validity
      i. The development environment represents an implementation of UML. Differences in how the UML is implemented will impact how successfully 3D UML can be applied.
      ii. Evolving development environments will reduce validity over time due to 3D UML integrations becoming obsolete and due to changing work practices.

7.3.2 Poor UML and Tool Implementations

There are recognised issues with existing UML notation and implementations contributing to the complexity of existing user tasks. At the 11th International Conference on Model Driven Engineering Languages and Systems, one comment from the panel discussion on the challenges of Model Driven Engineering, from the CEO of the Object Management Group, Richard M. Soley, was that the ‘UML is the worst modelling language except for all the others.’ This captures the fact that the UML as a notation is
accepted more because there is nothing else better than due to the excellence of the notation. Another concern raised in the panel was the poor implementation of UML tools. In the user task studies undertaken as part of this thesis, there were consistent complaints about the tool usability.

Although the results in this thesis show advantages in the use of 3D compared to 2D, the comparison is against a poorly implemented 2D notation. Improving the implementation of the 2D notations was not the topic of our research; however we would have undoubtedly also experienced improvements through a 2D user-centred redesign of the original UML implementation (i.e. New 2D Vs Old 2D). Although our research focuses on 3D advantages, we present some future research suggestions for 2D improvements arising from our findings in Section 7.4.1.

### 7.3.3 3D Bias

There are recognised issues with 3D bias which have impacted on the results of this study in both negative and positive ways. The following headings cover the positive (pro-3D) and negative (anti-3D) bias aspects to be considered.

In *User Task Analysis* participants who are pro-3D are more likely to participate in a study which investigates the use of 3D to help them in their everyday tasks. In the initial stages of the study though we are not testing for 3D benefit but instead analysing current tasks. The issues captured in *User Task Analysis* have been validated with model metrics and with feedback from expert evaluators, so we are confident that the results of this stage are not skewed by this pro-3D effect.

In *Heuristic Evaluation* expert participants who are pro-3D are more likely to participate even though the goal of *Heuristic Evaluation* is to identify usability issues (i.e. negative feedback). This was witnessed in the Heuristic Evaluation workshop and in Heuristic Evaluation responses. Pro-3D participation will likely skew results with positive qualitative responses in favour of the 3D visualisation. In this phase however positive responses are used as only anecdotal evidence and the focus is on reporting and addressing the negative issues.

In *Formative Evaluation* we have the impact of the pro-3D participation. Of the 5 professionals who directly took part, 2 claimed to be “gamers” (i.e. played computer
games). Although a gamer may be pro-3D, their expectations of a usable interface may be higher than a non-gamer. We also had participants with experience with 3D development, in our case 2 of the 5; these participants are more likely to have lower expectations of a usable interface due to an understanding of the implementation difficulties.

In Summative Evaluation we have indicated a number of areas to address validity in Section 4.8. For this phase we not only need to reduce the impact of pro-3D bias but also the influence of the researcher.

7.3.4 Demographics

In a user task analysis session undertaken with a developer of a 3D virtual reality application late in our study we highlighted the issue of demographics on the usability of 3D. The developer relayed detailed stories of older users being physically less adaptable to immersive environments. We have not explicitly recorded demographic information, however from personal contact and through the industry experience stated, our participants’ ages can be assumed to be ~35-45 years old. As our study consists of users of a similar demographic our study can be considered valid for that age range. The model metrics gathered in our study are from unknown age groups, however the metrics are used to quantify the 2D problem rather than predict 3D benefit.

However we consider demographics is an issue to external validity in the area of 3D graphics. Age does not just have an impact on a person’s physical ability to adapt to a new concept, it also an indication of the experiences a participant has had through life and their familiarity with 3D concepts. For example in Australia the gaming demographic changes over time consistent with an aging “gamer” generation. In 2009 the average game player age is 30 years and the average non-game player age is 40 years, the trend predicts that by 2014 the average game player age will be 40 years [15].

The results of our research may therefore be less valid for mature engineers; however the impact of the age variable may reduce over time due to the pervasiveness of 3D graphics. We would expect that our research results may be more valid for younger engineers and students, due to the increased exposure and familiarity with 3D interaction.
7.3.5 Higher Level Issues

The higher level ancillary impacts of the visualisation have not been considered in this research. We have not considered if improving the visualisation approach reduces performance in other areas.

As an analogy, pocket calculators are a powerful tool that allows multiplication quickly and intuitively, however despite the existence of calculators we teach our children to do multiplication in their heads. Although the skill is hard to learn, once it becomes “second nature”, this skill greatly aids in application of mathematics in new ways. In a similar way if we are forced to learn how to construct a complete model of software in our heads, then this may be more efficient than any visualisation tool. As evidence to this, the quantity of users’ sketches of diagrams were found to diminish over time, which suggests that they first struggled with the diagram and tool concepts, however over time learnt to create their own mental sketches. The same users claimed to have fewer problems with the understanding of diagrams, which might suggest that the learnt mental sketch compensated for lacking information in the diagram.

The impact of this and the positive or negative aspects are unknown. For example it may be possible the effort to gain a mental model, although difficult, may be of greater benefit overall than providing a visualisation which reduces this effort. Alternatively, the reduced effort of the visualisation in 3D may accelerate the ability for the user to create an effective mental model.

7.4 Future Research

In this section we discuss future directions for research. We first look at leveraging this research for enhancing existing UML interfaces in the area of 2D and 2.5D (layered planes). We then discuss opportunities for extending this research in the area of 3D UML.

7.4.1 2D and 2.5D

Opportunities exist for applying the results of our research to existing UML tools in the area of pseudo-3D techniques. We suggest a number of concepts that could be researched without major modification to existing tools and that may provide measurable
benefit. The advantage in these areas is that the results are likely to be adopted by the tools because they are more easily integrated into a 2D interface.

a) **Pseudo-3D Diagram Navigation** – Navigation in large UML diagrams typically uses standard window interaction mechanisms, being scroll bars and a menu driven zoom factor. In addition to this, some tools provide a separate overview “map” which gives an indication of where in the diagram the current view is looking. Pseudo-3D navigation would implement 3D like zooming. A good example of this is Google maps where the mouse scroll wheel is used to zoom in and out centred around where the mouse is pointing. The idea is to maintain a mental map without distracting the user with zoom selections or referring to another source for location information.

b) **2D + 2.5D Diagram Layering** – In the area of mapping relationships between diagrams, 2D and 2.5D techniques could be used to combine diagrams in the same view and this may provide measurable benefit. Although they may lack the underlying 3D geometric description to allow varied use of the information, they may address specific problems and have more immediate application in existing tools.

c) **3D UML Sequential Evaluation** – The area of applying our refined methodology to any UML concept with the aim of improving the design through 2D or 2.5D (or 3D) should provide valuable results. For example applying our suggested *Heuristic Evaluation* to all UML diagrams should highlight a number of usability issues that can be addressed as research topics. Applying *User Task Analysis* would also provide the basis for ongoing research in a number of areas.

### 7.4.2 3D UML

Despite the pseudo-3D possibilities, our argument in 2.5D versus 3D is that our 3D is a geometric model and this model enables the information to be reused in different circumstances. Although in our research the visualisations look like 2.5D, if we had based them on 2D technology we would have not be able to easily extend research in other areas such as augmented reality. Our research is just the start of 3D UML and pure 3D notations may evolve (such as geon like extensions). The important factor in our
research is that due to the underlying 3D model we are not limiting the research possibilities\textsuperscript{34}. We summarise future research opportunities below:

a) 3D UML State Machine Diagram + Summative Evaluation – As we have highlighted there is further research required to measure benefit. This is continuing research which will take many years to cultivate the conditions for a valid experiment. Our approach is to integrate our 3D UML State Machine Diagrams into an existing tool and measure benefit within the context of everyday use. The issues related to this have been described in Section 5.6.

b) 3D UML Mechatronic Diagrams – We see many opportunities in the area of combined mechanical and software 3D views. We have presented debugging through augmented reality and we see this extending to field servicing of products (e.g. service personnel being able to view internal state). In the design phase, UML could be integrated with CAD models. These concepts could enable a UML design artefacts to add value throughout the entire life cycle of a system.

c) 3D UML Holistic Diagrams – We see a strong opportunity in focusing research on the educational aspect of students learning systems enhanced by a 3D UML view. Professional engineers are a rare resource; however students learning software are more accessible. In addition to access to participants there is also ready access to information about user tasks, the issues students have with learning, experts in education for heuristic evaluation and, most importantly, the measurement criteria for Summative Evaluation.

### 7.5 Closing Remarks

In this thesis we have chosen to break with tradition and instead of presenting a thought provoking quote as an introduction to each a chapter, we have instead presented a thought provoking image. The images have started from the first graphics produced by humans which predate writing itself, to the advanced use of 3D graphics to study our very own thought processes. In closing though we present a quote by Harel when

\textsuperscript{34} As example of the flexibility that a 3D model introduces, the author has sourced a memento of this PhD thesis in the form of a 3D UML diagram laser etched in a crystal. The manufacturer is able to produce a 3D artefact directly from our UML representation.
reflecting on the lessons learned from developing statecharts. This quote also reflects what we have aimed to achieve with our research methodology:

‘This is something I would not hesitate to recommend to young researchers; in order to affect the real world, one must go there and roll up one’s sleeves. One secret is to try to get a handle on the thought processes of the engineers doing the real work and who will ultimately use these ideas and tools… If what you come up with does not jibe with how they think, they will not use it. It’s that simple. [38]’
Appendix A – UML User Questionnaire

As discussed in the plain language statement, we are conducting research into 3D visualisations with UML. We are gathering information about UML usage and experience and would like you to complete this questionnaire. Please do NOT include any details that will identify you or your company.

The questionnaire is related to the UML Tool: **IBM Rational Rose RealTime (RoseRT)** and **State Machine Diagrams**

**UML Tool Experience:**

In this section we are getting an understanding of the level of exposure you have had to RoseRT.

1. Are you currently using RoseRT in your everyday work?

2. When was the last time you used RoseRT to achieve a work related goal?
   (E.g. yesterday, 2 years ago)

3. Approximately how many hours in the last 6 months have you spent using RoseRT?
   (E.g. 40 hours, ~20 hours a week)

4. Approximately how many hours in total have you spent using RoseRT?
   (E.g. 400 hours, ~20 hours a week for the last 2 years)

5. How did you learn to use RoseRT?
   - IBM Rational Training Course
   - In-house Training Course
   - Self Taught
   - Other:

**UML Tool Usage:**
In this section we are getting an understanding of what motivates you to start-up RoseRT in the first instance and then what motivates you to navigate and use a State Machine Diagram.

If you are not currently using RoseRT, answer these questions based on your role at the time you did use RoseRT.

If you use RoseRT for a number of different projects, please answer these questions based on the most significant RoseRT project you are working on.

6. Within your role what is your primary goal when using RoseRT?  
(E.g. Updating an existing system with new features, creating a completely new system)

7. Please list important high level tasks undertaken with RoseRT to achieve the above goal?  
(E.g. creating a display driver, updating input behaviour, debug application)

8. Within RoseRT what high level tasks above cause you to open and use a State Machine Diagrams?  
(E.g. creating the display driver behaviour, understanding connected capsule behaviour)

9. Please list important high level tasks that you complete within a State Machine Diagram?  
(E.g. Adding states and transitions, watching behaviour during debugging, review existing state machine)

**User Experience:**

In this section we are getting an understanding of what general industry experience you have had.

10. Under what job title do you work with RoseRT?  
(E.g. Student, Systems Engineer)

11. How many years experience have you had with this type of role?
(E.g. 6 months project work, 20 years Systems Engineering experience)

12. Have you had commercial experience with other UML tools and what were they?
(E.g. Rhapsody, Rational Software Modeller)

13. What training have you had in UML in general?
(E.g. Training course, University, Books)

14. Please rate from 1 to 5 how you feel UML helps you in your role (1 = Not at all, 5 = Essential)

1 - Not at all helpful 2 - Helpful in limited areas 3 - Moderately helpful 4 - Very helpful 5 - Is essential

Comments:

3D Experience:

In this section we are getting an understanding of your exposure to 3D navigation

15. Please list any 3D software applications that you have had experience with in the past 2 years and approximately how many hours total spent using these?
(E.g. Computer Aided Design – 13 hours, Computers Games – 2 hours a night for 4 months)

16. For each application, what aspects of 3D navigation did you find useful in completing your tasks?

17. For each application, what aspects of 3D navigation did you find hindered you from completing your tasks?

18. For each application, was your 3D navigation experiences positive, negative or neutral.

Comments:
Appendix B - Heuristic Evaluation Sheet

Thank you for agreeing to participate in this research in testing for benefit in extending UML with 3D visualisations. This document presents the information on providing Heuristic Evaluation of a 3D UML visualization. It presents the research background, the user task targeted, the 3D solution and evaluation form.

**Background**

You have been asked to participate because of your expertise in UML or expertise in the particular tool and tasks being utilised in the research (or both). Heuristic Evaluation is part of the Sequential Evaluation methodology (pictured below) and is “rule of thumb” evaluation by expert evaluators to provide a first round review of a proposed visualisation for undertaking particular tasks. The purpose of the evaluation is to critique a visualisation using guidelines (where possible) that are either documented or from personal experience.

As the research being undertaken is 3D UML visualisation, it is understood that there are no guidelines currently for such visualisations. Part of the research is to uncover possible guidelines that could be applied successfully to this and other similar visualisations.

The foundation of Sequential Evaluation (and the research has a whole) is that benefit is measured against common real-world tasks. The first phase analyses how users use UML currently in their everyday tasks, in this instance State Machine Diagrams, we then test for benefit against those tasks.

User Tasks have been documented from a small team. As experts you may also comment on how broadly applicable those tasks are to the wider UML/product community.
**User Tasks**

From studying a small team of RoseRT users it was determined that their individual high level tasks were similar and users iterated through the tasks in a similar way. There was also similar overlay in the tasks, where users could not clearly define where one task type ended and another started (i.e. tasks merged into each other and aspects of one task complimented another)

The goal of all users was the creation of a different subsystem as part of a complete product. Each user had their own package within the model, which integrated to other subsystem packages.

The high level tasks defined to achieve the goal above with RoseRT were as follows:

- Analysis + Design (pen + paper designs and “in the head” thinking)
- Creating Structure and Behaviour (translating design into an implementation)
- Refactoring (refining implementations, fixing bugs and adding new features)
- Test + Debugging (testing implementation)

Investigating each task further, it was found that, since the system behaviour was defined through state machines, users were required to refactor state machine diagrams a large amount of the time (30%-40% of their week), this was either as a result of the creation process, test + debugging outcomes or changes in project requirements.

The team made extensive use of hierarchal state machines (as was also found in a survey of 4 other models from other sources), with one user stating they had state machines with 8 levels. Through discussions about how they went about refactoring such state machines, it was uncovered that the users (from the researcher’s perspective) were compensating for a limitation in the UML representation of such diagrams.

On one hand the users needed the abstraction of individual substate diagrams; however they also needed to understand the complete diagram especially from the perspective of event processing. One user claimed to be often “staring into space for 15 minutes” trying to determine the consequences of deferring one event (later discussion found he was constructing the complete state machine in his head). Another used “find” functionality to find all items in the model with an event name so they could check them off individually.

From this analysis we determined that refactoring of hierarchal state machines was a critical and common task being undertaken by RoseRT engineers (and possibly other engineers that use state machines for code generation). It was also determined that this task was missing additional information in relation to what events are processed and where. This analysis indicated that the initial idea of 3D state machine diagrams had merit; however it needed to be extended further to incorporate missing information required for the task.
3D UML State Machine Diagrams and Event Notation

To solve the limitations in the UML for the task of refactoring state machine diagrams, it is proposed that:

- 3D state machine diagrams can augment the current diagram mechanism to enable a view of the complete state machine diagram. 3D diagrams do not replace 2D diagrams; only present them in a different way.
- Additional event notation is needed to explicitly define how each level of a state machine diagram hierarchy handles events. The notation is not currently in UML but can be expressed in both 3D and 2D forms.

Examples of these are presented in more detail below

3D State Machine Diagram

Attached with this document (with instructions of usage) are interactive examples of 3D state machine diagrams. These have been derived directly from existing model data and have not been altered (i.e. they have not been manipulated to better suit 3D). Below is a screenshot example. For 3D state machine diagrams, each substate diagram is presented in the corresponding layer and at position associated with its superstate (the exact positioning algorithm which suits most state diagrams is yet to be determined).
Event Notation for Hierarchical State Machine Diagrams

To support the cognition of how events drive an entire hierarchical state machine, the following notations are proposed. Since events that are not processed (i.e. trigger transitions) in the current state diagram level are propagated upwards, abstract notation is needed to aid the reader to analyse events without reviewing every diagram and every transition. The notation only associates events with diagrams and does not go so far as to indicate which transitions actually handle the event in the associated diagram.

For 3D state machine diagrams, due to all states being visible, the notation left (and in 3D in the previous page) indicates that an event is processed in the associated diagram. Events that are processed at different levels are linked so that the implications of propagation up the hierarchy can be easily analysed. Events that are “let through”, that have no impact at higher levels, can also be easily ignored from analysis (due to having no impact).
For 2D state machine diagrams, the problem still exists for letting the reader know an overview of where an event is processed. This forces the reader to search up and down the hierarchy for information on triggers of transitions.

To solve the issue we propose new 2D UML notation derived from our 3D notation. For the associated diagram, the notation should be able to indicate:

- An event is only handled in this diagram
- An event is handled in this diagram and above
- An event is handled in this diagram and below
- An event is not to be handled in this diagram (optional and used as a design aid/note)

Possible notations for each are pictured right. These are envisaged as being connected to the diagram edge top left, to visually represent events entering.

The notation can easily be generated from model data as it simply lists unique trigger events from the diagram transition information. Only one instance of each event is needed (i.e. the diagram may contain a hundred transitions triggered by “X” but only one event is show).

**Evaluation Section**

Based on what you have read above we now ask you to evaluate the 3D state machine diagram examples given against the task of refactoring existing state machine diagrams. Imagine that you are using the diagram prior to making a change or reviewing a change post editing a 2D diagram. Also pay particular attention to how you now evaluate how events are propagated (since previous diagrams did not support this).

Please utilise and note any generic guidelines that you apply in your evaluation.

Note: The diagrams provided have the following limitations
- It is not expected that editing of the diagram been done in 3D, it is only a view
- History points are missing the “H” and connection lines to the perimeter
- Choice points are missing “C” notation
- Some spurious adornments exist (i.e. the odd circle which is normally invisible)
- “Hover” to see inside code has not been implemented

As we are all conscious of time, please fill out the following as best you can within your time constraints and note the time taken. Answer important questions first and others as time permits:
User Task Questions

Important:

1. If you have experience with RoseRT users, state approximately how many users and if the tasks of those users for refactoring state machine diagrams are similar to those described above.

2. If the tasks are similar, what percentage of an average user’s week (assuming fulltime RoseRT) do you estimate to be refactoring state machine diagrams?

3. If you have experience with other UML tools and users that make use of state machine diagrams (which are used to generate code), how does the task of refactoring and the use of hierarchal state machines compare?

As time permits:

- Comment on experiences with RoseRT users comprehending hierarchal state machine diagrams
- Comment on how other tools handle such diagrams
- Comment on whether “hand coded” state machine diagrams also experience the same problems (i.e. do they get as complex?)

3D State Machine Diagrams Questions

Important:

4. List any issues that you think would prevent the 3D state machine diagram being useable.

5. List any issues that are a result of failed guidelines you have applied (noting those guidelines).

As time permits:

- Comment on if the 3D diagram enhances your comprehension of the state machine
- Comment on enhancements that could be made to the diagram
- Comment on anything which frustrated you with the diagram
Event Notation Questions

Important:

6. The event notation proposed has been derived solely from the 3D research and not through general state machine literature review. Is the issue of the event notation limitation familiar to you through other sources?

7. Do you think that some form of event notation, incorporated in the state machine diagram would be of benefit?

As time permits:

- Comment on other 2D notation experience which may achieve the same result of comprehending event impact on state machines (tables for example)

General Comments

Important:

8. Please provide a brief summary of your overall opinion of 3D state machine diagrams for the refactoring task.

As time permits:

- Add any other comments you wish here.
Bibliography


[58] T. Lindner, ""Production Cell" Case Study", 


[66] NVIDIA Corporation, "NVIDIA Scene Graph Software Development Kit (NVSGSDK) ": NVIDIA Corporation, 2008.


[73] OpenSceneGraph, "OpenSceneGraph Website", http://www.openscenegraph.org/projects osg, 2009


[75] osgswig, "SWIG based bindings for OpenSceneGraph Website", http://code.google.com/p/osgswig/.


[89] University of Paderborn, "Railcab Visualisation in Augmented Reality", http://www-nbp.uni-paderborn.de/index.php?id=34&L=1,

[90] University of Paderborn’s Fujaba, "Fujaba Tool Suite Project Website", http://wwwcs.uni-paderborn.de/cs/fujaba/, 2009


