

Technical Report

X3D-UML: 3D UML State Machine Diagrams Evaluation Methodology

Paul McIntosh
School of Computer Science and
Information Technology, RMIT
University
GPO Box 2476V, Melbourne
VIC 3000, Australia
+61 (3) 0434 524935
paul.mcintosh@
internetscooter.com

Margaret Hamilton
School of Computer Science and
Information Technology, RMIT
University
GPO Box 2476V, Melbourne
VIC 3000, Australia
+61 (3) 9925 2939
margaret.hamilton
@rmit.edu.au

Ron van Schyndel
School of Computer Science and
Information Technology, RMIT
University
GPO Box 2476V, Melbourne
VIC 3000, Australia
+61 (3) 9925 9677
ronvs
@rmit.edu.au

ABSTRACT

This technical report presents detail of the evaluation methodology used for researching the use of 3D for UML state machine diagrams. The methodology is derived from a user centred design approach called “Sequential Evaluation”, which has been applied in the past to immersive virtual reality environments. This technical report first provides background into 3D for UML, then background into 3D for UML state machine diagrams and then how “Sequential Evaluation” can be applied to evaluating 3D UML state machine diagrams. This technical report does not contain results of user studies but uses information available with the UML toolset under test to demonstrate application of the evaluation methodology. The results shown are only “indicative” i.e. they indicate the types of results that can be produced through the “Sequential Evaluation”. Actual user study based results of the application of this methodology are to be published in MODELS’08. This report builds on previous research, coined X3D-UML, where UML (Unified Modelling Language) is extended through X3D (eXtensible 3D) to create a standards based approach to 3D software visualisation.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques—Computer-aided software engineering (CASE); I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality; D.1.7 [Programming Techniques]: Visual Programming

General Terms

Documentation, Performance, Design, Management, Measurement, Human Factors, Standardization, Languages.

Keywords

X3D-UML, UML, X3D, Web3D, 3D Software Visualization, VRML, Virtual Reality Modeling Language, Unified Modeling Language, 3D State Machine Diagram

1. INTRODUCTION

3D computer graphics have been used extensively in computer games. As with demand on computer processing speed, hard drive space and ram size, demand for better 3D graphics has resulted in personal computers having 3D graphics capability as standard. Personal computers also play a large role in many software development houses and, although the capability may be still used for after hours gaming, the 3D graphics functionality remains untapped for daily software engineering tasks.

Although the idea of 3D software visualisation might be appealing to some and possible application domains wide and plausible, there has been a shortcoming in research evidence that shows measured benefit of software visualisation to industry [1]. Without such evidence, software notations and their associated CASE (Computer Aided Software Engineering) tools, will not attempt to better the performance of software engineers through the use of 3D.

This paper presents how an aspect of an existing software visualisation notation, the state machine diagram, can be given a 3D visualisation in such a way that its use can be measured and validated. By maintaining all existing tools and processes at the software engineer’s disposal and adding to that environment our 3D visualisation in a controlled way, strong evidence can be gained as to the benefits and issues solely attributed to the 3D view.

This paper covers firstly the motivation and background of the 3D visualisation research to date. Problems with current 2D state machine diagrams and our possible solution employing X3D-UML are discussed. We explain Sequential Evaluation and elaborate on how it is applied to evaluating 3D state machine diagrams. We further present our methodology using indicative user task analysis results derived from UML tool documentation and a set of 30 UML example models provided with the UML tool.

2. MOTIVATION AND BACKGROUND

Our research to date is motivated by the premise that advances in software methodologies are driven by evolution and that this evolution is heavily influenced by enumerable and uncontrollable variables. While it may be possible to derive an experiment to prove beyond doubt that one visualisation technique provides benefit over another in one specific situation, it is very difficult to extend those results to typical real life software development environments. Our research is therefore aimed at extending existing development environments and notations. Although those environments and notations might not necessarily be ideally suited to 3D, the results of this research are more broadly applicable to today's software engineering practices.

For example, Ware et al.[2] tested the performance of 3D software visualisation when applied to viewing complex software systems. Their results show a measurable reduction of error rates, due to the improved capacity of engineers to distinguish between edge crossings more easily. Although this evidence is valid, one problem has been that it was not based on a common standard visual notation that would be found available in present day environments, such as the UML (Unified Modelling Language). The benefits gained may be countered by any number of other external factors, such as an engineer's inexperience with the notation, lack of third party tool support, being unable to print the diagram to meet documentation requirements or a company policy about using open standards.

The UML is an existing visual notation which is a commercially accepted standard. Although it is currently a 2D notation, it has been noted by Booch et al. [3] that

"the UML allows you to create three-dimensional diagrams, meaning that they are graphs with depth, allowing you to "swim" through a model." [3]

There has been some past research into the domain of 3D UML, with notations being suggested by Gil and Kent [4] and Gogolla et al. [5]. Furthermore, the user experiments conducted by Dwyer [6] provide some qualitative evidence that complex software system architecture can be more readily understood as UML in a 3D space.

Our work is closely related to and follows on from that of Gil and Kent [4] who suggested that 3D software modelling may provide a means of combining UML diagrams to aid in understanding the modelled system. Gil and Kent also recommended that such diagrams require validation in real world projects and suggested that VRML (Virtual Reality Modelling Language) could be a means of easily producing such visualisations for this purpose.

Our previous research, McIntosh et al[7], has been based on rendering 3D UML visualisations using X3D (eXtensible 3D), the XML (eXtensible Markup Language) successor to VRML. Similar to VRML, 3D models are able to be described using text and displayed within a web browser capable of rendering X3D. X3D is an open, ISO approved standard [8].

Although VRML, as a Web3D technology, has been in existence for over 10 years, McIntosh et al [7] outline how only very recent advances in this Web3D rendering technology allow the display of real world software systems in such a way. The example, given in figure 1, was a real world software system of over 700 classes, rendered as a simple 3D UML class diagram. The use of X3D for

such diagrams has been demonstrated further with class diagrams of over 4000 classes created [9].

During our previous research we noted that in one demonstration of the visualisation, an audience member proclaimed that viewing the visualisation in motion made them "feel sick". This anecdotal evidence highlighted the fact that issues, specific to the 3D aspect of the visualisation, can be quickly realised if the viewer is already familiar with the notation. Based on this observation and with the aid of X3D-UML, it is possible to conduct an experiment which proved (or disproved) that viewing software system as a static 3D view is better than a moving 3D view, on large projector screens. This suggests that X3D-UML can be used as a means of testing 3D notation and to eventually determine if there is a valid use for such notation, once the extraneous issues have been uncovered and removed.

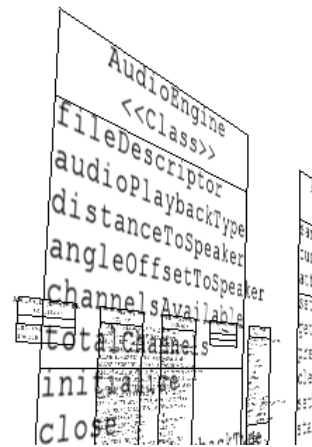


Figure 1 - An example of X3D-UML, with over 700 classes from an existing software project, translated and displayed as UML in a 3D space.

The initial incarnation of X3D-UML utilised XSLT (the language for translating XML documents), to translate source code into X3D-UML. To add flexibility of functionality above XSLT and to enable better integration with UML model data, X3D-UML has now been implemented as a library derived directly from UML Diagram Interchange Specification [10] and extended as required, with X3D being generated from the library. Although it is acknowledged that the use of this specification is limited, due to poor adoption by tool vendors, it is extremely powerful for quantifying the changes required to UML to facilitate new 3D notations. The diagrams presented in this paper are achieved through simply adding a "Z" value for the third dimension to the position attribute of the GraphElement class, in addition to the existing "X" and "Y". As graph nodes and edges inherit from GraphElement, this step can enable all UML diagrams to have elements which can exist on different planes in the same view.

One important aspect of X3D-UML is that since it builds on UML infrastructure, it can leverage existing UML functionality such as tool integration, code generation, round trip engineering, exporting/importing via XMI (XML Metadata Interchange) and pure visualisation importing/exporting via X3D; all of which have been considered to be practical aspects desired of software visualisation tools [11].

3. 2D STATE MACHINE DIAGRAM RESEARCH QUESTION

In this paper, we address the research question of how to visualise large state machines, with their layers of substates, in three dimensions. Our first investigation in 3D extensions to UML is to extend the state machine diagram so that substates can be more easily visualised. We have decided to base our investigation on a UML tool which makes extensive use of state machine diagrams, called IBM Rational Rose RealTime, also known as “Rational Rose Technical Developer” <http://www-306.ibm.com/software/awdtools/developer/technical/> and here referred to by its common name RoseRT.

RoseRT as a tool in itself is a very good candidate for aiding the investigation into extending state machine diagrams, since the source code is generated directly from these diagrams and this code is not normally edited outside of the tool. The benefit of such a system is that the diagram is not simply a documentation object which may be read once, then put aside and ignored or allowed to become outdated; instead it is the primary means of visualising and interacting with the code. The software engineer must understand the state machine diagram and must interact with it in their day-to-day tasks.

RoseRT implements the state machine diagram using a layered approach. An example of a RoseRT state machine diagram implementation is pictured in figure 2. The top-level state machine diagram presents the user 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 to indicate that there is further information available. By clicking on the state, a new window tab opens for the user and it contains the respective substate diagram, which itself may also have further substates, and so on. This layered approach is good at abstracting detail but may cause problems when trying to understand the full state machine diagram, due to necessary detail being hidden or buried. In addition, having separate diagrams requires the viewer to create their own mental map of the associations when following state transitions.

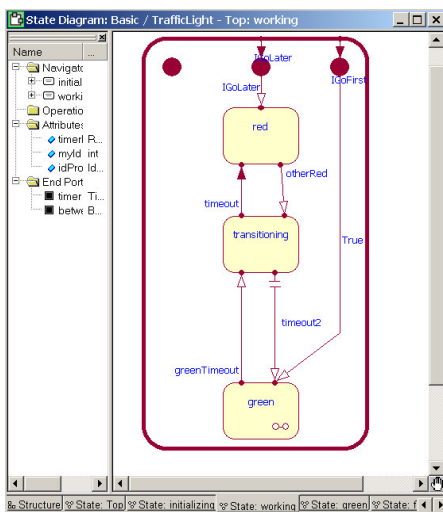


Figure 2 - Example of a complete state machine with substates displayed in different windows, using RoseRT.

An alternative approach to presenting state machine diagrams used by other UML tools, such as IBM Rational Software Modeller, is (by default) 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 one view; however the complexity of the state machine diagram is limited to the amount of information that can be drawn in a single diagram. Also diagrams become difficult to manage as, for example, adding a new state lower down in the hierarchy, requires the whole state machine diagram to be laid out again to accommodate the new object. An example of an IBM Rational Software Modeller equivalent of the RoseRT diagram Figure 2 is given below in Figure 3.

4. POSSIBLE 3D UML STATE MACHINE DIAGRAM SOLUTION

The X3D-UML extension presents a 3D state machine diagram using a combination of the two approaches discussed in the previous section. Substates are still displayed as separate “sub diagrams” but the third dimension (Z) is used to position substate layers below and relative to their superstate. From a “front on” view the state machine diagram may appear similar to the single complete view of the 2D state machine diagram (as shown above in Figure 3), however the user has much more ability to navigate and position the state machine diagram to whatever view they feel aids them in understanding and/or explaining the complete diagram. An example of an X3D-UML state machine diagram, generated directly from RoseRT model information is presented in figure 4. This is a view of the same RoseRT diagram as presented in figure 2.

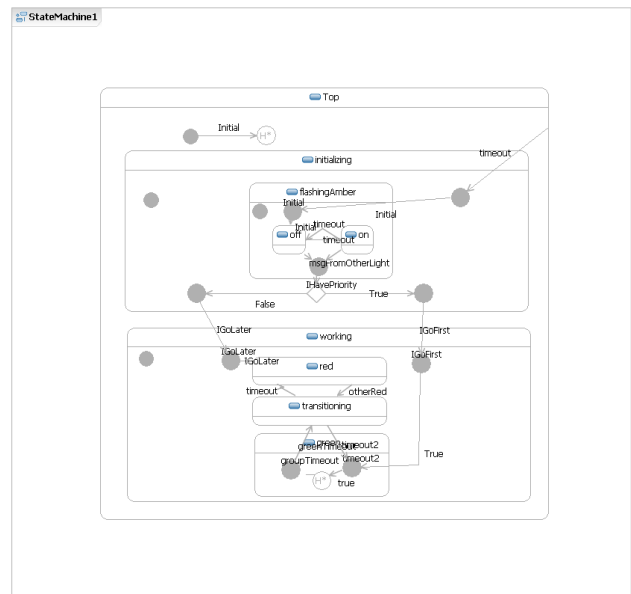


Figure 3 - Example of all substates and superstates displayed in the same diagram, using Rational Software Modeller.

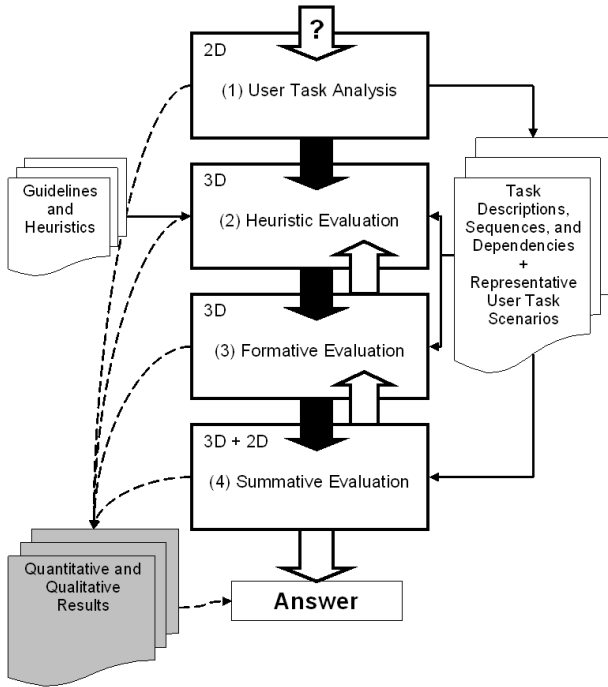


Figure 5 - Sequential Evaluation methodology as applied in researching advanced UML visualisation through X3D. “Sequential Evaluation” is a methodology defined by Gabbard et al. [12] and a slight variation is also presented by Bowman et al. [13].

5.1 Sequential Evaluation Chromacoding Example

In this section the analogy of “chromacoding” is used as a presentation aid for how our research methodology is executed. Chromacoding is a technique of applying colours to source code to help the software developer more quickly interpret the code. An example is shown in figure 6 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).

Making use of the chromacoding analogy, the following section walks through a sequential evaluation process as applied to X3D-UML:

The process starts with the research question; in our case for this paper, the question might be: “Is there benefit in 3D state machine diagrams for software engineering tasks?” For the chromacoding

example analogy, one question could have been “Does colourising source code text speed up software development?”

```

using MindTouch.Deki.Logic;

namespace MindTouch.Deki {
    using File = System.IO.File;
    using RPC_File = MindTouch.Deki
    using HttpPostedFile = MindTouc

    public class DekiContext {

        //--- Class Fields ---

        //--- Class Properties ---
        public static DekiContext C
            get {
                DekiContext dc = Cu
                if (dc == null) {
                    throw new Inval
                }
                return dc;
            }
    }
}

```

Figure 6 - Example of chromacoding of C# code from Visual Studio 2005. In this example comments appear in green text, operators in black and keywords in blue

(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 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 “Heuristic Evaluation”, with the help of information from “User Task Analysis” the initial extension is created based on the research question. Experts are called on to review the implementation before showing it to the users, where possible guidelines are to be followed. 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 as much”

(4) In “Summative Evaluation” the extension has been refined with the help of information from “Formative Evaluation” and users are called on to test the implementation against the old 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” and “users would like to choose their own colour scheme”.

The above example shows that from running through the Sequential Evaluation process we can demonstrate that there is a significant benefit in the chromacoding extension. We have also produced valuable 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.

It should be also noted that if problems have occurred, then it is possible to iterate through steps. If users found the implemented colour scheme distracting, we are able run through another evaluation pass where users are able to choose their own colour scheme.

Above is a summary of the Sequential Evaluation process, and in the next sections, we present the detail of each stage.

5.2 User Task Analysis

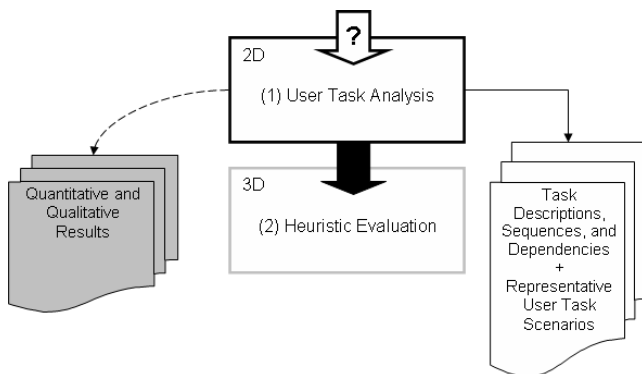


Figure 7 - User Task Analysis

“User Task Analysis” has been derived from the book “User and task analysis for interface design” [14], which outlines 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 they 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 do 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. User sessions are undertaken with the researcher and the user away from the work/study environment. In addition to this, user profiles are captured with a questionnaire prior to the user session.

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 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 thinking of an underlying metaphor when using the diagram?
- Do users get frustrated with the current diagram and, if so, how?

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, model metrics can uncover how many such diagrams exist. In section 6 we present more detail on using model metrics to predict benefit in new visualisation techniques.

5.3 Heuristic Evaluation

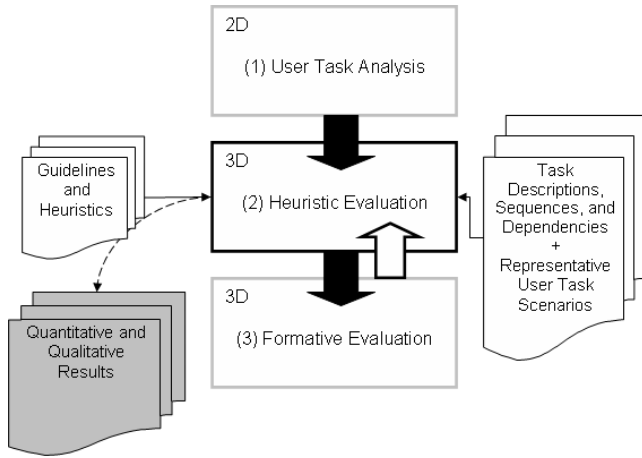


Figure 8 - Heuristic Evaluation

Heuristic Evaluation is “Rule of thumb” evaluation by expert evaluators to provide a “sanity check” for the visualisation against 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. It is not expected that the expert evaluators will be able to provide specific feedback on 3D UML (i.e. expert comparison against 3D software visualisation standards, as there are none), however it is expected that either UML or 3D expertise will be applied to give valuable first round feedback and information.

Expert Evaluators

The following groups of people have been categorised as expert evaluators:

- Tool Specialists (evaluators with direct experience with the 2D UML tool being used for the 3D comparison)
 - Consultants
 - Product 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 currently no guidelines on how to present UML diagrams using 3D. There are however a number of other individual sources of guidelines and heuristics for UML and 3D which could be utilised. These guidelines can also be used to invalidate the use of an unusual, but possible, 2D diagram that achieves a similar effect as the proposed 3D diagram.

Guideline might include the following:

- UML Style Guidelines
- Toolset Guidelines provided in training and documentation
- 3D User Interface Guidelines
- Emergent Guidelines (specific 3D UML guidelines evolving from this research)
- X3D Guidelines
- Personal Expert Evaluator “Rules of thumb”.

As an example of using guidelines, the 2D complete state machine diagram presented earlier in figure 3 does not meet guidelines set by Ambler [15]. Although it achieves a similar effect to a complete 3D state machine diagram, it is not recommended due to overuse of substates in a single diagram, making the diagram too complex. Ambler’s guideline #206, states to “Create a hierarchy of state machine diagrams for very complex entities” [15]

5.4 Formative Evaluation

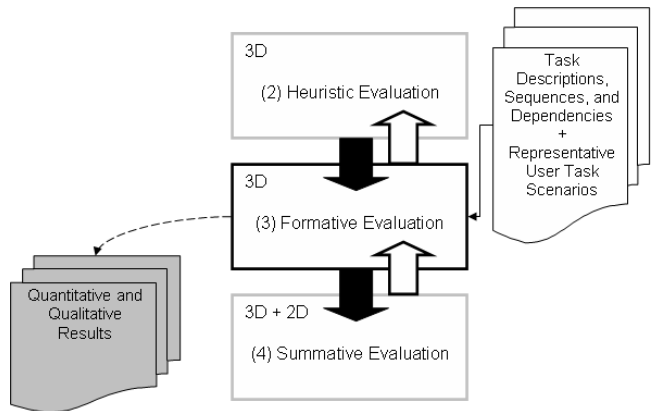


Figure 9 - Formative Evaluation

“Formative Evaluation” involves seeking feedback from users as the visualisation evolves, prior to it undergoing direct comparison tests (in Summative Evaluation). The researcher works 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).

5.5 Summative Evaluation

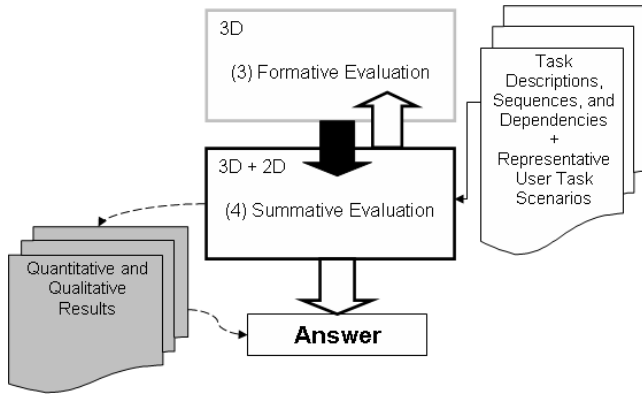


Figure 10 - Summative Evaluation

“Summative Evaluation” will directly compare the 2D and 3D extension by testing performance of the final visualisation against representative user task scenarios captured in the user task analysis stage. Manual and automated metric gathering can be put in place and statistical analysis used to determine real world benefit in terms of the productivity measures.

The metrics to be gathered are:

Time – Time to complete tasks defined from user task analysis

Accuracy – The number of mistakes

Efficiency – Calculated value based on time and mistakes

Comprehension – Tests requiring information available in both a single 2D and a single 3D diagrams to be answered.

Emergent Knowledge – Tests requiring information available in a single 3D diagram but requiring combinations of 2D diagrams or other sources to be answered.

User Perceived Comparison – Post hoc questionnaires where users rate ease of use of each visualisation

6. INDICATIVE USER TASK ANALYSIS

For the purposes of further demonstration of benefit of our 3D state machine visualisations, we are able to produce some indicative user task analysis results, which we have collated prior to involving users. By reviewing the RoseRT online help, some predefined user tasks can be analysed and these help present further the methodology in real terms. We realise that although these are actual user tasks, without conducting user sessions there is no evidence as to the significance of our results for these tasks (i.e. how often they are performed, if at all and how critical they are).

In RoseRT, for understanding a complete multilevel state machine diagram the software engineer must navigate and study the complete state machine diagram to all levels. In RoseRT this can be broken down into navigation interaction techniques. The user is given a number of navigation options for traversing up and down the state machine diagram hierarchy, which are listed below:

Superstate to substate navigation

- Double click on state machine diagram with icon
- Right click on state and select “Go Inside”
- Select the substate and Double click it in the Navigator (tree browser control)
- Selecting the substate and pressing “CTRL+NUM+” (Holding the Control Key and pressing “+” on the numeric keypad)
- Selecting the substate tab in the browser window (if it is present)

Substate to superstate navigation

- Selecting the superstate tab in the browser window
- Right click on state and select “Go Outside”
- Select the superstate and Double click it in the Navigator (tree browser control)
- Pressing “CTRL+NUM-” (Holding the Control Key and pressing “-” on the numeric keypad)

Each of the above interaction techniques requires some level of physical effort by the software developer, whether it is a mouse click sequence or keypad sequence. By analysing the number of interaction techniques, it is possible to capture a solid measure of software developer’s current physical effort for navigating a diagram and then compare the physical efficiency gains that a complete, single view 3D state machine diagram view would give.

To measure physical interaction technique effort, metrics about the extent of substate usage within RoseRT UML models can be generated. To gather the statistics we developed a RoseRT script “statelevels.ebs”, which utilised the RoseRT extensibility interface (API). This script walks through each model and reports the following:

- name of the model file
- metrics on model size (Packages, Capsules, Protocols, Classes, States)
- metrics on numbers of states at each substate level

Below is an example of the script output from the “Traffic Lights” model, which is an example model, provided for learning purposes with RoseRT. The output shows that the model has a total of 30 states, contained amongst 6 capsules (which contain the top level state machine diagram). 80% of the states in the model are not visible at the superstate level and require some form of navigation to view.

```

Model: C:\Program Files\Rational\Rose
RealTime\Examples
\Models\Cplusplus\TrafficLights\TrafficLights.rtm
Metrics
totalPackages, 6
totalCapsules, 6
totalProtocols, 6
totalClasses, 18
totalStates, 30
total Level 1 States, 6
total Level 2 States, 12
total Level 3 States, 10
total Level 4 States, 2
total Level 5 States, 0
total Level >5 States, 0
Average States Per Capsule = 5
Percentage of Substates = 80

```

Using the above metrics we can produce a profile of the average state machine diagram and deduce from that a metric indicating the average number of interaction techniques associated with each diagram, assuming a particular usage pattern. Assuming that the user will navigate down to each state at least once in any given diagram they are working on, we give level 2 states the value of 1 for interaction techniques, level 3 has the value of 2, level 4 the value of 3 and so on.

For the “Traffic Lights” model, the average state diagram contains 1.0 top level states (some state diagrams are empty), 2.0 level two states, 1.7 level three states and 0.3 level four states. Calculating the total interaction techniques gives us an average interaction “cost” of 6.33 interaction techniques per diagram compared to a possible zero for a 3D state machine diagram. Also, we note that the interaction techniques carry a mental mapping “cost” which is likely to be higher than the simple physical effort “cost”. For each interaction that the user needs to navigate between diagrams, they also need to mentally retain the association between the diagrams themselves, and remember where they are currently within the hierarchy of diagrams. The deeper the hierarchy the more difficult this mental mapping would become.

Although the above metric may appear contrived, it is founded on logical assumptions and this analysis will ideally eventually form the basis for a useability performance model. Performance models are a valuable tool as they provide calculated predicted benefit prior to expensive user research [13]. However, before we can validate this sort of model, we need to analyse current user tasks and resolve some issues or unknowns such as, whether or not the user navigates up and down the hierarchy, which would mean adding more interaction techniques, or whether the user needs to use interaction techniques in the 3D view.

To provide more indicative results, all the example models provided with the RoseRT toolset have been processed with our script. The models surveyed may not be truly representative of commercial models, since they were intended solely for training purposes. However, we note that one model, the “DCS” (Distributed Connection Service) is a commercial grade model and serves as the source code for a feature provided with RoseRT.

To give an a general overview of the indicative benefit of 3D state machine diagrams, in Figure 11 we present a bubble plot of the survey results, where the average number of interaction techniques “cost” per diagram is plotted against the number of state machine diagrams each model contains. Bubble size is based on total interaction techniques per model (average interaction techniques per diagram x number of diagrams). 3D state machine diagrams have the most benefit for large models that use substates regularly or models of any size that have deeply nested state hierarchies. Models without substates receive no benefit and do not appear on the plot, however there should be no negative impact for those models because the 3D functionality will simply be ignored by the user.

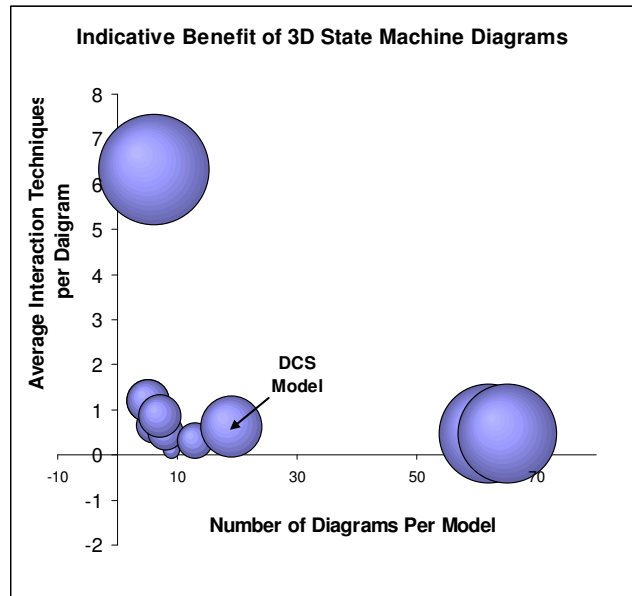


Figure 11 – Indicative benefit of 3D state machine diagrams represented by bubble size indicating total interaction technique saving per model. 3D state machine diagrams have the most benefit for large models that use substates regularly or models of any size that have deeply nested state hierarchies. The “DCS” model is highlighted because it represents a model likely to be found in industry.

The results for our 30 example models surveyed showed the following properties:

- 12 models had substates, and thus would benefit from a 3D state machine diagram representation.
- Of the models with substates, the average interaction technique saving was 1.06 per diagram.

- Only 8 models contained more than 10 diagrams, indicating that the majority of the models surveyed were small.
- The “DCS” commercial model contained 19 diagrams and the performance model predicted the average interaction technique saving as 0.63 per diagram.
- The example models provided with RoseRT may be suitable for education research into 3D state machine diagrams for learning tasks. The models are designed purely for learning and present a variety of model sizes and state diagram hierarchies.

7. SUMMARY AND FUTURE RESEARCH

In this paper, we have discussed how 3D software visualisations can be approached as an evolutionary step from current UML visualisation environments. We have presented the case for modelling a 3D state machine as an X3D-UML diagram and measuring the benefit. Rather than testing a visualisation which may be a “paradigm shift”, that requires the software engineer to learn new notations and new ways of working, we have built on their current knowledge and work flow.

Because the visualisation is leveraging off existing implementations, these implementations can be utilised to provide indicative performance models based on the data as it is today. Our indicative results show that even for the small example models, 40% make use of substates in state machine diagrams and would benefit from reduced interaction techniques and mental mapping requirements that a 3D view would provide. Though the final results for a completed study may show no overall user benefit for 3D state machine diagrams, this initial metric gives some indication of the value in pursuing the study further beyond the initial concept.

We have presented a complete methodology, “Sequential Evaluation” [12, 13], and discussed how it can be applied to evaluation of 3D UML state machine diagrams. The basis of our methodology presented in this paper, is user task analysis and measuring against real user tasks. The results of this study will provide strong evidence that can be directly applied to industry.

The evaluation of 3D state machine diagrams is currently being undertaken using commercial users and commercial models. Published results from that evaluation will also include suggested improvements to the methodology based on its use for evaluating 3D UML. Two more X3D-UML visualisations are planned to be evaluated using the methodology, UML diagrams extended to visualise complete software systems using 3D and 3D hardware design models integrated with UML models.

8. ACKNOWLEDGMENTS

We would like to thank RMIT University for the PhD scholarship funding of this research.

9. REFERENCES

- [1] Diehl, S. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software*. Springer-Verlag New York, Inc., 2007.
- [2] Ware, C., Hui, D. and Franck, G. *Visualizing object oriented software in three dimension*. IBM Press, 1993.
- [3] Booch, G., Rumbaugh, J. and Jacobson, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.
- [4] Gil, J. and Kent, S. *Three dimensional software modelling*. IEEE Computer Society, 1998.
- [5] Gogolla, M., Radfelder, O. and Richters, M. *Towards Three-Dimensional Animation of UML Diagrams*. Springer, 1999.
- [6] Dwyer, T. *Three dimensional UML using force directed layout*. Australian Computer Society, Inc., 2001.
- [7] McIntosh, P., Hamilton, M. and Schyndel, R. v. *X3D-UML: enabling advanced UML visualisation through X3D*. ACM Press, 2005.
- [8] Web3D Consortium *Extensible 3D (X3D) - ISO/IEC FDIS (Final Draft International Standard)*. Web3D Consortium, 2004.
- [9] Anslow, C., Noble, J., Marshall, S. and Biddle, R. X3D web software visualization in action! In *Proceedings of the Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion* ACM, 2007.
- [10] Object Management Group *Unified Modeling Language: Diagram Interchange - version 2.0*. Object Management Group, Inc., 2003.
- [11] Bassil, S. and Keller, R. Software Visualization Tools: Survey and Analysis. In *Proceedings of the Proceedings of the 9th International Workshop on Program Comprehension*. IEEE Computer Society, 2001.
- [12] Gabbard, J. L., Deborah, H. and Swan, J. E. *User-Centered Design and Evaluation of Virtual Environments*. IEEE Comput. Graph. Appl., 1999.
- [13] Bowman, D. A., Kruijff, E., LaViola, J. J. and Poupyrev, I. *3D User Interfaces: Theory and Practice*. Addison-Wesley/Pearson Education, 2005.
- [14] Hackos, J. T. and Redish, J. C. *User and task analysis for interface design*. John Wiley & Sons, Inc., 1998.
- [15] Ambler, S. W. *The Elements of UML(TM) 2.0 Style*. Cambridge University Press, 2005.