THORR: A Focus + Context Method for Visualising Large Software Systems

Eoin McCarthy, Chris Exton
SVCR Group
Department of Computer Science and Information Systems, University of Limerick

Eoin.D.McCarthy@ul.ie, Chris.Exton@ul.ie

Abstract

Many attempts have been made to construct tools that aid a programmers understanding of source code and system structure. Syntactic highlighting and tabbed interfaces can only be useful to a degree; greater steps need to be taken to accelerate programmer comprehension and in reducing programming and maintenance times. This paper presents a new programming environment for software engineers, THORR. The programming environment hopes to overcome some of the shortcomings currently associated with similar visualisation projects, in terms of balance between level of detail and context, by using 'Degree-of-Interest' information visualisation techniques to implement an attention reactive user interface. An overview of the prototype tool and its basic functionality is also given.

1 Introduction

Large scale software systems, like production-sized legacy programs can be incredibly difficult to maintain or update. Much of the time expended in performing these tasks is taken up by refreshing a programmer's knowledge of the system or training a new engineer into understanding the system(Knight, 2001). Under these circumstances, a programmer's productivity can decrease as the work can be laborious and tiresome. As a result errors can occur or projects can be late.

Software visualisation was brought about as a means of easing the maintenance section of the software lifecycle. It is believed that by visualising a software system graphically, be it program or algorithm animation, the knowledge decay that programmers experience can be slowed, and the length of time it takes to remember or discover code can be reduced.(Ball and Eick, 1996), (Grundy and Hosking, 2000).

During the design phase of software development, programmers will use diagrams to illustrate the software design in the form of UML diagrams, using industrially recognised tools such as Rational Rose, and Select Enterprise. However, it is difficult to find equally prominent tools that aid programmer understanding of an implemented system. One of the reasons for this is maybe that many visualisation systems are tested using small software examples and as a result may not scale well to industrial-sized software systems.

In this paper, we present a new software visualisation tool for software engineers called THORR, in the hope of reducing programmer effort, and shortening maintenance times. We will attempt to overcome the problems of scalability, and navigating large data sets that have prevented other visualisations from being widely accepted.

Section 2 will present the methods of visualisation that THORR utilises, describing the innovative animation techniques they have employed. Section 3 describes the architecture of the visualisation tool we have created based on the DOI technique, how it generates and processes valuable architectural information. Section 4 identifies particular needs that user may have when using a visualisation system and discussed how THORR aids those needs. The fifth section deals with the implementation of the tool including some metaphors and methods used in developing and presenting the visualisation. Section 6 discusses an evaluation performed on the tool, and section 7 provides some concluding remarks.

1.1 Software Visualisation

While it has been described in many different ways for the purpose of this paper software visualisation will be defined as:

> *"...the use of the crafts of typography, graphic design, animation, and cinematography with modern human-computer interaction and computer graphics technology to facilitate both human understanding and effective use of computer software."*(B Price, 1992).

2 Visualisation Techniques

2.1 Fish-Eye Lens

The Fish-Eye lens distortion technique was original described by Furnas (Furnas, 1986). It was based on a technique best described as 'thresholding'. It was primarily used to visualise hierarchical structures, in which each element was assigned 2 numbers. The first number was based on its relevance and the second number was based on the distance between the information element in question and the element regarded to be the focused node. A threshold value was then selected that was compared with a function of these two values to determine which information would be presented and suppressed.

2.2 Focus + Context

One of the biggest difficulties in visualizing large quantities of information is the lack of screen space in which to visualize it. Users can become disoriented or even 'lost' in a visualisation where complex navigation is required to obtain information.

Focus + context techniques were introduced as means of displaying huge quantities of information on one screen. A distortion algorithm is used to magnify a certain part of the screen while de-emphasizing the rest. As a result, the user can have a constant overview of the information while focusing in on one particular area.

Card et al(Card and Nation, 2002) use the terminology *focus + context* when referring to visual techniques that provide simultaneous access to both overview and detailed information. According to Card, *focus + context* is based on the following three premises:

- The user requires both the overall picture and a detailed view.
- There may be different requirements for the information in the detailed view than in the overall picture.
- Both displays may be combined into a single dynamic view.

2.3 Degree-Of-Interest Animation

Developed at XEROX-PARC(Card and Nation, 2002) the DOI (degree-of-interest) tree is an example of an attention reactive interface, used for visualising hierarchical information. The *focus + context* method, which it utilises, differentiates it from other similar visualisations. While other *focus + context* visualisations apply distortion algorithms to the screen itself, fisheye views and perspective walls for example, the DOI tree applies transformations and magnifications to the tree to illustrate interest in specific nodes.

The researchers at XEROX PARC expanded on Furnas's work. In Furnas's graphs, all nodes were given the same intrinsic values, hence when one node is focused on; all other nodes are treated in the exact same manner. The DOI tree assigns different values to varying nodes depending on their relationship with the focused node. There calculation treats the children of the focused node as ordered and assigns a fractional degree-of-interest offset to the children, then a smaller fraction to the focused nodes' grandchildren and so on. Each time a new node is selected as the focused node, the DOI offsets for all nodes must be recalculated.

3 Support for User Needs

3.1 Scalability

The problem in creating an effective scalable visualisation tool is multi-faceted. Designers have to consider massive amounts of input to the tool; huge amounts of information processing and then must also have efficient algorithms, which display the information in an insightful manner. When these considerations are undertaken, then there is a chance for creating a truly scalable system.

There are many tools that have been implemented to show off one particular animation or visualisation technique, which they deem to be highly important(Robertson et al., 1998), (Sanjaniemi and Kuittinen, 2003) to name but a few. When these tools are evaluated, it is usually done with a sample set of miniature or 'toy' software systems. There is not enough emphasis placed on whether or not these tools will actually work in industry. They have not considered what will happen if the change the sample size from 200 LOC to 2million LOC, and more often than not unless the metaphor for visualisation has been design with huge programs in mind, it will not scale well.

The presentation of large amounts of information is often spread across an area that spans multiple screens, but THORR will attempt to visualise a meaningful overview of a system in a single display, which will result in a need for novel and complex layout algorithms.

3.2 Interaction

The difficulty in implementation of the interaction mechanism required in a visualisation is directly proportional to the amount of information it is required to visualise. When dealing with small sample sets, very simple interaction techniques will be adequate both in terms of user-interface and navigation of visualisation. It is difficult for a user to get confused when dealing with small amounts of information and as a result even badly designed interaction techniques could be sufficient.

When the amount of information increases to a very large number of information elements, there must be more advanced techniques put in place. Users navigation must be stringently designed and aid the user, by only constraining movement to what is required. When dealing with huge graphs, complete freedom of movement can result in major difficulties. (Cockburn and McKenzie, 2000), (Wood et al., 1995)

THORR will be designed with all of these interaction difficulties in mind. It must allow users to retain overall context at all times. It must also give users the ability to move from high-level information, which has very little detailed information, to lower level more comprehensive information with ease. The most successful interaction mechanisms constrain its users in some way, so THORR will be designed to limit the movements that can be carried out by the users. THORR will also implement a 3D interaction technique, which when combined with the other interaction requirements will result in complex implementation but will enhance the users visualisation experience.

3.3 Interoperability

For a tool to be truly accepted in industry it must allow for interfacing with other tools that software maintainers have at their disposal, whether they be other visualisation, design, or re-engineering tools. It must be as open as possible, in terms of what platforms it can run on, what languages it can accept as input, and how extensible it is. With so many tools only in prototypical stages, it is difficult to find ones that accept multiple languages as input, as they are generally focused on proving the concept of their own specific visualisation technique. These 'toy tools' also create their own informal definition of their visualisation layout resulting in an output formation that is only useful to that specific tool.

A common interchange format would be useful to allow interoperability between tools. Such as interchange would act as a 'buffer' or 'interface' between different tools and it would make it easier to judge the benefits of individual tools. Identical information sources could be used when performing comparisons between tools, resulting in better overall qualitative results. o input or export formats have been described.

The THORR tool will attempt to emulate some of the better design decisions that other visualisation systems have implemented. It will implement an input system, which will accept a common interchange. This results in a tool that will be language independent and also encourage interoperability with tools that accept similar formats. It will be designed to be somewhat extensible and also be implement in a language that is not operating system specific.

3.4 Automation

The amount of automation required in visualisation systems is inversely proportional to the simplicity of the interaction controls used in the system. In other words, the easier it is to traverse large amounts of information in a tool, the harder it should be to design and implement to the navigational controls. THORR will be required to automate some if not all of the more time-consuming visualisation tasks. It should be able to automatically set-up projects, and extract meaningful information from software systems without requiring an overt amount of user input. The visualisation should assist the user's cognitive processes by implementing a navigational system that will allow the user quick and easy access to all nodes in the graph, and it should be able to automatically decide which nodes to display in detail and which nodes should be filtered. There is, however, a fine line between helping users through the automation of tasks and hindering users by applying excessive constraints, so a balance between the two must be struck.

4 THORR Visualisation

Studies have shown that up to 70% of lifecycle costs are consumed within the maintenance phase, with 50% of these costs relating to comprehension alone (de Lucia and Fasolino, 1996),(Rajlich, 1994). Hence, according to these references, up to 35% of the total lifecycle costs can be directly associated with simply understanding the code.

The overall aim of the THORR visualisation then, is to decrease both the time and effort expended by programmers in trying to understand software systems. Visualisation systems have shown results in improving the speed at which users understand, remember, and find information.(Robertson et al., 1998),(Robertson et al., 1991). *Focus + context* methods have been proven increase the speed of use of visualisation systems by retaining overall context(Mitchell and Cockburn, 2003). We hope that by integrating these attributes, users will be able to quickly navigate through large-quantities of information extracted from software systems.

The visualisation itself is a high-level hierarchical visualisation employing degree-of-interest animation. The graph consists of a hierarchical tree of coloured cubes corresponding to various components of the software system, such as classes, superclasses, subclasses and interfaces. Edges between nodes correspond to inheritance parent-child relationships.

Its major advantage over similar tools is the fact that it can visualise thousands of nodes on one screen. It can be navigated without ever having to zoom in or out on a particular section of the tree, thus retaining context at all times. It also circumvents the need for scrolling as all the information is displayed on the one screen.

4.1 Degree-of-Interest Computation

When the tree is first laid out, each node is assigned a DOI value depending on its relationship to the root node.

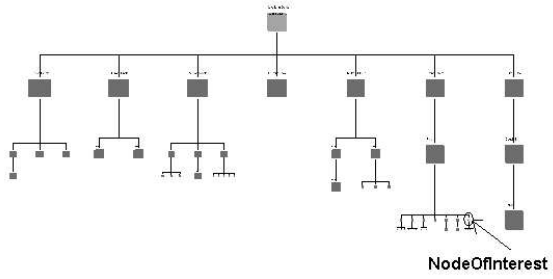

Figure 1: THORR graph in intitial state

When a user selects a node to be *interested* in the DOI for each node is then calculated according to the relationship it has with the focused node. All DOI values are between 1 and 0, where 1 is assigned to the focused node and 0 is assigned to the nodes that have little or no relationship with the focused node. To demonstrate this *focus + context* computation, consider the screen captures of the same graph taken from THORR at two different states in Figures 1 and 2. Figure 1 is a screen capture at the initial layout of the graph and Figure 2 captured the graph after*NodeOfInterest* has been selected by the user.

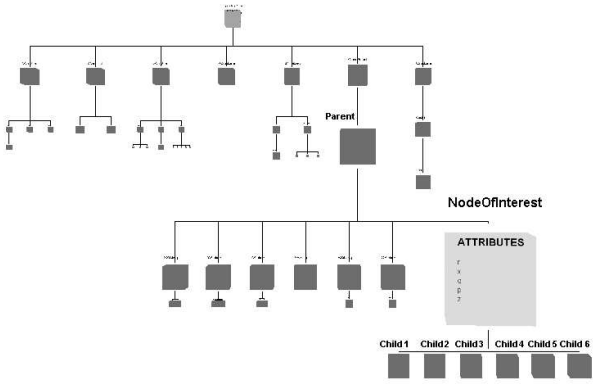

Figure 2: THORR graph in focussed state

When a user chooses a node to become the focused node, as in Figure 2, THORR firstly assigns that node a DOI of 1, then assigns a lesser offset to its children, and parents, then a smaller offset again to their respective children and parents and so on until the least related nodes have offsets of 0. Positions are then calculated, taking into account the changes of size that will occur, and a smooth interpolation between the two states is animated.

4.2 THORR Functionality

Text will always be an integral part of visualizing software, so the option of viewing the source code of individual methods has been included.A searching functionality has been incorporated into the tool to allow for even faster access to components of the visualisation. Searching the visualised information for specific or partial-phrases can yield in multiple results. The ability to capture both screen shots and the state of the graph for later analysis has also been integrated into the THORR visualisation. Simply clicking on the relevant screen capture will restore the graph to that state.

4.3 Node Characteristics

Each node is presented on screen in the form of a 3D cube. A user can interact with these cubes not only by double-clicking to expand them, but also by rotating them around the

Y-axis. This added ability allows information to be presented not solely on the front of the cube, but on 4 of the 6 sides.

This functionality is exploited by presenting different facets of Java classes. Constructors, operations and attributes were chosen as the relations to be presented on each of the sides. These divisions are similar to those implemented in the 'Javadocs' documentation tool of the Java API. Scrolling has been implemented to allow text to be presented in a sizeable manner, without reducing the amount of text that can be displayed on each side.

5 THORR Framework and Implementation

The THORR visualisation system focuses primarily on visualizing Java software systems and UML software designs in an efficient and comprehensible way.

5.1 Java

Java was selected as it is an excellent sample language to use in this prototype as it offers many different class relations such as inheritance, associations, inner-classes, super or sub-classes, local classes and packages, many of which are worth visualising. Java consists of a number of APIs, which are intended to be used or extended through inheritance with the aim of aiding productive programming by reducing programmer effort and increasing portability(Taivalsaari, 1996). While this ability to extend source code is useful, often to fully understand a class one must understand its superclasses and the classes it inherits from as well.

5.2 XMI

XMI(OMG, 1999) is the XML Metadata Interchange format standardized by W3C and the OMG. It is this standardisation that allows THORR to import both Java source code and UML diagrams. In the case of this tool, the information from the visualisation can be used in other tools and across platforms allowing for the combined use of many tools in a heterogeneous environment. A public domain tool, JavaRE(Andersson, 2001), analyses Java source and outputs information in the form of XMI documents.

6 Evaluation

An evaluation of Thorr was implemented in order to discern if the proposed methods of visualisation provide additional benefit to programmers. A simple tool A versus tool B evaluation was set up, between Thorr and the IDE Eclipse(Eclipse-Foundation, 2004). Eclipse was chosen because it is a widely used development environment when developing industrial sized Java software systems. It also has the ability to visualise inheritance hierarchies at a basic level and provides users with advanced search capabilities.

A pilot study was carried out to determine if changes to the evaluation design were required. There were a few small problems with the tool and the phrasing of the tasks that were identified and corrected for the actual evaluation.

The evaluation assigned 2 sets of tasks to 2 different groups. Group 1 carried out the Task Set A in Thorr and Task Set B in Eclipse, whereas Group 2 carried out Task Set B in Thorr and Task Set A in Eclipse. This swapping of task was employed as an attempt to negate dissimilarities in the programming experience between participants, the thinking being that participants that had difficulty performing task in Eclipse should also have some problems performing similar tasks in Thorr.

The tasks in each set were designed be exploratory and searching task that got more difficult as they proceeded through them. Participants were asked to locate certain classes, then attributes of classes, and also questions about the inheritance ancestry of classes. The length of time it took to finish each task was recorded and compiled.

The evaluation was carried out with 6 working professionals with varying levels of Java experience. 2 were classed themselves as being very experienced, whereas the other 4 described themselves as having intermediate knowledge of Java. The participants in each group had an even mix of average and experienced programmers. The results indicated that participants finished tasks on average 56% faster when using Thorr, and also finished more tasks correctly.

The users were also given a debriefing session in which they gave some qualitative information regarding the 2 tools. It was noted that there was a slight preference for using Thorr as an exploratory tool, as the participants preferred Thorr's navigation and interaction techniques. Some commented on its "..ease of use.." over Eclipse and stated that the "..layout was beneficial.." when exploring the graph.

7 Conclusions

Focus + context, and degree of interest techniques have proven effective in the field of information visualisation. The THORR visualisation tool aims to bring these helpful methods to the field of software maintenance. Its ability to visualise extremely large data sets, and also be able to focus on particular areas quickly and easily without losing context can be especially useful to software engineers. Further evaluation of the tool may help pinpoint what areas of the tool benefit users to most, in order to improve upon weaker areas of the tool and exploit its potential qualities.

References

Marcus Andersson. Javare - java roundtrip engineering, 2001. URL `http://javare.sourceforge.net/index.php`. Honours Reports of the University of Canterbury.

I S Small B Price, R M Baecker. A taxonomy of software visualisation. In *Proceedings of the 25th Hawaii International Conference on System Sciences*, volume II, pages 597–606, 1992.

Thomas Ball and Stephen G. Eick. Software visualization in the large. *IEEE Computer*, 29 (4):33–43, 1996. URL `citeseer.ist.psu.edu/ball96software.html`.

Stuart K. Card and David Nation. Degree-of-interest trees: A component of an attention-reactive user interface. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, page 231246, 2002.

Andy Cockburn and Bruce McKenzie. An evaluation of cone trees, 2000. Department of Computer Science University of Canterbury Christchurch.

Andrea de Lucia and Anna Rita Fasolino. Understanding function behaviors through program slicing. In *Proceedings of the 4th International Workshop on Program Comprehension (WPC '96)*, page 9. IEEE Computer Society, 1996. ISBN 0-8186-7283-8.

Eclipse-Foundation. Eclipse ide, 2004. URL `http://www.eclipse.org`.

G.W. Furnas. Generalized fisheye views. In *Proceedings of CHI '86, New York*, pages 16–23. ACM, 1986.

John Grundy and John Hosking. High-level static and dynamic visualization of software architectures. In *Proceedings of the 2000 IEEE International Symposium on Visual Languages (VL'00)*, page 5. IEEE Computer Society, 2000. ISBN 0-7695-0840-5.

Claire Knight. System and software visualisation, 2001. URL `citeseer.ist.psu.edu/knight00system.html`. Handbook of Software Engineering and Knowledge Engineering.

David Mitchell and Andy Cockburn. Focus+context screens: A study and evaluation, 2003. Honours Report from the University of Canterbury.

OMG. Xmi - xml metadata interchange, 1999. URL `http://www.omg.org/technology/xml/index.htm`.

V. Rajlich. Program reading and comprehension. In *Proceedings of Summer School on Engineering of Existing Software*, pages 161–178, 1994.

G. Robertson, M. Czerwinski, K. Larson, D. Robbins, D. Thiel, and M. van Dantzich. Data mountain: Using spatial memory for document management. In *ACM Symposium on User Interface Software and Technology*, pages 153–162, 1998. URL `citeseer.ist.psu.edu/robertson98data.html`.

George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: animated 3d visualizations of hierarchical information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194. ACM Press, 1991. ISBN 0-89791-383-3.

J. Sanjaniemi and M. Kuittinen. Program animation based on the roles of variables. In *In Processding of the ACM 2003 Symposium of Software Visualsiation (Softvis 2003)*, pages 7–16. ACM Press, 2003.

Antero Taivalsaari. On the notion of inheritance. *ACM Comput. Surv.*, 28(3):438–479, 1996. ISSN 0360-0300.

A.M. Wood, N.S. Drew, R. Beale, and R.J. Hendley. Hyperspace: Web browsing with visualisation. In *In Third International World-Wide Web Conference Poster Proceedings*, pages 21–25, 1995.