CORBAview : A visualisation tool to aid in the understanding of CORBA-based distributed applications

Declan Ryan
Department Of Computer Science
University of Limerick, Ireland
declan.p.ryan@ul.ie

Dr.Chris Exton
Department Of Computer Science
University of Limerick, Ireland
chris.exton@ul.ie

Keywords: POP-I.C. educational technology, POP-II.A. novices, POP-III.D. visualisation

Abstract

Middlewares provide the fundamental technology needed to create distributed applications, allowing developers to concentrate on their own specific needs. The developer may end up creating a very small part of the overall system themselves. They are able to make use of tried and tested middleware to aid in their application development. The tools provided by the CORBA middleware mean that much of the code used to create the distributed application is automatically generated for the developer. For a student learning about these middlewares, much of the development of the application that they are creating is done by pre-compiled, third party software. Understanding how the distributed application works can be difficult when you have only developed a very small part of the overall system.

This paper describes CORBAview, a visualisation tool that gathers and displays the network communication and inner workings of the objects in a CORBA-based distributed application.

Introduction

As large scale computer networks become more prominent in business and home use, so too does the middleware that supports their use. These middlewares provide the fundamental technology needed to create distributed applications, allowing developers to concentrate on their own specific needs, without having to develop the application from scratch. One such middleware is CORBA (Common Object Request Broker Architecture). The CORBA specification has been adopted by more than 800 companies, and is a vendor independent architecture (OMG 2002). CORBA is based on the principles of object orientation. All CORBA applications are made up of objects. The interfaces to all of the objects is strictly defined, whilst the internal workings of the objects are hidden from the rest of the system. Objects with the same functionality from different vendors could have the same interfaces, but the internal structure and design may be completely different.

A typical example would be the CORBA ORB (Object Request Broker). The ORB handles request routing, fault tolerance, load balance etc. It is the hub of the CORBA-based application. The ORB must provide certain services, with set interfaces to each, but each vendor may handle the provision of these services differently. Through the use of an IDL (Interface Definition Language), all objects can communicate with each other, regardless of the language the were written in. The developer specifies the interfaces to the services that will be provided by the server and accessible by the clients. The IDL compiler then creates the stubs and skeletons that will be used by the clients and servers respectively to convert the arguments from the native language to the IDL, or vice versa. This ensures that objects written in different langauges can commuincate with each other, and that the developer need only specify the interfaces, the actual distribution of the requests and replies are handled automatically

Developers can concentrate on creating their distributed application, knowing that the networking and distribution aspect is taken care of. But this can lead to difficulties for developers learning about distributed applications. Because one of the aims of CORBA is to make creating a distributed application seem as close to creating a local application as possible, a lot of the underlying work done by the clients and servers is hidden from the developer. The inner workings of the ORB is completely hidden (if it is third party software), and it is difficult to tell what is happening in the background, such as network communication.

Target Audience and Current Teaching Methods

CORBAview is aimed at anybody who is studying the fundamentals of CORBA and creating CORBA-based applications. These may be students or real-world developers. We will refer to these target users as students. A prerequisite for students using this tool would be a knowledge of the basic concepts of object orientation. Ideally the student should also have an understanding of the basics of network computing.

Tools are being created to aid in teaching in many computer science fields. The reason behind these tools is that using a more interactive approach will be more beneficial to the student compared to the standard methods. Interactive methods are also being used in place of the standard lecturing format (Rodger 1995), (Cordes & Parrish 1993). One of the earliest visualisation tools for education was created by Baecker (1981), which was a short video animation of sorting algorithms. Baecker makes the claim that a programs behaviour cannot be described by a static drawing, that there needs to be dynamic sequencing. The majority of educational visualisation tools use a similar logic, they aim to teach the student by allowing them to see a step by step execution of a program, rather than an overview of the function and results. Nørmark et al. (2000) created a tool called Java Eluciator that links together both the prorgam and documentation. Fragments of the program are shown along with documentation, aswell as the entire program. The tool aims to help the student by linking documentation and source code together so that they are both accessible to them at the same time.At the moment, there are few educational tools available in the area of distributed computing and CORBA.

Reading material available that deals with creating CORBA-based applications, deal with the creation of the client/server objects (Orfali and Harkey 1998). The functions of the ORB are discussed, but how the ORB goes about providing these functions is largely ignored.

Eisenstadt et al. (1993) suggested that many tutoring systems focus too much on examples which are too simple to give an accurate reflection of real world applications. CORBAview can show the student what is happening in every part of the distributed application, and not just specific, student-created objects, whislt also linking documentation with the visualisation of a dynamic, CORBA-based application.

Tool Requirements

Visualisation tools can either gather data from a monitored source(s), or the visualisation can be done using data created by some form of simulation tool. Tools that do gather data from a real source, rather than simulation, need some way of gathering the required data. Visualisation tools can be broadly categorised into two areas, tools whose purpose is educational, or those whose purpose is analytical. This is regardless of what they monitor/visualise, whether it be CORBA, a network, graph algorithms etc.

Analytical Tools : The aim of analytical tools is to gather information about the execution and flow of an application with respect to network traffic, execution time and detection of bugs. One of the fundamental requirements for analytical tools is that they do not affect the execution of the system that is being analysed. The effecting of the execution can sometimes be caused by the slowing of the executing application, due to the gathering of information by the analytical tool. Sometimes this is caused by the modification of the system being monitored to allow for the gathering of data. If the tool effects the execution of the system, then any results obtained by the tool are flawed, as the execution of the unanalysed system will be different.

Educational Tools : With an educational visualisation, the same criteria should also be applied. The purpose of an educational tool is to aid in the understanding of how the system, or aspects of the system work. Gathering data for analytical purposes is not the primary aim of most educational visualisation tools. However, this shouldn't mean that making modifications to the system being monitored should be acceptable. Making modifications means putting an extra burden on the student, as they have to learn how to use the tool, on top of trying to gleam knowledge from the visualisations it provides. Therefore, the data gathering part of an educational visualisation tool should not require the user to make modifications to their code. The way in which the data is gathered also affects the type of visualisations that can be created. A tool that gathers information from the computer network can be used to show the network usage, remote communication etc., but it will show nothing about the internal workings of any of the objects. We can see the network communication, but the not the context in which it was created. Conversely, a tool that gathers information about a particular object can be used to show information about that particular object, but will show nothing about the communication with any other objects on the network.

What is needed is a tool that will gather information from all of the objects, as well as the network communication, whilst retaining the links to the object-oriented source.

Current tools for CORBA

Current tools for visualising CORBA applications vary in both their purpose and the manner in which they gather data. A problem with some of the current tools whose purpose is education/understanding is that they do not accommodate the end user, the end user must accommodate these tools. The often require the developer (I will use the word developer to talk about both novice and expert programmers using these tools, as not all of them are aimed at students) to modify their code to allow the tool to gather the necessary data.

One such tool is OBViouS (Oldengarm & van Halteren 1998) , which requires the addition of code at certain filter points. OBViouS is a tool that has system designers and developers as its target audience. The developer must edit their code in certain places to allow for the gathering of data. Some of the problems with this tool is that firstly, it is intrusive as it requires the modification of code. Secondly, the user must learn how to use the tool. They need to know where in the code they should add the extra code used to gather data. Thirdly, many of the objects such as the ORB, IDL compiler etc. are pre-compiled, it may not be possible to modify the source code. Even if it is possible to modify the ORB, doing so may not be desirable, as it is a large and complex piece of software, and to make modifications could take a great deal of time.

Another tool is VEDA (Miller 2000), which modifies the IDL compiler to allow the addition of code to the CORBA stubs and skeletons, the objects which act as proxies for the clients/servers. This means that each time it is used, the IDL generated code would need to be recompiled depending upon whether or not the CORBA-based application is being visualised. VEDA is aimed at students learning about CORBA. The information being visualised only relates to the remote communications, as methods in the client stubs and server skeletons are only invoked when communication with a remote object is required. Due to this, we can only gather information regarding the network communication between objects when remote requests are being made and replys are sent. We still cannot see any other information about the client and server objects, only details of the communications between them.

CSMonitor (Choi et al. 1998) uses interceptors to intercept messages being sent between the objects that make up the distributed application. CSMonitor does not require any modification to source code, howvere, like VEDA, CSMonitor only gathers infomration relating to the communication between objects. We still cannot see the inner workings of any of the objects.

A problem with the first two tools is that they change the sequence of execution of the applications, as there is now extra code that must be executed. Of all the tools, VEDA is the only one specifically aimed at students. The users of these tools (students and developers) are required to learn how to use the tool, and modify their code accordingly to allow data to be gathered and visualised. CSMonitor is different to both OBViouS and VEDA as no modification of source code is necessary. However, only

information relating to the communications between objects can be gathered. CORBAview can gather infomration about all aspects of the distributed application in a non-intrusive manner.

Data Gathering

CORBAview captures data using an application based on the JDI (Java Debug Interface) of the JPDA (Java Platform Debugger Architecture) called visCORBA. The JDI is a high level interface to support debugging of a JVM (Java Virtual Machine). Instead of debugging the JVM, we are using JDI to gather information about the CORBA-specific events that occur in the JVM. A second JVM is created to run the application, whilst the first JVM monitors the CORBA specific events the are taking place in the second JVM. This second (target) JVM is running the user specified program, whether it be the ORB, server or client. The key here is that visCORBA only monitors the events of the target JVM; it does not change the way these events run. The effect visCORBA has is that the target JVM, and consequently the program running on this JVM, run more slowly, compared to when they are not being monitored. Because this data is not being gathered for performance analysis, this slowing of the JVM has a minimal effect on the overall system.

Every event that occurs in this target JVM can be mirrored by the first JVM. Through this mirror interface we can have access to global JVM properties. We can decide exactly what is to be mirrored on this target JVM.

There are 4 main types of events that occur in the JVM :

- *Class Prepare Events :* Class Preparation is when the constructor of a class is invoked , and the fields have been initialised. Preparation only occurs once for every class.
- *Method Entry Events :* Method entry is when a method has been invoked, but before any code has been executed. Information regarding arguments and return types is gathered.
- *Method Exit Events* : A method exit is when all code in this method has been executed. Information regarding return values (if any) is gathered. These are the two types of events that occur most frequently. For every method entry, there is a method exit. General information regarding the class, method name and method type is gathered for both.
- Watchpoint Events : Watchpoints are set on all fields in a class when that class has been prepared. An access watchpoint event occurs when a field has been accessed. A modification watchpoint occurs once a field has been modified. When a modification watchpoint occurs information about the old and new values is gathered.

Filters are used to extract only the data that we require. The filters are used to *define* what packages we want to gather information about; they are not used to gather any information. This is done through the services provided by the request package of JDI. By default, we only gather information related to CORBA classes and methods. Filters are only used to gather information that is relevant to CORBA, they do not alter how these methods and classes are used and implemented. The sequence of execution of the target JVM is unaffected. The filters that the student can set are categoriesd based upon the CORBA class hierarchy.

Figure 1 shows the GUI for visCORBA. The student is required to specify the same arguments that they would give at the command line, along with the category of CORBA events that they wish to gather information about. Each category of events relates to a different aspect of CORBA. A student can gather as much or as little data as is required.

Fields in the visCORBA GUI:

- *Applications Package, Application Name* : Standard argument passed at the command line.
- *User Packages to monitor* : The student can specify particular packages that they have created that they wish to gather information from
- *ORBInitialPort, ORBInitialHost* : CORBA specific arguments that are passed at the command line

- *Choose Event Filters* : This is where the user specifies the filters for the categories of CORBA events about which they want to gather information. All categories or any combination of categories can be chosen.

The student can decide whether they wish to view the events as they occur by seeing a textual output, and they can specify where the events should be stored. The text output shows the sequence of event types that have occurred, such as class preparation, method entry etc, but it does not show any of the communication , as each invocation of visCORBA could be gathering data about objects that are each at different locations.

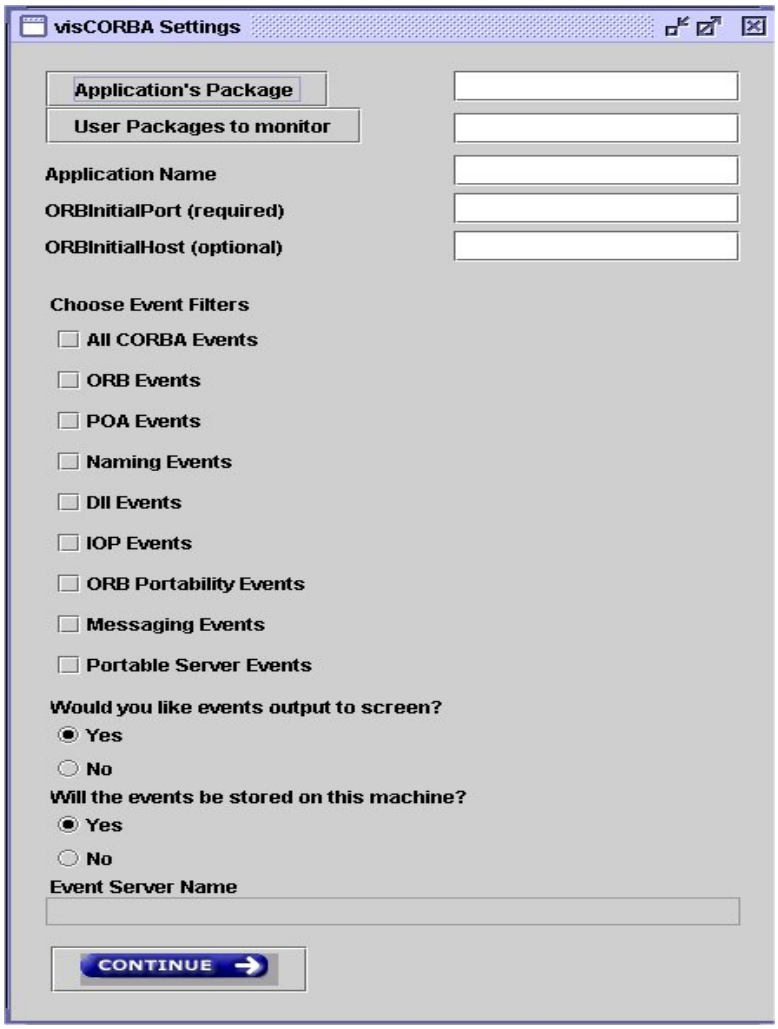

Figure 1 – visCORBA settings

Once the information for an event is gathered, it has to be stored somewhere. All of the event information is sent to a central point where it will be used for later visualisation. A central events server waits for connections from the various invocations of visCORBA. When it receives event information, it is stored in a file, along with information about the event time. The student specifies the location of the events server when they specify the program arguments and the filters to be set .

Because there may be may objects with the same name on the network, we must ensure that all information is stored correctly, and that information for each object is stored separately. A secure socket connection is set up for each invocation of visCORBA, so each invocation is connected to a different port. By doing this, we can ensure that information about the different objects is kept separately from each other. When information regarding the various objects has been gathered, we can now produce a visualisation of this information.

How do we visualise the Objects?

The visualisation does not use any novel or new approach in presenting a computer network. The same principles used in the data gathering part of the tool have been applied to the visualisation part of the tool. If the student must learn how to understand the visualisation technique being employed, then the tool has already, at least partly, failed (Ernst & Storey 2003). The goal is to teach the student about CORBA, not show them a new and innovative visualisation technique.

The main concept that has been applied is to show the user a general overview at first, and then allow them to focus on specific details (Shneiderman 1997). A visualisation tool that shows the student every detail at once can be too confusing if they cannot filter the information they deem to be unnecessary.

What information is deemed necessary or unnecessary should be a decision made by the student. They should be able to see as much or as little information as they see fit.

We show a general overview first, and allow the user to zoom in and focus on more specific details. The objects we are visualising can be located at many different points on a network. So we need to show the network topology of the objects, and the communications between them. But we also want to show the sequence of events that occur in the execution of the different objects. Therefore we need to show an overview of the network and communications, and allow the user to then focus on the sequence of execution of the various objects.

As a communication occurs, a line is drawn between the source and destination objects. The student can click on any of these lines and see the details of the communication, such as the method that has been invoked, and the data that is being passed over the network.

When a student wishes to see more specific details regarding a particular object, they can zoom in to reveal the details of the object and the sequence of events. These events are not shown all at once, they appear at regular intervals, so the user can follow the sequence of events at their own pace (how they control the pace is discussed in the next section). When the user chooses a particular event, a tree with details of that event is expanded. Figure 2 shows the details of a particular event that has been chosen. Once the student is finished viewing details about a particular event, they can revert to the network overview, and choose another object.

Control of events

When the user first specified the object to be monitored, they were also able to specify filters relating to what category of events they wished to gather information about. But when they are visualising the objects, they may want to run the visualisation many times, each time only looking at certain types of events. Or they may decide that they now want to see less information than they actually gathered. Whatever their requirements, at any point during the visualisation, the student can specify the category of CORBA events, and what JDI event types they wish to view.

The filters can be applied globally to all objects, or separate filters can be applied to each of the objects individually, it is totally up to the student. Figure 3 shows the filters being applied to all objects. These filters can be set before the visualisation occurs, so they can run the visualisation as many times as they wish, each time concentrating on a different aspect(s) of CORBA, without having to gather the data all over again. The filters that can be set for the visulisation are dependant upon the filters that have been set when the data was gathered.

The student can also control the speed at which the visualisation occurs. A slider is provided to allow the user to dynamically alter the speed, allowing them to speed up or slow down the visualisation by controlling the number of events that are shown per minute. The student can also pause the visualisation at any time. This can be useful if you want to concentrate on events that have occurred, before seeing the events that are to follow.

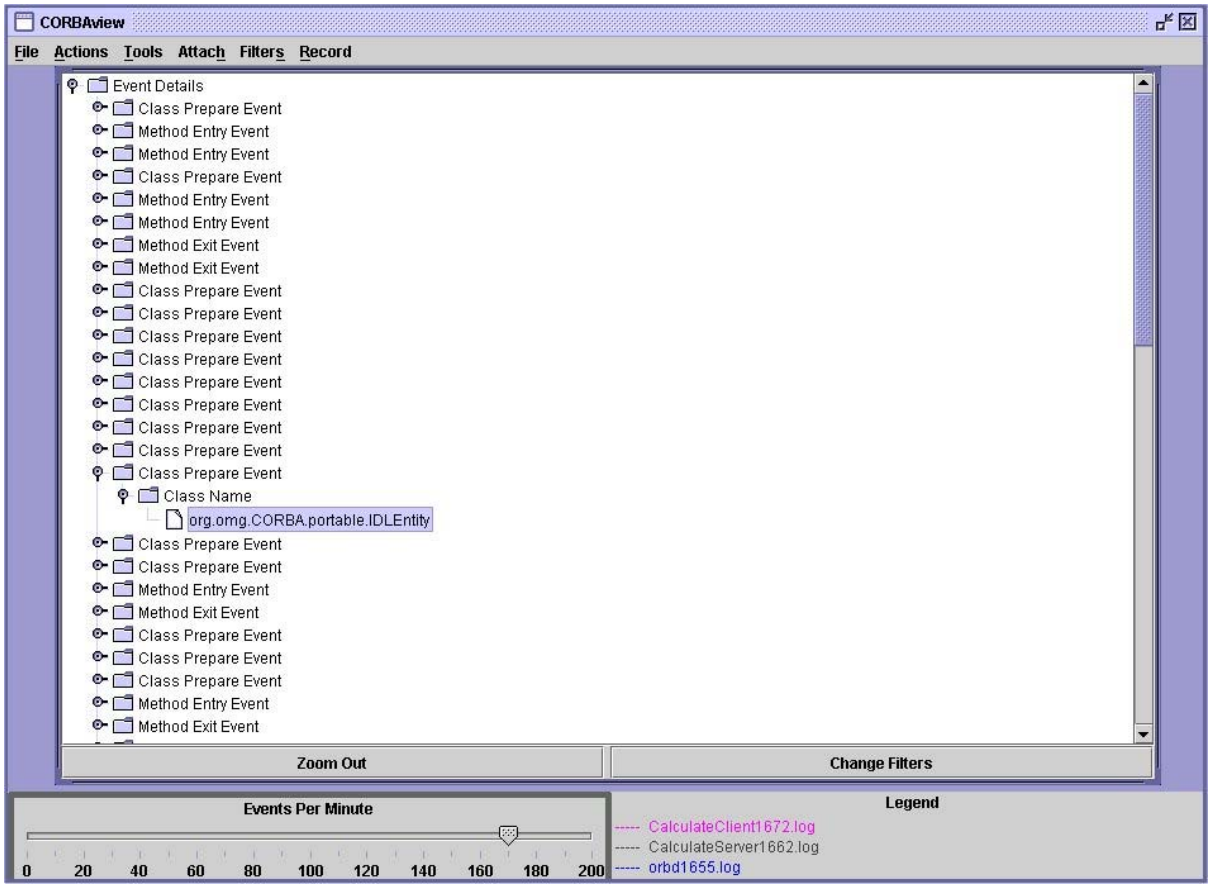

Figure 2 – Event Details

Repetition of Events

The student is able to see the sequence of events of all the objects from their creation to their destruction. But if the user is only interested in a particular sequence of events, then having to visualise everything that has come before can be time consuming and annoying. Perhaps the events that the student wishes to concentrate on deal with a request made by a client and the response form the server. Before a request can be made, the client(s) and server(s) must connect to the ORB, and before that, the ORB itself must be initialised. The student would have to wait for all of these events to be visualised before they could see the events that are of interest to them.

Recording Events : To accommodate the student in such a situation, when CORBAview is visualising the objects, at any point the student can tell the tool to start recording. When this happens, the tool records the point at which the user decided to start recording, and it records the moment at which they decided to stop. This information is then stored, so that the next time the user wants to visualise the objects, they are given a list of any recordings they have made. If the student picks one of these recordings, the objects are visualised as normal, but the student is only shown events that occurred between the time they started recording and when they stopped.

The student can still set filters and control the speed at which the visualisation occurs. Regardless of the filters that were set at the time of recording, all of the events that occurred are recorded, so the user is able to set different filters each time they open one of these recordings.

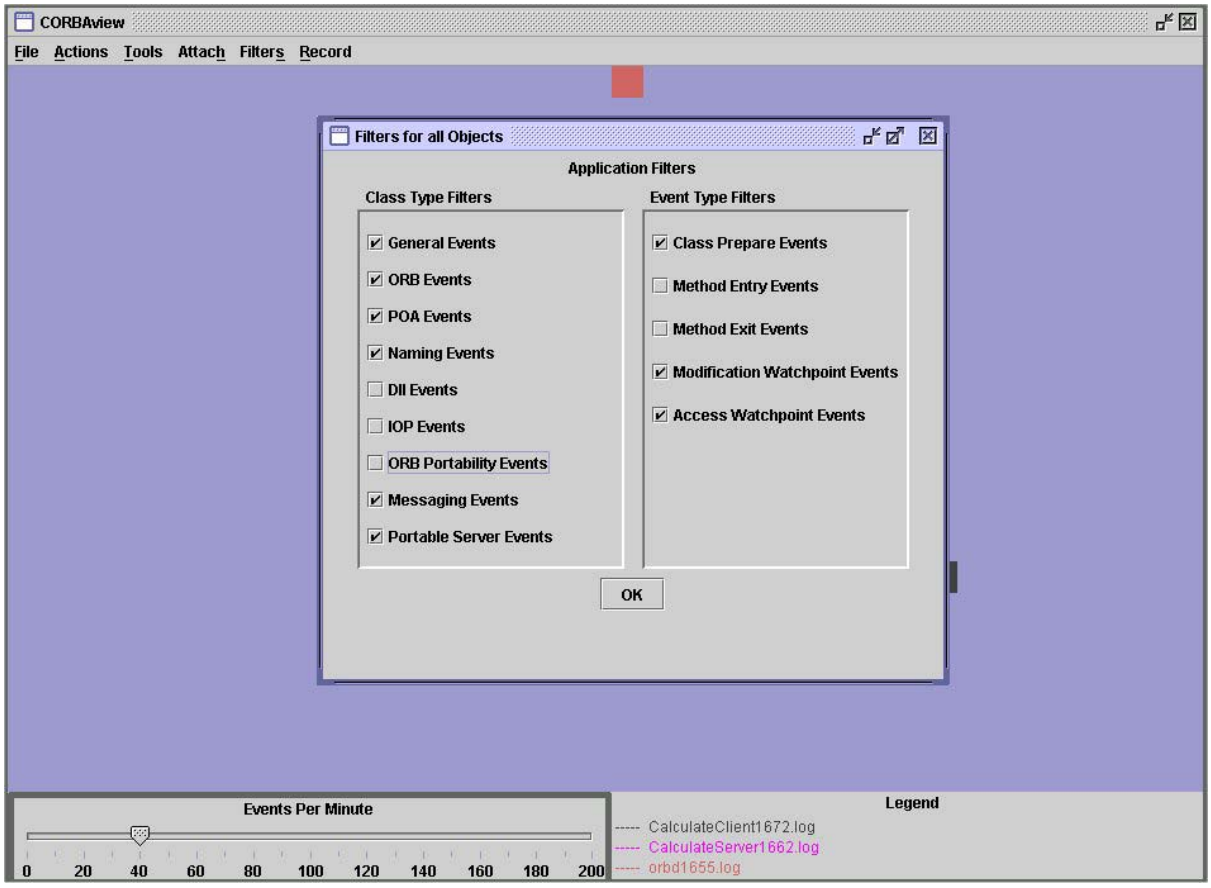

Figure 3 – Setting Filters

Maintaining links to the source and documenation

One of the advantages of Java is that the user can download the source code for all of the Java classes and methods, as well as having an on-line API (Application Programmer Interface), which can also be downloaded. By having the source code available, the developer can see how any method or class has been programmed, and they can also see how particular methods handle the functions that they provide. The API gives a description of the functions of classes and methods, and it also provides information about any related classes or methods.

The student can choose to attach the downloaded API or Java source code. They tell CORBAview where the source code or API can be located. The student now has the option to open the source code, or appropriate API web page when they have choosen to view details about a particular event. By attaching the source code, the student can see the sequence of events that have occurred, and go directly to the source code that is involved. If they choose to go to the API , they can read a description of that particular class or method.

Using these options the user can directly see the implementation of a particular class/method, or read a more detailed description of what this particular class/method is used for.

Conclusions and Further Work

The major goal of CORBAview is to create a tool capable of providing an understanding of how CORRA works by allowing users to record and visualise the inner workings and remote communications of the objects that combine to form a CORBA-based application. CORBAview allows the student to manipulate the visualisation by allowing them to show as much or as little information as they require using filters. The visualisation can be run a limitless number of times, each time with the possibility of different filters being set.

The student can record the events that take place at a certain time, allowing them to visualise these events at a later time, without having to view the events that occurred before or after them.

Further work to be carried out includes usabiltiy studies. The aim is to use students as they are learning how to create CORBA-based applications, and allow them to use CORBAview as well as having the standard methods of lectures/tutorials.

References

Baecker, R. (1981), With the assistance of Dave Sherman. *Sorting out Sorting*, 30 minute colour sound film, Dynamic Graphics Project, University of Toronto, 1981.

Choi, C.H., Choi, M.G., Kim, S.D. (1998), CSMonitor: a visual client/server monitor for corba-based distributed applications. *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC '98*), Taipei, Taiwan, Dec 1-4, 1998, pp. 338-- 345.

Cordes, D., Parrish, A.(1993), An incremental approach to software engineering in a science-based computing curriculum. *ACM Conference on Computer Science* 1993: 182-188

Eisenstadt M., Price B. and Domingue J. (1993), Software visualisation as a pedagogical tool, *Instructional Science*, Vol.21, 1993, pp.335-364

Ernst, N. A., Storey, M-A, (2003) A preliminary analysis of visualization requirements in knowledge engineering tools. University of Victoria, Victoria, CHISEL Technical Report August 19, 2003

Miller, R (2000)., A toolkit for the visualisation of CORBA applications. MSc. Thesis, Trinity College Dublin, 2000

Nørmark, K., Andersen, M., Christensen, C., Kumar V., Staun-Pedersen, S. and Sørensen, K. (2000) Elucidative programming in Java. In *Proceedings on the Eighteenth Annual International Conference on Computer Documentation* (SIGDOC). ACM, September 2000.

Object Management Group, (2002). *The Common Object Request Broker Architecture and Specification*, Revision 3.0, OMG Document 02.06.01, 2002

Oldengarm, P, van Halteren, A. (1998), A multiview visualisation architecture for open distributed systems. *22nd International Computer Software and Applications Conference*, August 19-21, 1998, Vienna, Austria. IEEE Computer Society 1998, ISBN 0-8186-8585-9

Orfali , R., Harkey, D. (1998), *Client/Server Programming with Java and CORBA*, Second Edition, Wiley Computer Publishing, ISBN 0-471-24578-X.

Rodger, S. (1995) An interactive lecture approach to teaching computer science, SIGCSEB: *SIGCSE Bulletin (ACM Special Interest Group on Computer Science Education*), 27, 1995.

Shneiderman, B., (1997) A grander goal: a thousand-fold increase in human capabilities. *Educom Review*, 32, 6(Nov/Dec 1997), 4-10.