High Performance Java
JTAP/JISC Report

David Walker and Omer Rana
Department of Computer Science,
University of Wales - Cardiff,
POBox 916, Cardiff CF2 3XF, Wales

Contents

List of Figures

1 Introduction

The aims of this project were to establish a collaboration with Professor Geoffrey Fox at the *Northeast Parallel Applications Center (NPAC) at Syracuse University, New York*, into the use of Java in high performance computing. We now intend to disseminate the results of this collaboration to the wider UK research community over the summer of this year. We first identify the significance of the term *High Performance (HP) Java*, and then identify software tools that may be employed by other research groups within the UK. This is a preliminary report, and outlines the work done so far. A further report after the HP Java conference will be made available towards the end of summer.

Over the last few years, Java has become an important component within a Web based environment. This coupled with its simplicity, object-orientation, portability and multi-threading, makes it an ideal language for the development of parallel applications. In this report we highlight tools developed at NPAC which could impact work within the parallel and scientific computing research community.

2 HP Java Tools

NPAC has been involved in the development of various tools for Java oriented applications. Such tools fall into three general categories:

1. Tools which allow conversion of existing parallel or scientific codes to Java, either as cross compilers converting Fortran code to Java, for instance, or providing additional sets of Java packages which augment the functionality of existing parallel libraries: such as Java wrappers for MPI, or the Java Numerical Libraries.
2. Visualisation tools which allow multiple users to interact using Web interfaces, such as *Tango*. Also, visualisation tools which allow users to run applications across a network using client-server interactions Web based interfaces, such as *SciVis*.
3. Extensions to the Java programming language to provide support for parallel programming. The intention being to capture data structures which exist within traditional high performance languages, like Fortran, and extend these to Java, such as the *Java-Ad* environment developed by PCRC[1] members.

2.1 Tango

Tango provides a framework for a number of different Internet based services to be integrated into a single environment. Since the Web allows interoperability

[1] Parallel Compiler Runtime Consortium

across various platforms, a collaborative framework based on Java was found to be ideal for performing experiments by many different types of users. The main *Tango* window is illustrated in figure 1.

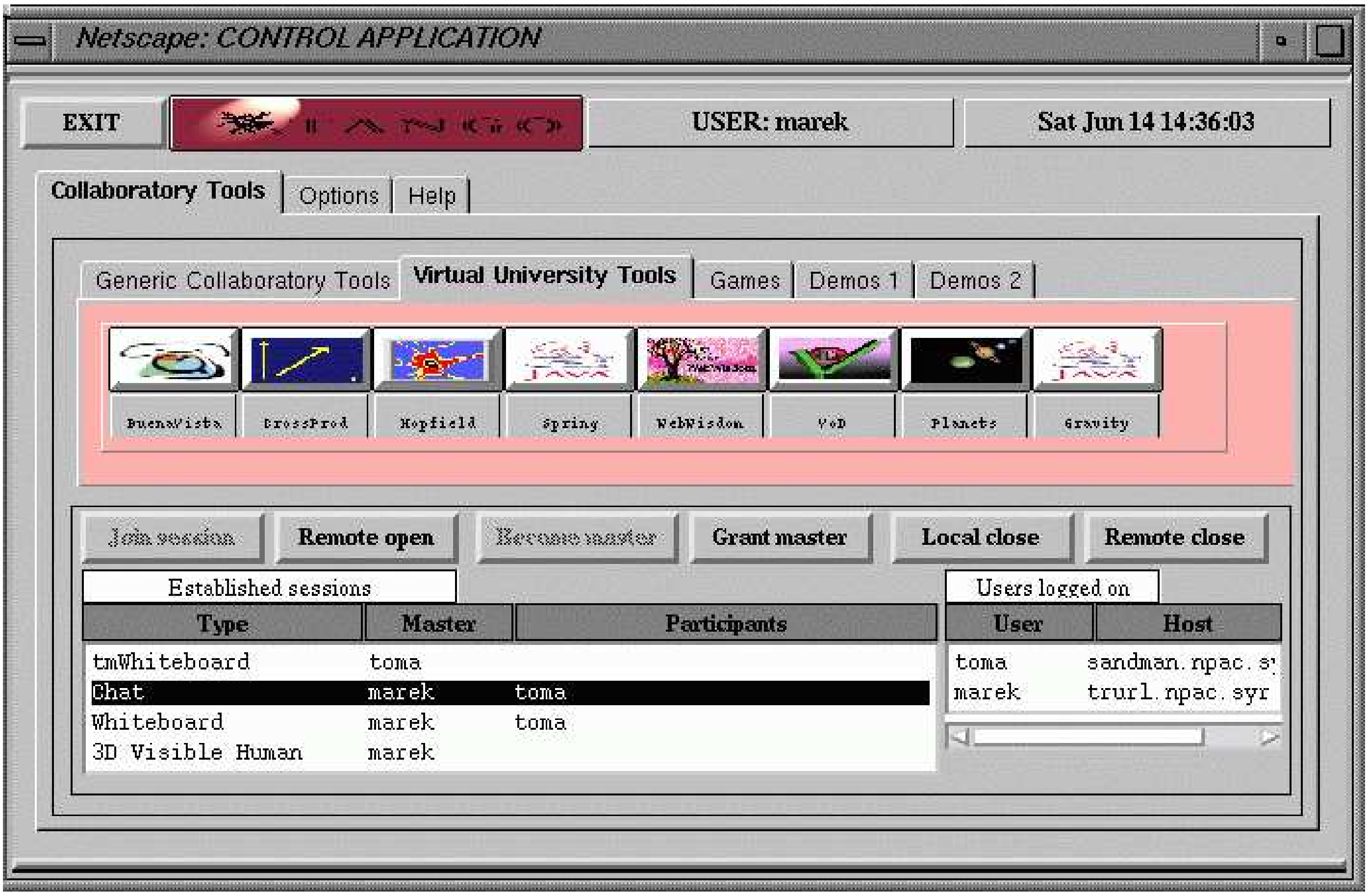

Figure 1: *Tango Main Screen*

Tango allows various users to collaborate using *chat* and *simple chat* applets illustrated in figure 2 and 3, which are similar to the *Internet Relay Chat* (IRC) clients generally available. The chat clients are however provided as an integrated tool set to support *Computer Supported Collaborative Work* (CSCW) oriented environments. Other tools for collaboration include a *Whiteboard* as shown in figure 4, and applets for sharing information, as in figure 5, where the shared information may be graphical and allow for user interactivity. A user may run applications developed as applets and do so across the internet, provided a *plug-in* component is available to support the application program,

Figure 2: *Tango Simple Chat*

and one which may be used with a Web browser such as *Netscape*. The user may configure the environment to connect to particular servers as illustrated in figure 6, allowing a server to be identified by a URL reference.

The general philosophy behind *Tango* is illustrated in figure 7, and the concept of using specialised plug-in components in figure 8.

2.2 Fortran to Java

The Fortran to Java *(f2j)* utility provides a means to convert a Fortran program to its Java equivalent. The user specifies the Fortran 77 source and the program generates the equivalent Java class. Consider the Fortran code sample:

Figure 3: *Tango Chat*

```
      program foo
      integer a(1:10)
      integer b(1:10,-10:10)
      integer c(1:10,-10:10,-10:0)
      integer i,j,k

      do 10 i=1,10
      a(i)=10 - i
      do 10 j=10,-10,-1
      b(i,j) = i + j
      do 10 k = -10,0,2
10    c(i,j,k) = i + j - k * i / j
      end
```

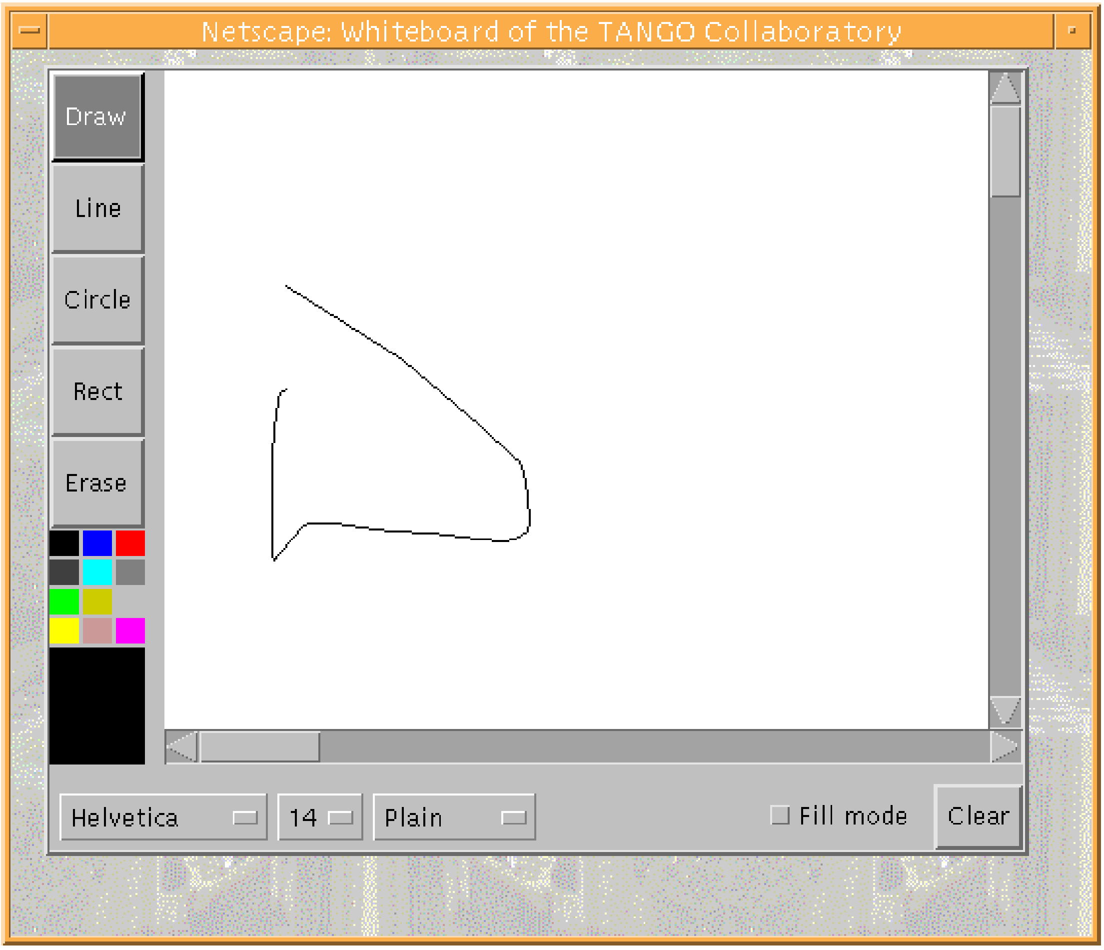

Figure 4: *Tango Whiteboard*

the following Java program is generated:

```
class foo_mc  {
     public static  void main(String args[])
    {
      int a[] = new int [10-(1)+1] ;
      int b[][] = new int [10-(1)+1][10-(-10)+1] ;
      int c[][][] = new int [10-(1)+1][10-(-10)+1][0-(-10)+1] ;
      int i,j,k ;
      for (i=1;i<=10;i=i+1)
      {
        a[i-(1)]=10-i ;
        for (j=10;j>=-10;j=j-1)
        {
```

```
            b[i-(1)][j-(-10)]=i+j ;
            for (k=-10;k<=0;k=k+2)
j10    :              c[i-(1)][j-(-10)][k-(-10)]=i+j-k*i/j ;
          }
        }
      }
  }
```

This example demonstrates the use of multi-dimensional arrays in Fortran, and how the cross compiler deals with these in Java, which is not very strong in its support for multi-dimensional arrays. The next example illustrates the use of type casts, for combining different data types:

```
        program comp
        complex com1,com2
        com1 = (1.2,2.3) + (2.3,2.2) * (2.3,2.5)
        com2 = (1.0,1.0) / (1.0,1.0) * (1.0,1.0) - (4.3,3.4)
        com2 = com1
        end
```

the subsequent translation in Java is:

```
 class comp  {
      public static  void main(String args[])

     {
       Complex com1,com2 ;
       com1=((new Complex((float)1.2,
 (float)2.3))).add((new Complex((float)2.3,
 (float)2.2)).mult((float)2.3,(float)2.5)) ;
         com2=((new Complex((float)1.0,
 (float)1.0)).div((float)1.0,
    (float)1.0).mult((float)1.0,
         (float)1.0)).minus((new Complex((float)4.3,
 (float)3.4))) ;
       com2=((Complex)com1.clone()) ;
     }
 }
```

Plans are underway to provide the *f2j* facility on the Web, to allow users to submit their Fortran 77 source file via the Web for translation to Java.

2.3 SciVis: Scientific Visualisation

SciVis allows data visualisation and analysis over a network using client-server based interaction in Java. Users may view different types of data, such as 1D, 2D, 3D and vectors using the SciVis visualisation tool, also allowing the user

to perform animations from the data sets if desired. The data analysis is performed with user definable *filters*, which act on particular data sets provided locally or referred to across the Internet. SciVis allows collaborative filters to be provided by different users on a system, or the user may design their own filter using special APIs provided.
As SciVis is based on Java sockets, it may be used as a front end to a range of different applications that support sockets, and where the users provide their own data sets. The user may thus write the data as a stream to a named socket, allowing SciVis to read this data by connecting to this socket. SciVis also provides a collaborative environment where user can exchange their particular data sets or their personalised filters. SciVis allows for *master-slave* or *master-master* interaction, allowing a single visualisation tool to interact with many different servers generating data sets.

Figure 9 shows the main *SciVis* server window identifying the possible options available to a user. Figure 10 shows the *Collaboration Monitor*, which allows various user to interact via a text based interface (similar to the chat environment discussed in *Tango*), except that the user may also visualise and share data (as shown in figure 11 for 2D visualisation and figure 13 for 3D visualisation). The user may either specify the available filters from a menu, or change parameters associated with the filter, or the user may select the *Editor* from the main *SciVis* server window (figure 9) to implement a new filter as a Java program, which allows a user to specify the functionality of a filter by extending the main *Filter* class. The drawing tool identified in figure 12 may be used to mark particular regions of interest on generated graphs, for multi-user collaboration.

2.4 WebFlow/WebVM

WebVM (Web Virtual Machine) provides a model to extend the capability of Web servers by a collection of CGI (Common Gateway Interface) modules, provided as part of the WebVM distribution. The CGI modules work in a client-server fashion, where the server receives a set of arguments from the client, performs some computation at the server, and returns the result as a MIME object to the client. With WebVM, the client input is specified as a URL of a MIME object, from where it is fetched and sent to the server domain, a suitable transform or filter is applied to it, and the resulting MIME object sent back to the client. Such passing of MIME objects by their URL, could allow for the conversion of image formats such as AutoCAD DXF to VRML, remote image editing or remote VRML authoring kits.

The overall architecture of WebVM is provided in figure 14, where the computational paradigm is centered around particular dataflows, with user CGI processes acting as computational nodes/modules, and with system CGI processes providing the required system control and management functions.

Each CGI module can perform I/O operations on local disks, with the transfer of MIME objects between servers implemented using the HTTP protocol and carried out by the system CGI module called the *Flow Executive*. Hence, as illustrated in figure 14, module M_1 creates an output MIME object $O1^{(1)}$ from server S_1. This output object is transferred to server S_2, where it is stored as input object $I_1^{(2)}$, ready to be used by module M_2.

The transfer commences after M_1 has finished its computation and written $O_1^{(1)}$ to local disk. Following the write, M_1 opens a connection to server S_2 sending a POST message to the *Flow Executive* F_2, passing the URL of the stored object $O_1^{(1)}$. F_2 subsequently issues a GET message to server S_1, retrieves $O_1^{(1)}$, and stores it as $I_1^{(2)}$. F_2 then starts the module M_2 running in the background and exits. At this stage M_1 can also exit and all connections between servers S_1 and S_2 may be dropped.

Module M_2 reads the input object $I_1^{(2)}$, performs local computations on it, and writes it as an output objects $O1^{(2)}$, $O2^{(2)}$ (generally as files), and notifies the *Flow Executive* processes in the destination servers of these objects.
In this example M_1 and M_2 are neighbouring nodes, whereas this standard structure can be replicated to form a much larger computational graph called *Compute-Web*. Web servers are therefore used as the general infrastructure within such a system to allow modules from several Compute-Webs, belonging to different users, to interoperate. A server may be locally configured, with the initial state of the WebVM being Web servers acting as network daemons. Dataflow within the system is initiated by a browser which activates a module, e.g. M_1 above by calling its URL, and any subsequent modules at the server are activated whenever the state of their input object changes, and the F processes are activated upon by each object transfer request, issued by the M processes.

The *Flow Executive* module acts as a single management module, however future versions are to contain further modules allowing for more complex cases, such as the ability to update configuration files at a node, to support Compute-Web editing through a *Link Manager* system module etc.

2.5 Java MPI wrappers

The *Java MPI wrappers* provides an Java language interface for MPI programs, which cater for the design of objects, their methods and the ability to integrate native methods written in C/C++ (for instance) into Java. A set of Java classes based on fundamental MPI objects are provided, such as *groups, communicators etc*, generating a one-to-one mapping between MPI calls and their equivalent Java wrapper bindings. Standard MPI primitives are provided as a Java package, with classes such as:

- public class MPI;

- public class Comm;
- public class Group;
- public class Datatype;
- public class Op;
- public class Status;
- public class Request;
- public class Errhandler;

Consider the following example program in Java, which makes calls to the MPI Java packages:

```
Example1.java :
import MPI.*;
public class Example1 {
  static public void main(String[] args) {
    MPI JMPI = new MPI(args);
    int myid = JMPI.COMM_WORLD.Rank();
    int numprocs = JMPI.COMM_WORLD.Size();
    System.out.println("Process "+myid+"/"+numprocs+
      " on "+JMPI.Get_processor_name());
    JMPI.finalize();
  }
}
```

The statement:

```
MPI JMPI = new MPI(args);
```

will create an MPI class instance called JMPI. The constructor of the MPI class will subsequently transform the String[] arguments into C style string array reference, call MPI_Init and create a communicator COMM_WORLD, and finally create default MPI reduce operators MIN, MAX, SUM, * etc.

2.6 HPJava

HPJava is a framework to extend the *Java* programming language to provide support for scientific parallel programming. It is intended as a framework which would combine tools, class libraries and language extensions to support parallel processing paradigms such as *shared memory programming, message passing and array-parallel programming.* Once such a framework is in place, bindings to higher level libraries and application specific libraries such as *CHAOS*, *ScaLAPACK*, *Global Arrays* or *DAGH* may also be developed.

The first step is to introduce characteristic ideas of other high performance languages, such as the distributed array model and array intrinsic functions and libraries of *HPF* into the Java programming language as extensions. The resulting programming model would be SPMD (Single Program Multiple Data), allowing direct calls to MPI (see section 2.5) or other communications packages from the HPJava program. Providing distributed arrays as language primitives would allow the programmer to simplify error-prone tasks such as converting between local and global array subscripts and determining which processor holds a particular element. The compiler for HPJava would make calls to a run-time library and generate underlying Java code. The translator is being implemented in a compiler construction framework developed in the *PCR Consortium* .

Within *HPJava* the PCRC run-time Kernel is referred to as *Java-Ad*, and implemented as a C++ library, and a hybrid of the SPMD and data parallel approaches. The following types of functionalities are provided in *Java-Ad*:

1. *Multidimensional Arrays:* Multidimensional arrays allow regular section subscripting, similar to Fortran 90 arrays. Such arrays are a new entity and coexist in the language with ordinary Java arrays, e.g.

   ```
   float [[,]] a = new int [[5,5]] ;
   int [[,,]] b = new int [[10,n,20]] ;
   ```

   An example of section subscripting is:

   ```
   int [[]] e = a [[2, 2 :]] ;
   ```

   where e becomes an alias for the 3rd row of element a. *Java-Ad*, unlike Fortran, does not however permit vectors of integers as subscripts.

2. *Distributed Arrays:* Distributed arrays may be viewed as coherent global entities, but their elements are divided across a set of cooperating processes. Distributed arrays are based on the concept of a *Process Array*, over which elements of a distributed array are scattered.

   For instance, a 2 by 2 process array can be defined as:

   ```
   Procs2 p = new Procs2(2,2) ;
   ```

   whereas a 6 element, one dimensional process array is:

   ```
   Procs1 q = new Procs1(6)
   ```

3. *Ranges:* A Range object defines a range of integer subscripts, and how such subscripts are to map into a process array dimension. Each value in the range is mapped to the process (or slice of processes) with that coordinate. Hence, a distributed range object may appear in place of an integer extent in the constructor of the array, as follows:

```
float [[#,#,]] a = new float [[x,y,100]] on p;
```

which defines a as a 2 by 2 by 100 array of floating point numbers. As the first two dimensions of the array are distributed ranges – the dimensions of $p - a$ is realised as four segments of 100 elements, one in each of the processes.

4. Other extensions include *BlockRange* which is a subclass of *Range* and describes a simple block distributed range of subscripts, *Distributed Parallel Loops* similar in concept to the *FORALL* construct of Fortran, *Subranges* and *Subgroups* which can be viewed as a slice of a process array, formed by restricting the process coordinates in one or more dimensions to single values. Hence, the active process group or the group over which an array is distributed may be just some slice of a complete process array.

The objective of PCRC within which *Java-Ad* is being implemented, is to:

> *Develop a public domain common runtime infrastructure supporting parallel compiler/interpreter development and general purpose high level SPMD programming for distributed-memory machines, and a Java empowered WebTop computing mechanism, which uses this infrastructure as underlying engine, for world wide collective computational solution for science and engineering.*

3 HP Java Workshop at Cardiff

The tools mentioned in this report are available at:

`http://www.cs.cf.ac.uk/User/O.F.Rana/hpjava/`

Further tools will be made available at this site as and when they are developed at Cardiff.

We plan to organise a workshop on high performance Java at Cardiff University during the summer this year. The workshop will be aimed at providing a focal point for various groups (academic and commercial) working on high performance Java within the UK. We also plan to provide a demonstration of various tools developed at NPAC (as outlined above) and tools developed locally at Cardiff for collaborative computing using Java. Such software is particularly beneficial to the high performance community, providing support for both parallel and numerical computation in Java.

4 Conclusion and extensions

NPAC is a dynamic organisation offering new and more sophisticated tools for scientific computing based on the Java programming language. Our initial contact with them has been very successful, and we will be utilising the techniques learnt from this experience within our *Problem Solving Environments* being investigated at Cardiff. We also plan to work with *Sun Microsystems* (Bristol) and find areas of overlap for possible future collaboration.

Some of the tools at *NPAC* are prototypes, and as such, restricted in functionality or scope. However, with the improvements in Java and the accessibility of other libraries and support environments, we intend to make future tools developed at Cardiff accessible to a wide set of users.

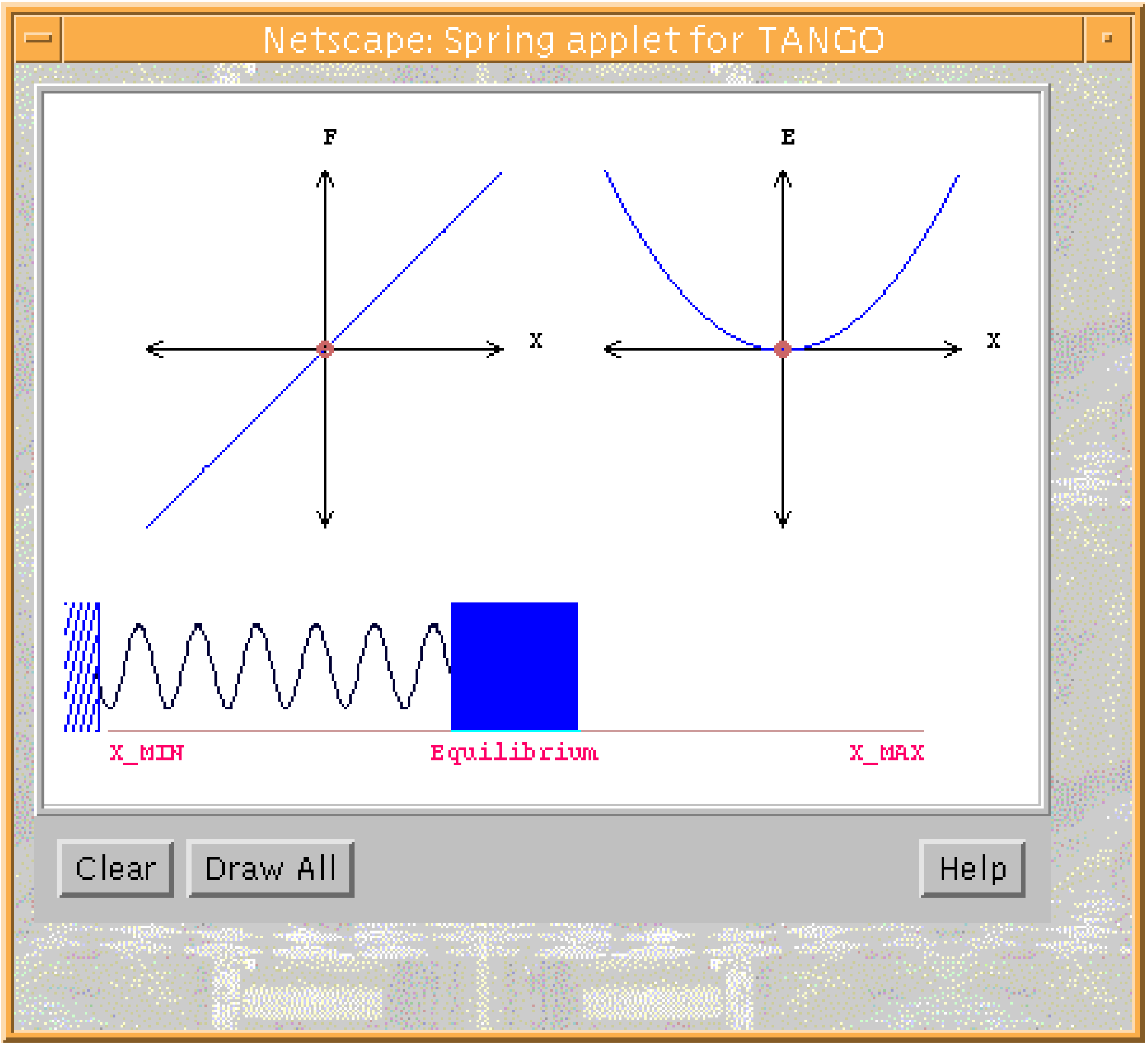

Figure 5: *Tango Springs*

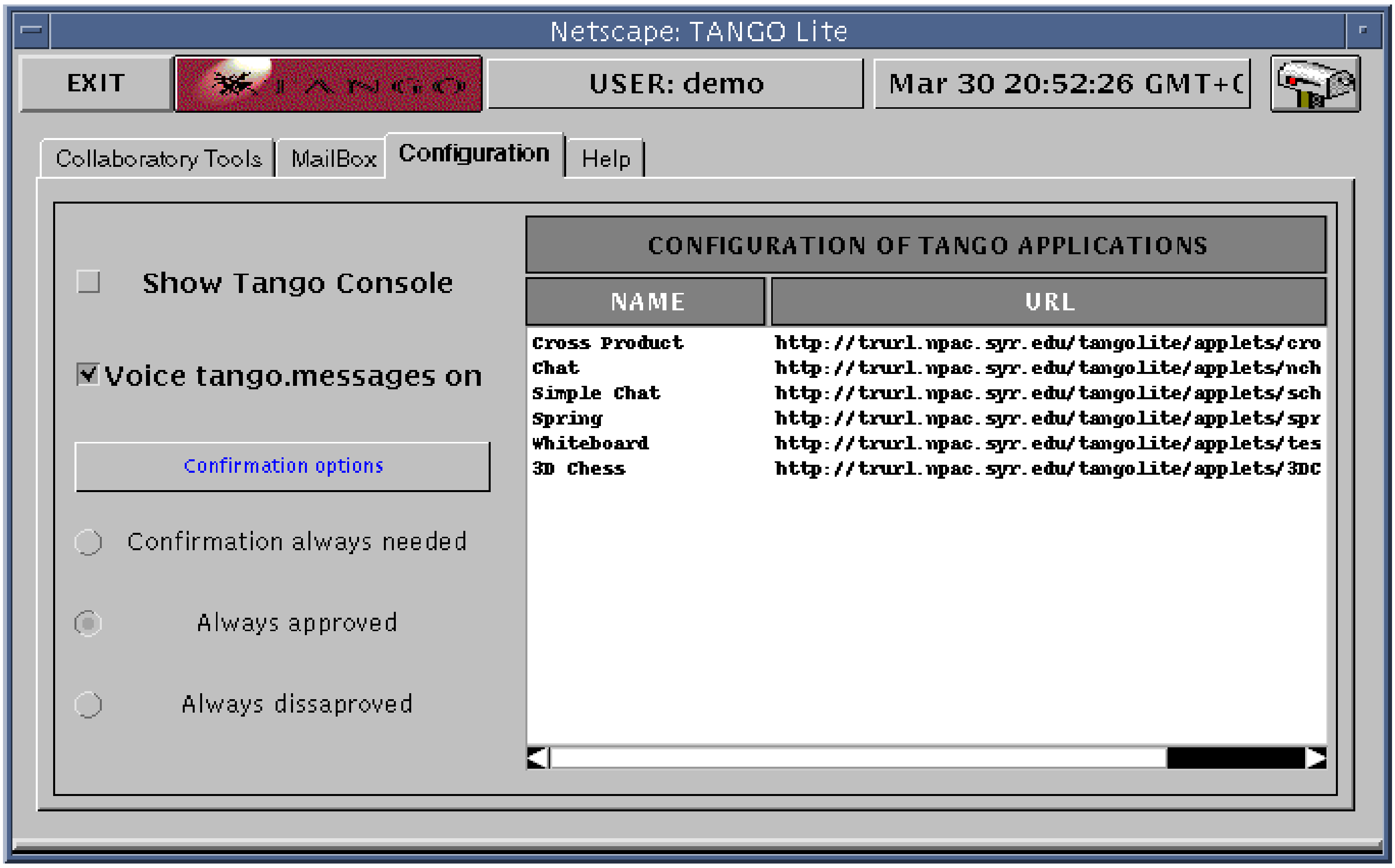

Figure 6: *Tango Configuration*

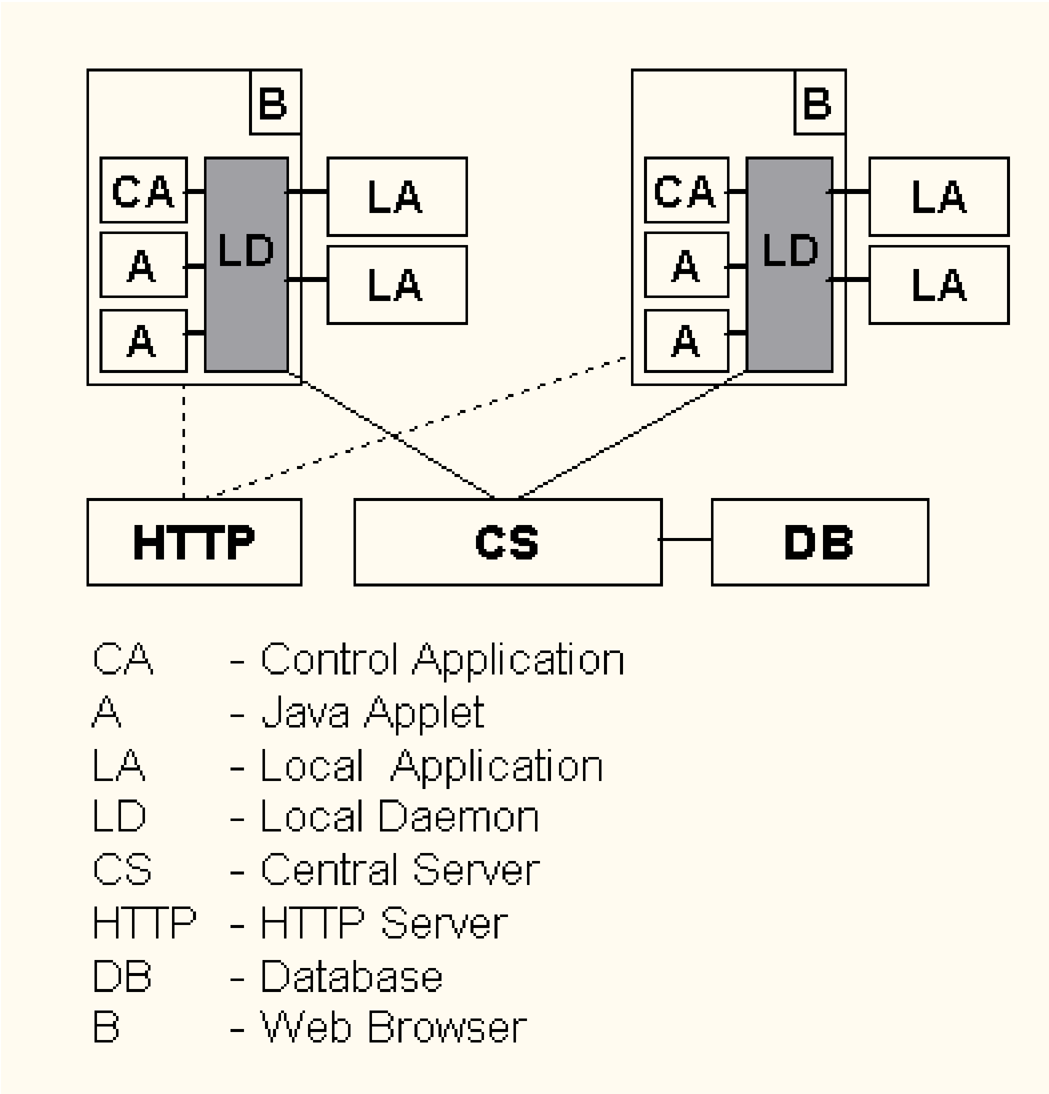

Figure 7: *Tango Architecture*

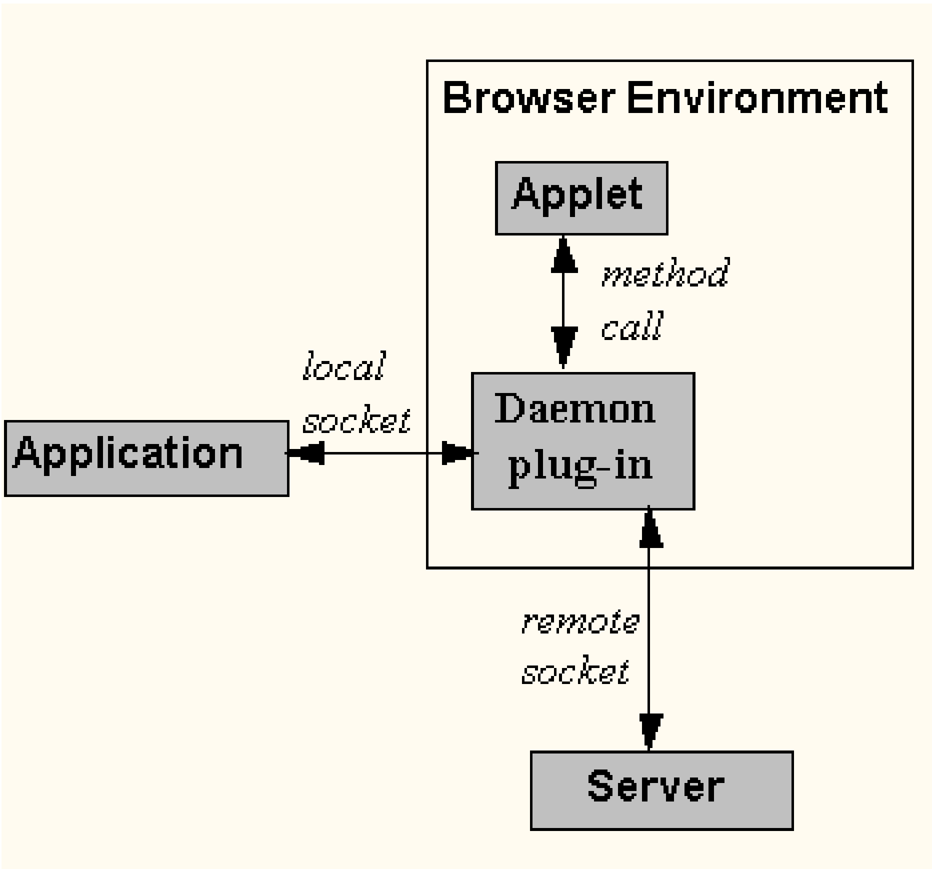

Figure 8: *Tango Plug-In Components*

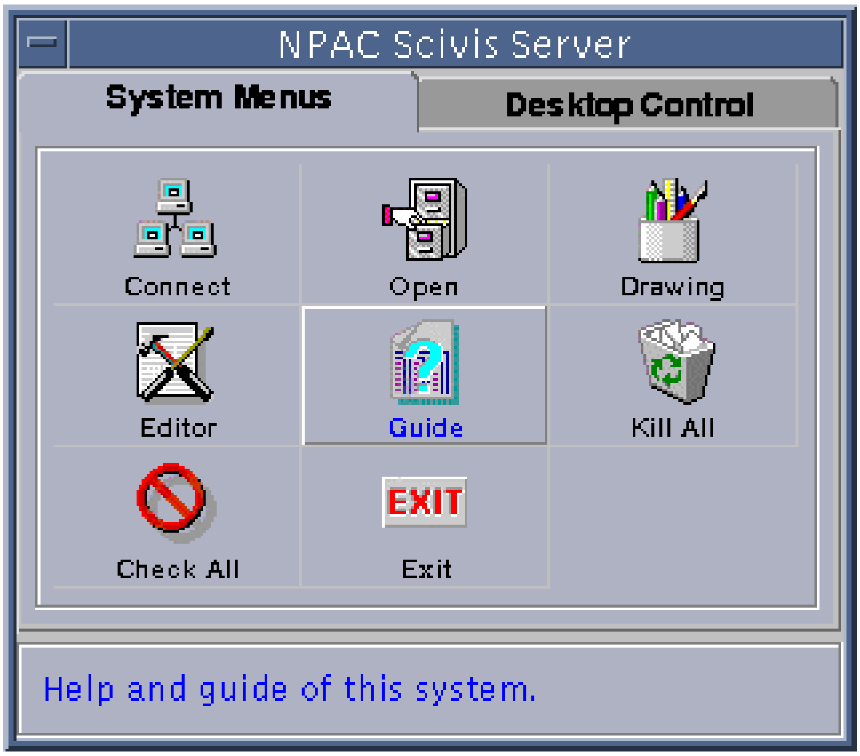

Figure 9: *SciVis main screen*

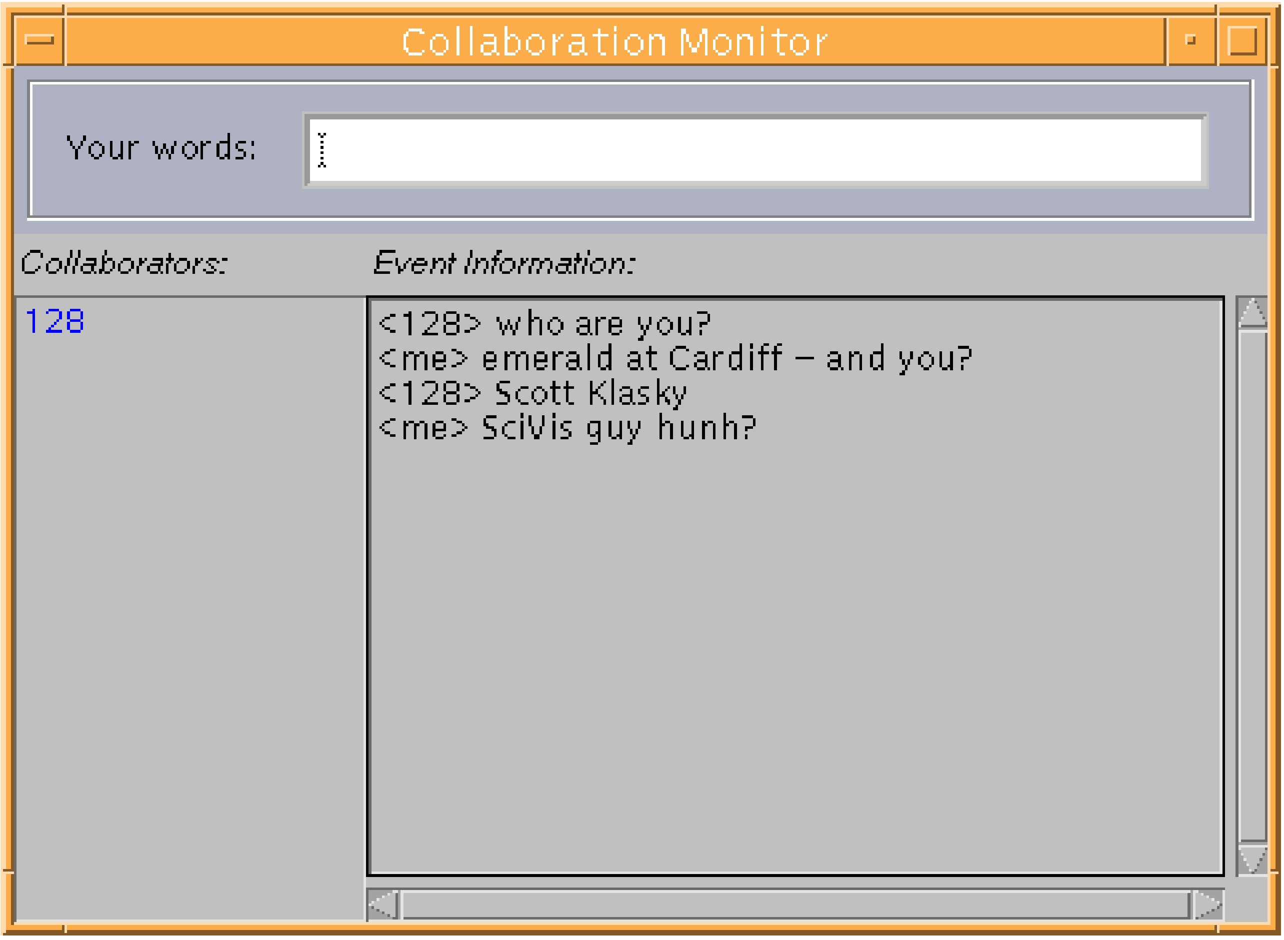

Figure 10: *SciVis Collaboration Monitor*

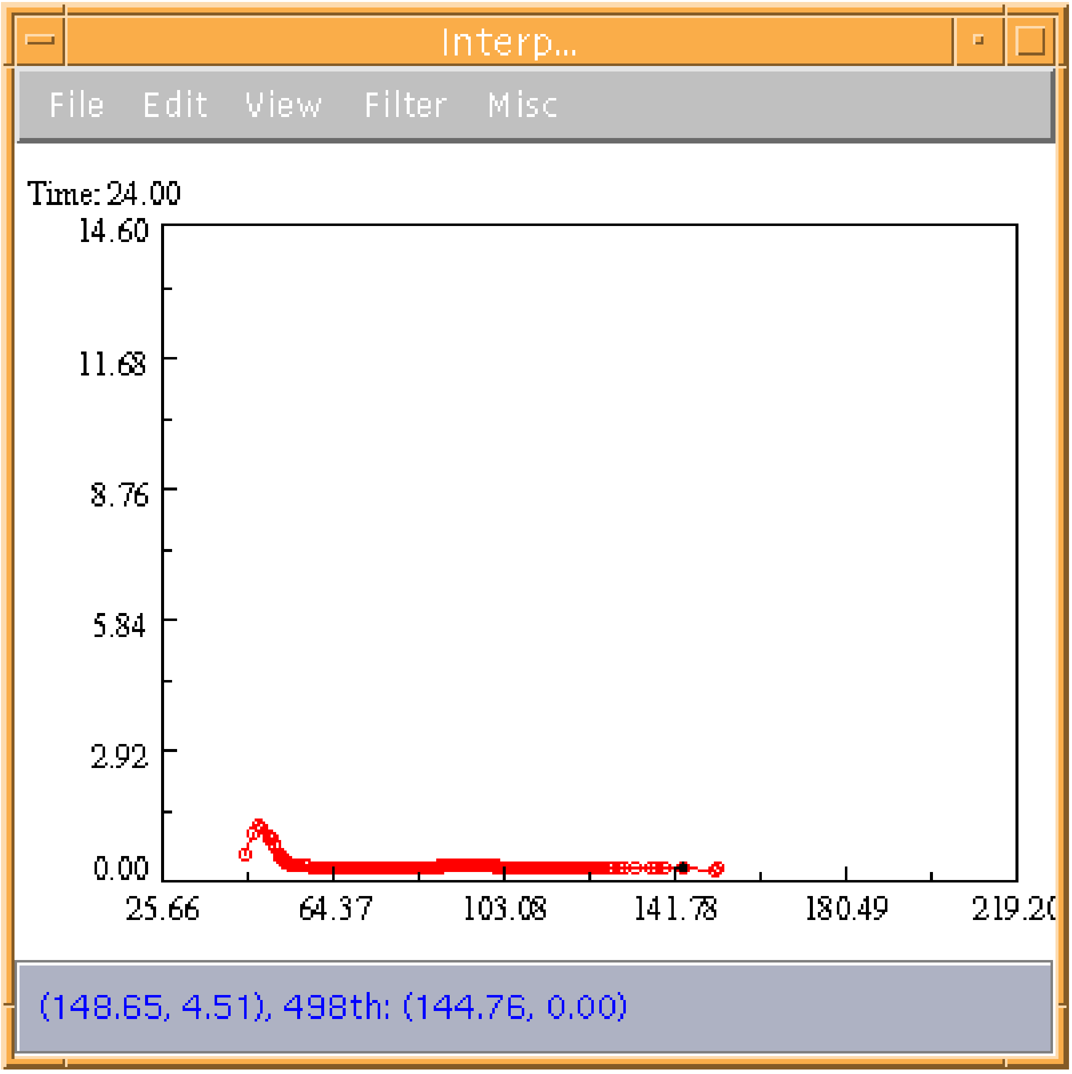

Figure 11: *SciVis 2D Data Visualisation*

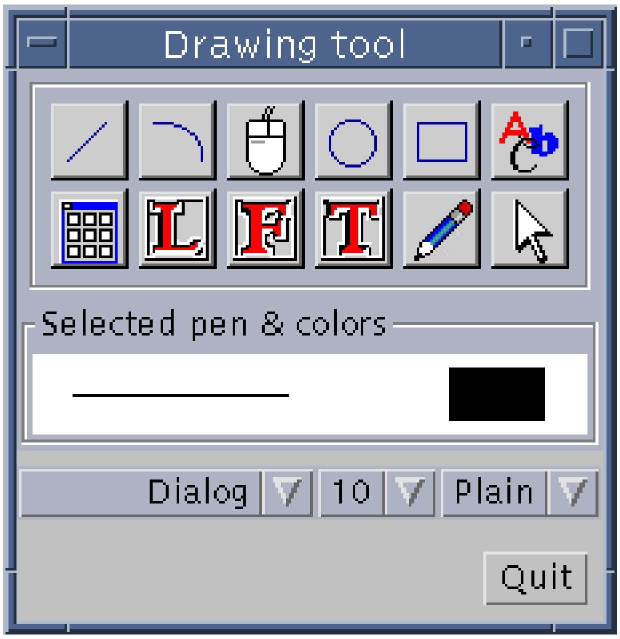

Figure 12: *SciVis Drawing Tool*

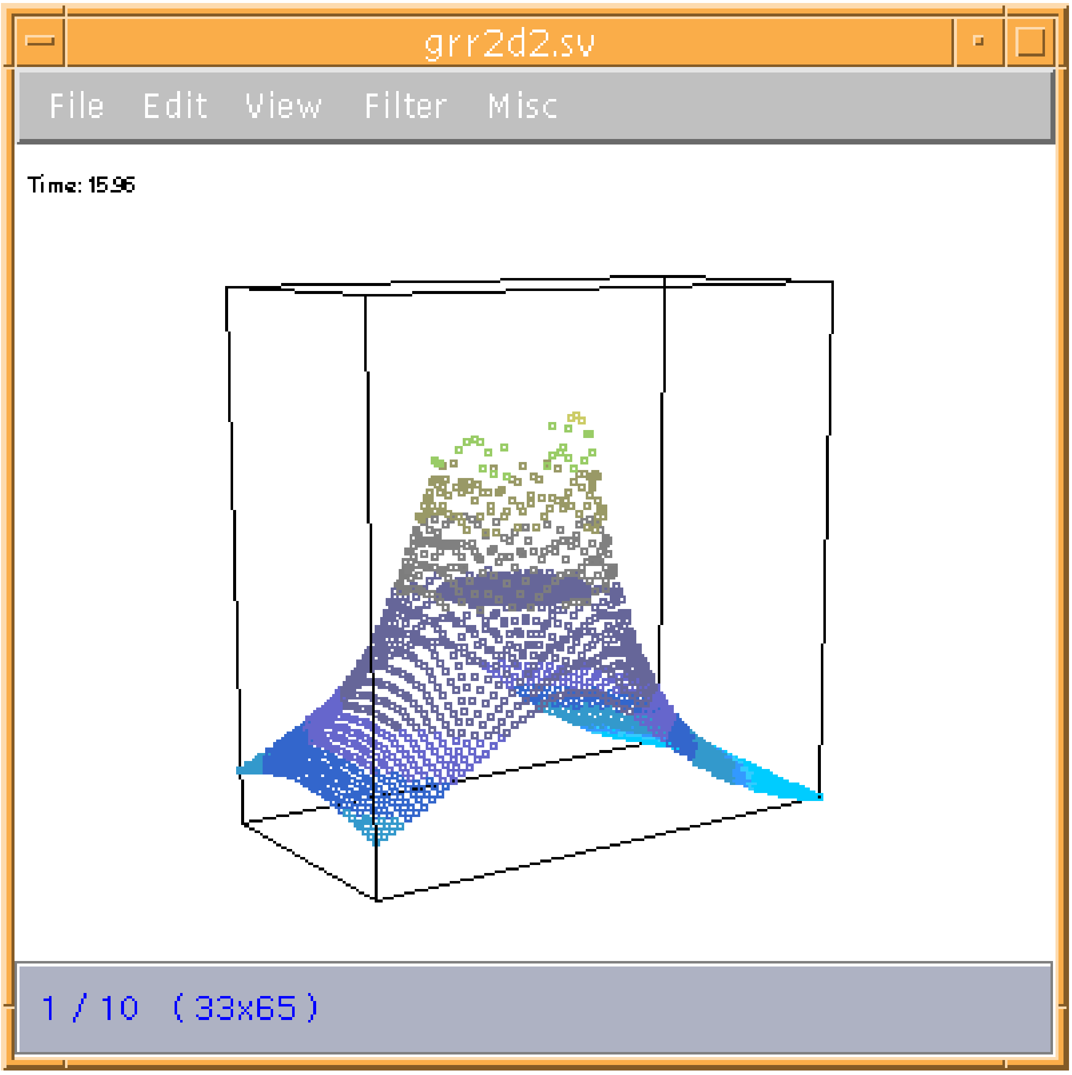

Figure 13: *SciVis 3D Data Visualisation*

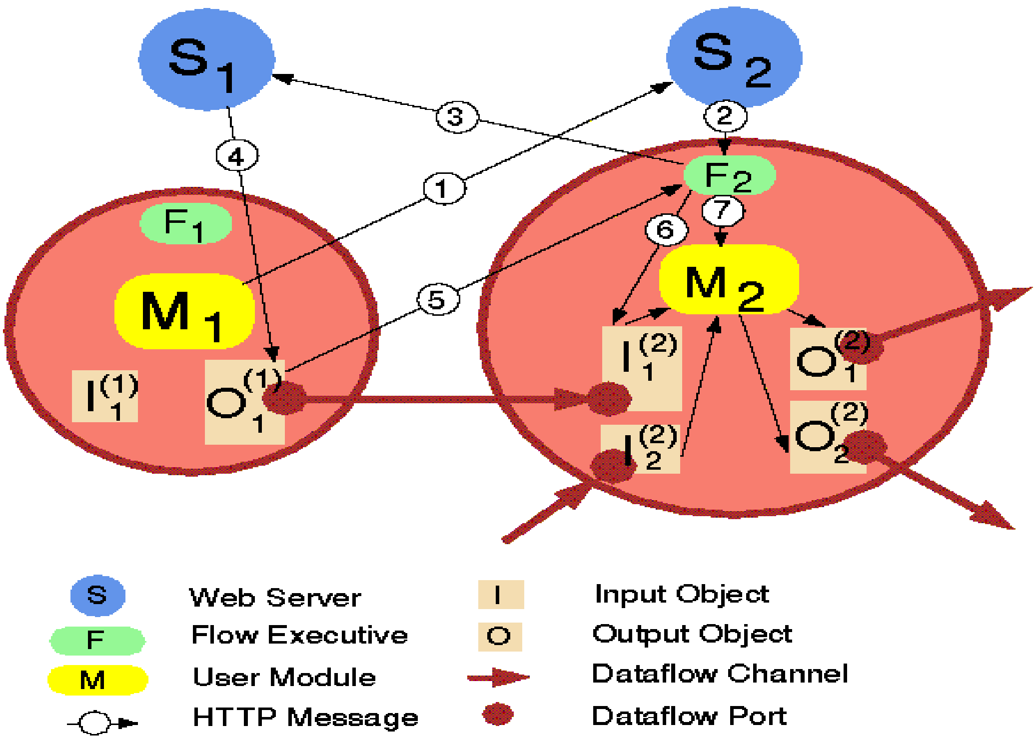

Figure 14: *WebVM*