Controlled by the Web

Alan Falconer Slater
Department of Computing and Electrical Engineering
Heriot-Watt University
Edinburgh
Scotland

afs@cee.hw.ac.uk

http://neptune.cee.hw.ac.uk/~afs/home-page.html

Abstract

A hypertext version of this paper is also available [1].

This paper provides a brief description of a software tool developed within the TLTP Interact Project [2] which allows application programs, concurrently executing within a Unix environment, to be controlled by scripts delivered to Mosaic from W3 servers. This tool was developed to allow the coupling of interactive simulations of scientific and engineering phenomena with courseware provided by W3. This software is known as the *Interact Communication Facility* (ICF).

The features of the ICF include:

- The ability to allow application programs to receive control messages and data from scripts embedded as links within HTML documents.
- The use of HTML fill-out forms to enter data intended for application programs.
- A simple interface to allow programs to control Mosaic, including the automatic execution of Mosaic if it is not currently being used.
- A means of allowing secure execution of applications from scripts. This avoids the security problems associated with allowing Mosaic to interpret arbitrary shell scripts

This paper provides an overview of the ICF together with an example showing the use of Mosaic in conjunction with a simple graphical program.

1 Introduction

The goal of the INTERACT project is to develop computer-based learning systems which aid students with their understanding of the complex phenomena underlying engineering domains. The primary pedagogical tools provided to students will be interactive, direct-manipulation, simulations of relevant aspects of the behaviour of the domain system under consideration.

Ideally, the software implementing these simulations will enforce a strict separation between the computational core and the user-interface which grounds abstract elements in a concrete scenario. This layered architecture has been designed to allow the application of the same core simulation across a number of engineering domains. An object-oriented simulation toolkit is currently being developed within INTERACT — the INTERACT *Presentation Manager*[3].

Simulation applications are delivered to students with accompanying multimedia courseware. This courseware will be delivered using the World-Wide-Web, using the X version of Mosaic[4] as the browser to be used by students and courseware authors.

A number of requirements were identified for the communication between simulations and the hypermedia used. These requirements include:

- Users should be able to run applications from within courseware.
- Commands to be sent to applications should be embedded within courseware. For example, to change an input value on a running simulation.
- Applications should be able to control the hypermedia browser. For example, to provide context-sensitive help on a particular item within the simulation.

The *Interact Communication Facility* (ICF) was developed to provide this functionality.

2 The Architecture of Interact Systems

The educational systems built by the INTERACT project are comprised of three main classes of programs:

Simulations — which replicate some relevant portion of the behaviour of the domain system under consideration.

Data visualisers — which provide means of viewing simulation data in suitable, usually graphical, forms.

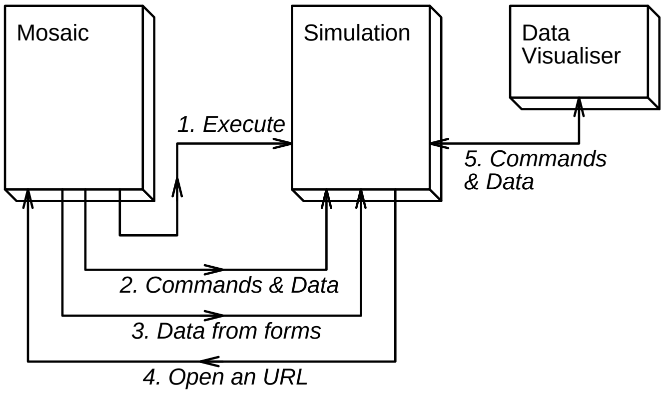

Figure 1: Communication between applications.

Hypermedia browsers — which provide supporting material, usually termed *courseware*, to guide and aid students in their use of the supplied simulations.

The INTERACT project has largely concentrated on the development of tools and simulations rather than on providing courseware. The design process has involved a large amount of evaluation, which has been carried out by Heriot-Watt's *Institute for Computer Based Learning* [5].

3 The Role of the ICF

The role of the ICF within INTERACT systems is to provide developers, both of higher level tools and complete systems, with a simple means of sending data and commands between applications. The various types of communication which must be provided are shown in Figure 1.

These messages are discussed below[1]:

1. Applications must be started from within the HTML.
2. References to commands which can control an application must be embedded within the HTML.
3. Data from fill-out forms [6] must be communicated from Mosaic to an application.
4. Applications must be able to command Mosaic to open an URL.
5. Simulations must be able to exchange commands and data with data visualisers.

[1]It should be noted that the term *application* is used here to signify any user program other than Mosaic — these can be simulations, data visualisers or more general programs such as editors.

4 The Components of the ICF

The software tools which provide the functionality described above are now described in more detail.

4.1 The Interact Command Language

The commands and data sent between applications in the ICF are expressed in a simple, easily extensible, extension language — the *Interact Command Language* (ICL). The ICL is intentionally restricted to fundamental data types and simple command constructs — although individual tool or application builders are free to extend the basic language for use in their system.

Applications which are to *receive* messages via the ICF must therefore include an ICL interpreter which has been extended with commands appropriate to that application [2].

Programmers register message handling functions with the application's ICL interpreter in a manner similar to event-driven user interface code. These functions are then called when that application receives an ICF message which contains the relevant commands.

4.2 Sending Messages

Messages in the ICF can be sent in three ways:

- Applications can use the `ic_send()` function to send messages directly.
- The ICL's `send` command may be used, this is simply a means of invoking `ic_send()` from within the ICL.
- The `icli` (ICF command-line interface) program can be used.

The latter mechanism is of most interest in that it is the means by which Mosaic can send messages. Originally, we modified the distributed code for Mosaic (`xmosaic` as it was then) so as to allow the 'viewing' of shell scripts — this modification became unnecessary with the inclusion of MIME support in Mosaic 2.0 [7]. Commands could then be included within HTML documents by referencing the appropriate script which would invoke the appropriate ICF program to send a message.

Unfortunately, it has been discovered that allowing W3 browsers to invoke delivered documents as shell scripts creates security problems [8]. Scripts executed in this manner by a browser may, for example, delete all of a users

[2] The ICL interpreter has proven to be a useful tool in its own right, the Presentation Manager uses the ICL to provide persistence, in text files, for its graphical and simulation objects.

files! The guidelines for using the ICF have therefore been changed to recommending the inclusion of HTML references to ICL scripts which can use the ICL's `send` command. The `icli` program being used as a safe replacement for the normal UNIX shell programs.

4.3 Message Passing

At it's lowest level, the ICF passes messages between applications using ONC Remote Procedure Calls (RPCs)[9]. Messages are routed via a single central server program (`ic_svr`) which is the only RPC server in the ICF. The first ICF application to be executed by a user runs this server which then exists until the user's last ICF application exits.

Each ICF application is identified by a suitable string. However, each user's ICF system is independent, the only constraint being that each user possesses only one application with a particular name. The selection of which application to send a message to is based purely on this name. The current implementations of the ICF do not support the transmission of messages between applications belonging to different users, although this could be added if required through some qualification of an application's name with a user name and remote host.

Delivery of messages to applications is a two stage process:

- Firstly, applications are simply notified that the `ic_svr` is holding a new message for them through a signal being sent between the server process and the application. The messaging receiving portion of the ICF in the application does nothing more at this stage than increment its count of messages waiting for that particular application. This is, as far as the application developer is concerned, handled completely transparently by a signal handling function provided as part of the ICF library.

- At some suitable time, from the applications point of view, it can decide to process all waiting messages. This is done by a single call to `ic_process_messages()`, this function iterates through all the waiting messages, retrieves their contents from the `ic_svr` and uses the application's ICL interpreter to read the data in the message and to invoke the registered functions.

This two stage process was designed so as to provide the simplest possible interface to application developers.

4.4 Executing Applications

Now that the ICF does not depend on allowing Mosaic to execute shell scripts, the requirement to allow applications to be started from within HTML documents becomes problematical. We cannot allow the ICF to execute arbitrary shell commands as this would reintroduce the security problem mentioned above. What is required is that the ICF should be able to execute a limited number of 'trusted' applications, which system administrators have decided pose no threat to security. It is obvious that these applications should *not* provide any means of executing arbitrary shell commands, as this feature may be accessible from the ICF.

In the current version of the ICF, the `ic_svr` program maintains a list of applications which can be invoked by sending the `ic_svr` program the appropriate message. This feature exploits the fact that the `ic_svr` program can itself receive messages (as opposed to the usual case, where messages are routed through this program).

4.5 Retrieving Data from Fill-Out Forms

The addition of fill-out forms [6] has arguably turned Mosaic into a generalised user-interface tool — to allow application developers to exploit the power of this new feature a mechanism has been added to the ICF which allows data from forms to be sent to applications in a manner similar to other ICF messages.

Data from a form in Mosaic reaches an application by the following, somewhat circuitous, route:

1. The HTML specification for the form must specify that the data be delivered to the `ic_form_svr` server script. This is a simple *Common Gateway Interface* (CGI)[10] script which converts the data into an ICL file. The `ic_form_svr` script is written in *perl* [11] using Steven Brenner's library of CGI utility functions [12].

2. The resulting script is returned to Mosaic, which passes it to the appropriate ICF program (i.e. `icli`). The execution of this scripts causes the data from the form to be sent to the user's `ic_svr`.

3. The `ic_svr` program then broadcasts the information to all applications, which are responsible for taking appropriate actions, if any.

Each form includes a *logical identifier* which is used to relate sets of forms that contain the same sets of fields and which map to the same actions by applications, but which may differ in their appearance to users. This feature allows a variety of related forms to be used without having to support a

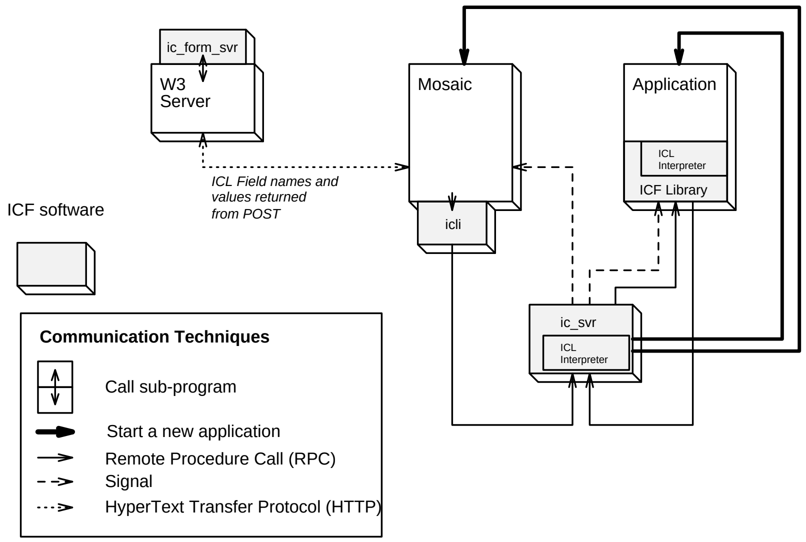

Figure 2: Communication techniques.

large number of user-interfaces in the program itself. New forms which are consistent with existing forms may be added without modifying the program in any way.

4.6 Controlling Mosaic

Another exception to the usual message delivery mechanism is when the message sent is intended to control Mosaic. These messages, which usually arise from an application using the `ic_goto_document(char *URL)` function, are actually sent to the user's `ic_svr` which uses the standard Mosaic remote-control mechanism to open the appropriate document[13]. If the user does *not* have a Mosaic running when such a message is sent the ICF will open a new Mosaic using the supplied URL as its home-page.

The various communication techniques used by the ICF are shown in Figure 2.

5 A Demonstration

The functionality described above will be demonstrated by showing how it can be used in the development of an extremely simple X graphical editor, `xged`. Although this application is *not* representative of the domains addressed by INTERACT, it is complex enough to provide a suitable demonstration.

5.1 Basic Message Handling

The `xged` program provides facilities to draw shapes of varying types in a window. To turn the basic program into an ICF application, two things must be done:

1. The program must initialise its ICF code and associate handler functions with ICL command names.
2. Messages must be retrieved by calling the `ic_process_messages()` function at appropriate times.

The code to do this is quite straightforward:

```
....
void xged_draw_command();
void xged_erase_command();
void xged_form_box_command();
....
if(ic_x_initialise(argc, argv, XtDisplay(drawing_area))) {
   /* Create simple commands. */
   ic_create_command("draw",  xged_draw_command);
   ic_create_command("erase", xged_erase_command);
   ....
   /* Form handlers */
   ic_create_command("form-Line",   xged_form_box_command);
   ....
   /* Add a work procedure to check for ICF messages. */
   XtAppAddWorkProc(app_context, xged_work_proc, top_level);
}
```

Where a suitable Motif work procedure would be:

```
Boolean xged_work_proc(client_data)
XtPointer client_data;
{
  ic_process_messages();
  return(False);
}
```

The information on the application's X display is only used if it is necessary for the ICF initialisation code to fork a new `ic_svr`. The new forked process uses this information to close the connection to the X server before exec'ing a new `ic_svr`.

The functions associated with the ICL commands `draw` and `erase` draw a specified shape and erase the current drawing, respectively. The functions

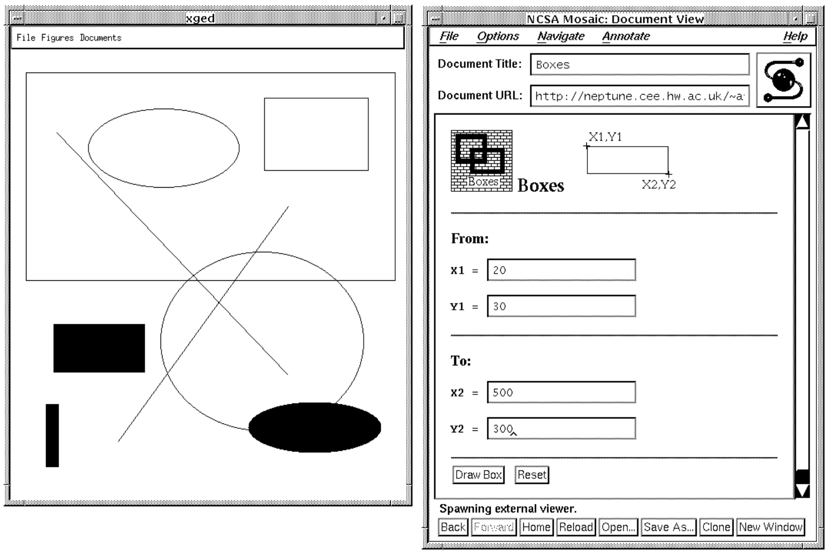

Figure 3: `xged` controlled by Mosaic

used to process data from forms, which in this case would draw a line using co-ordinates supplied from the form (see Figure 3.), are similar to the functions used to handle normal messages except that a different parameter binding mechanism must be used to handle the associative nature of the data supplied from the form (data is returned as a set of associations between strings representing form field names and their values).

5.2 Using Mosaic for Help

In `xged`, the `Help` button uses the ICF to open an appropriate HTML document in Mosaic. This document may contains direct commands to control `xged` together with forms which provide alternative means of accessing the shape drawing functions in the program.

```
void help_action(w, ip, call_data)
   Widget w;
   int   *ip;
   XmAnyCallbackStruct *call_data;
   {
      ic_goto_document(
         "http://neptune.cee.hw.ac.uk/~afs/xged/introduction.html");
   }
```

A picture of `xged` being used with Mosaic is shown in Figure 3.

5.3 Adding Features

The fact that the ICF provides `xged` with the ability to translate textual descriptions of drawings into pictures can be exploited to add the loading and saving of drawings to files. A function is provided in the ICF library which will load the contents of a file and use the ICL interpreter to convert these to data and commands. This means that file loading in `xged` is provided almost 'for free' as a result of using the ICF. Saving drawings in files is performed by iterating through all the shapes in a drawing and writing out the ICL which, when loaded, will cause `xged` to reproduce the required shapes.

In practise it is probably more useful to distribute such data files as commands scripts which will cause these commands to be sent to the application, rather than simply loading them from local files. This raises the possibility that arbitrary data-objects can be distributed on the W3 along with more familiar types of data such as HTML documents and images.

6 Status of the Software

At present, the *Interact Communication Facility* (ICF) is being used by members of the INTERACT project. Its usage so far has been mainly limited to the development of experimental prototypes, although it is viewed as being an integral component of the full teaching systems currently under development.

The ICF is currently running on Sun-4s, HP 700s and Digital AXPs, porting is mostly dependent on the availability of the appropriate ONC RPC software with it's `rpcgen` protocol compiler.

It is hoped that the ICF, which is currently being used mainly within the INTERACT project, will soon have reached a state where it is appropriate to make it publicly available via anonymous ftp.

7 Conclusions

This paper outlines a software tool, the *Interact Communication Facility* (ICF) which is designed to allow the integration of educational simulation software and courseware delivered via the World-Wide-Web.

8 Acknowledgements

The author would like to acknowledge the contribution of all the members of the INTERACT project team, whilst accepting full responsibility for the views expressed here.

I must also thank Calum Smeaton, the developer of the *Interact Presentation Manager*, for the copious feedback he has provided from his imaginative use of the facilities provided by the ICF

9 References

1. Slater, A.F. (1994), Controlled by the Web,
   Paper submitted to the First International Conference on the World-Wide Web,
   CERN, Geneva, Switzerland.
   http://neptune.cee.hw.ac.uk/~afs/home-page.html
   The HTML version of this paper.
2. Thomas, R. (1994), The INTERACT Project Home Page,
   http://www.eng.cam.ac.uk/interact/interact.html
3. Smeaton, C. (1994), The Presentation Manager,
   http://www.cee.hw.ac.uk/pm/nav.html,
   Documentation on the INTERACT *Presentation Manager*.
4. NCSA Mosaic Project (1994), NCSA Mosaic Home Page,
   http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html
5. Heriot-Watt's *Institute for Computer Based Learning*,
   http://www.icbl.hw.ac.uk/
6. NCSA Mosaic Project (1994), Mosaic for X version 2.0 Fill-Out Form Support,
   http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill-out-forms/overview.html
7. NCSA Mosaic Project (1994), Multimedia Configuration,
   http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/d2-multimedia.html
8. NCSA Mosaic Project (1994), Executing Shell Scripts Inside Mosaic,
   http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/executing-shell-scripts.html
9. Bloomer, J. (1992), *Power Programming with RPC*, O'Reilly & Associates, Sebastapol, CA.
10. McCool R., (1993), The Common Gateway Interface,
    http://hoohoo.ncsa.uiuc.edu/cgi
11. Wall, L. and Schwartz, R.L., (1990), *Programming perl*, O'Reilly & Associates,
    Sebastapol, CA.
12. ftp://ftp.ncsa.uiuc.edu/Web/ncsa_httpd/cgi/cgi-lib.pl.Z
    Steven Brenner's library of perl CGI utility functions.
13. NCSA Mosaic Project (1993), Using Mosaic by Remote Control,
    http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/remote-control.html