An Integrated Learning Support Environment for Computer Architecture

P.S. Coe, F.W. Howell, R. N. Ibbett,
R. McNab and L. M. Williams
Computer Systems Group
Department of Computer Science
University of Edinburgh, Edinburgh, EH9 3JZ, UK

December 17, 1996

Abstract

The Integrated Learning Support Environment (ILSE) for Computer Architecture provides structured on-line access to a large body of text and diagrams in which the diagrams (a) remain visible on-screen even when the text is scrolled, (b) may in some cases be animated to provide a visual demonstration of activities occurring within a computer system. It uses a WWW multi-frame window system, with separate frames for text, diagrams and navigation control. The animated diagrams are driven by output from an architecture simulation system which has been (re-)written in Java. Using Java, live simulations can be incorporated into the WWW pages and run remotely.

1 Introduction

The idea of creating an Integrated Learning Support Environment (ILSE) for Computer Architecture was inspired by a lecture given by the author of the MacFarlane Report [Mac92]. The ILSE aims to provide structured on-line access to a large body of text and diagrams in which the diagrams (a) remain visible on-screen even when the text is scrolled, (b) may in some cases be animated to provide a visual demonstration of activities occurring within a computer system. An early version of the ILSE used a specially written program which set up navigation and diagram windows and spawned a Web browser to display the text. The introduction of a multi-frame window facility in the html standard [Rag96], however, led to the abandonment of this system and the creation of a multi-frame Web version instead.

The text of the ILSE for Computer Architecture is based largely on the textbook "Architecture of High Performance Computers" written by Roland N. Ibbett and Nigel P. Topham and originally published by Macmillan Educational Ltd [Top89]. Volume I dealt with uniprocessors and vector processors, and Volume II with multiprocessors. This book is now out of print, however, so the copyright has reverted to the authors. Because the original text was prepared for the publisher using LaTeX and since much of the material is still appropriate to an MSc course module on Concurrent Computer Architecture being given by one of the authors (RNI), this text was an obvious starting point for the ILSE.

The animated diagrams are produced using a Java version of HASE [IHH95]. HASE is the Department's Hierarchical computer Architecture design and Simulation Environment, originally written in Sim++. By re-implementing HASE in Java, working simulation models can be incorporated into the ILSE, thus allowing students to experiment with them directly. This contrasts with using a traditional simulation written in a language such as Simula or C++, where exporting simulation code requires recompilation and installation on each different machine.

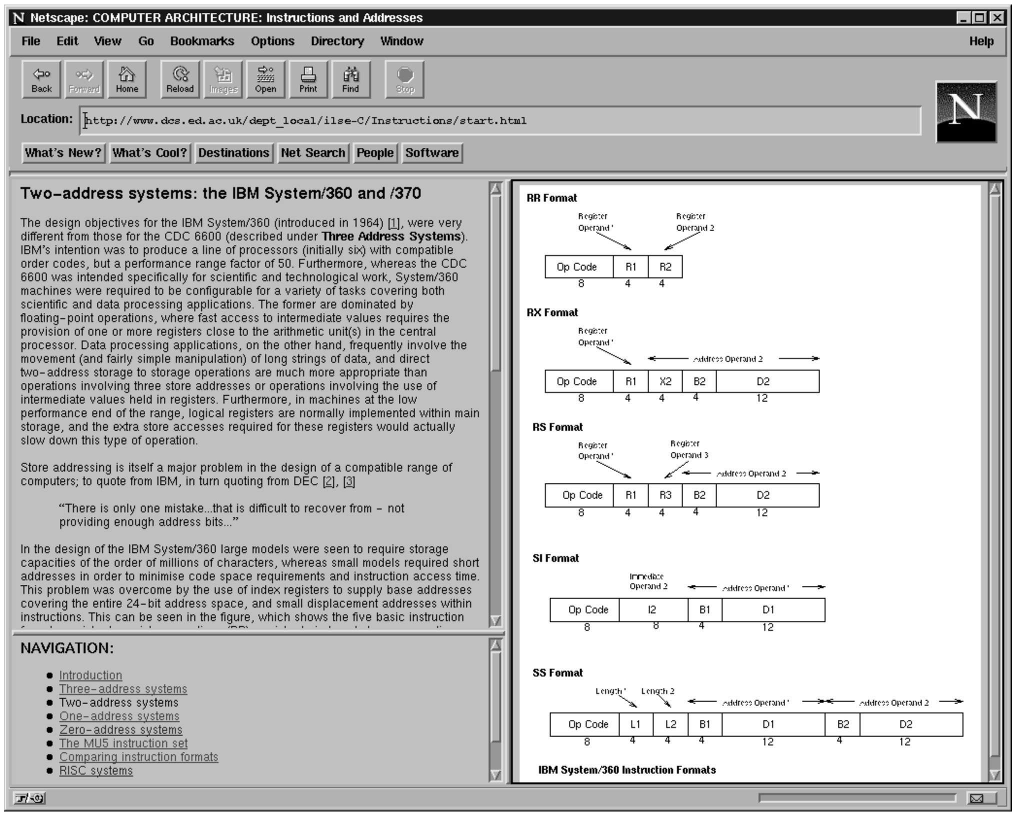
2 The Window Structure

Each window in the ILSE contains three separate frames (figure 1), a Text frame, a Figure frame and a Navigation frame.

2.1 The Navigation frame

The purpose of the Navigation frame is to allow structured access to the large volume of text available as part of the Computer Architecture ILSE. All links to the text and figures are contained in the navigation files. There are thus no 'spaghetti' links in the Text frame, apart from reference citations (section 2.4).

Whenever a selection is made in the Navigation frame, a new navigation menu is loaded. This then causes the corresponding text and figure to be loaded (not all sections actually have useful figures, of course, and in these cases a message to this effect is displayed instead).



http://www.dcs.ed.ac.uk/dept_local/ilse-C/Instructions/start.html

Two-address systems: the IBM System/360 and /370

The design objectives for the IBM System/360 (introduced in 1964) [1], were very different from those for the CDC 6600 (described under **Three Address Systems**). IBM's intention was to produce a line of processors (initially six) with compatible order codes, but a performance range factor of 50. Furthermore, whereas the CDC 6600 was intended specifically for scientific and technological work, System/360 machines were required to be configurable for a variety of tasks covering both scientific and data processing applications. The former are dominated by floating-point operations, where fast access to intermediate values requires the provision of one or more registers close to the arithmetic unit(s) in the central processor. Data processing applications, on the other hand, frequently involve the movement (and fairly simple manipulation) of long strings of data, and direct two-address storage to storage operations are much more appropriate than operations involving three store addresses or operations involving the use of intermediate values held in registers. Furthermore, in machines at the low performance end of the range, logical registers are normally implemented within main storage, and the extra store accesses required for these registers would actually slow down this type of operation.

Store addressing is itself a major problem in the design of a compatible range of computers; to quote from IBM, in turn quoting from DEC [2], [3]

> "There is only one mistake...that is difficult to recover from – not providing enough address bits..."

In the design of the IBM System/360 large models were seen to require storage capacities of the order of millions of characters, whereas small models required short addresses in order to minimise code space requirements and instruction access time. This problem was overcome by the use of index registers to supply base addresses covering the entire 24-bit address space, and small displacement addresses within instructions. This can be seen in the figure, which shows the five basic instruction

NAVIGATION:

- Introduction
- Three-address systems
- Two-address systems
- One-address systems
- Zero-address systems
- The MU5 instruction set
- Comparing instruction formats
- RISC systems

Figure 1: An example WWW window from the ILSE

2.2 The Text frame

The text is structured hierarchically, *i.e.* it is divided up into 'chapters', corresponding to book chapters, and each chapter is divided into sections and sometimes subsections. Clicking on a chapter heading in the Navigation frame brings in the Navigation menu for that chapter, together with the introductory text in the Text frame and the contents list in the Figures frame.

Where sections are divided into subsections, the format of the Navigation frame changes again when the section is selected. Each navigation menu contains links allowing movement back up the hierarchy.

For administrative convenience each chapter is held in a different subdirectory in the filestore. In order to maintain referential consistency among frames when moving between chapters, the frame structure has to be taken down and reloaded at each change.

2.3 The Figure frame

The prime purpose of the Figure frame is to display the diagram which accompanies the text in the Text frame. The idea is that by not embedding the figure in the text, it remains visible even when the text is scrolled.

A few, and hopefully an increasing number of figures are active, i.e. they contain visual demonstrations of some the activity taking place within a computer system (section 4).

The Figure frame is also used to display contents pages at the start of a section, and for references.

2.4 The References

Clicking on a reference citation in the text brings up a page which overlays the Figure frame. This page can be dismissed, so as to bring back the original diagram associated with that text. (For consistency of operation, this happens even when there is no figure for the section concerned.)

The reference pages can include hypertext links, *e.g.* to manufacturers' home pages. Clicking on one of these references spawns a separate instantiation of the browser, through which the referenced document can be viewed without leaving the ILSE.

3 Text Conversion

In principle a LaTeX-to-html conversion program was all that should have been needed to convert the original LaTeX source to html. However, this proved not to be quite so simple in practice for a number of reasons. For example, the source text included indexing commands, labels and citations which could not be handled properly by the converter. A utility was therefore written and used to strip these out by pre-processing each of the files. Furthermore, the available converter was designed to work in the context of a single frame window, with in-line diagrams. For the original textbook the publisher had produced the diagrams manually, and the figure commands in the LaTeX source simply left space for the printer to paste them in. Thus the figure commands also had to be edited out.

Some of the figures themselves had subsequently been reproduced for the lecture course using the idraw utility, and most of the rest have since been reproduced in this same way. The resulting .ps files have been converted to .gif format for use in the ILSE.

A further problem with the converter was that although it was designed to operate with quite large and highly structured LaTeX documents, experience with conversion of the HASE user manual showed that the naming convention used for the resulting .html files does not make for easy navigation and, furthermore, the diagrams can get out of sequence. Rectifying this situation can be very tedious. The ILSE text was therefore split manually into a set of individual files, one per section/subsection of the original text, and each file converted separately.

Conversion of the references has been done by hand. The method of displaying the references in the ILSE (*i.e.* as individual collections associated with individual text frames) meant that no simple automatic convertor would have been useful.

Some pages are difficult to convert to .html format, *e.g.* because they contain Greek characters or complex equations. There are two possibilities here. One is to produce a conventional .dvi file from the LaTeX source and convert it to postscript format. When the .html file for the relevant text frame is loaded it spawns an external postscript viewer, and the html text contains a message informing the user that will happen. The second is to convert the postscript file to a .gif file, and display this instead of a .html file in the text frame. The relative advantages and disadvantages of these two methods are still being investigated.

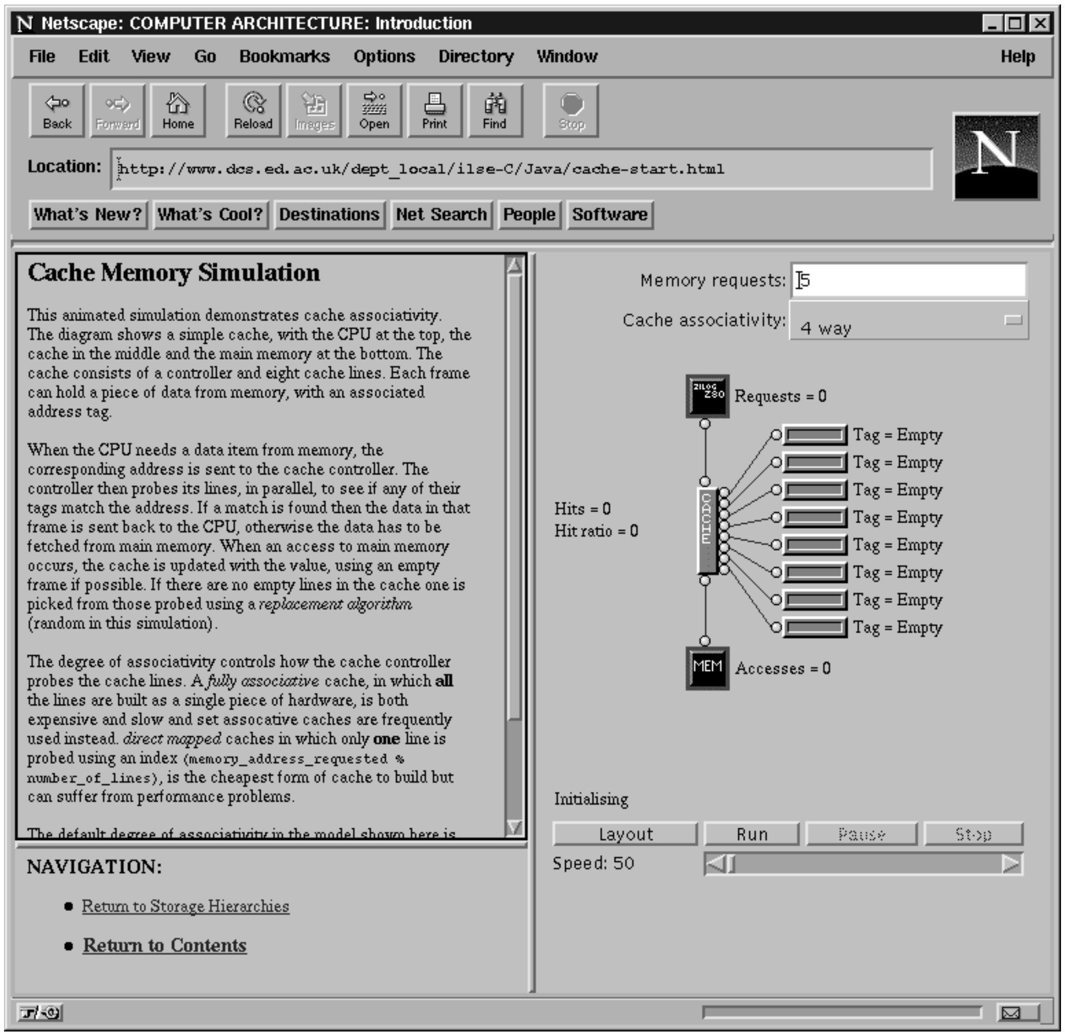
4 Creating Animated Diagrams

The animated diagrams are driven by output from the Department's Hierarchical Architecture design and Simulation Environment (HASE). HASE allows for the rapid development and exploration of computer architectures at multiple levels of abstraction, encompassing both hardware and software. One of the distinguishing features of the HASE environment is its support for animated diagrams of simulation models. In order to make use of this feature in the WWW environment, a version of HASE has been written in Java. This allows the simulations to be run remotely and the animation to be displayed using a Web browser such as Netscape.

The main aim of producing a Java based simulator was to allow inclusion of live simulation models into Web documentation.

The animation facilities are illustrated in figure 2, which demonstrates cache associativity. The CPU is at the top, the cache in the middle and the main memory at the bottom. The cache consists of a controller and eight cache frames. Each frame can hold a piece of data from memory, with an associated address tag.

Text boxes and buttons allow the user to control the simulation and change initial parameters. Entities and ports have their own icons loaded


Netscape: COMPUTER ARCHITECTURE: Introduction
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
What's New? What's Cool? Destinations Net Search People Software
Requests = 0
Tag = Empty
Tag = Empty
Tag = Empty
Tag = Empty
Tag = Empty
Tag = Empty
Tag = Empty
Tag = Empty
Hits = 0
Hit ratio = 0
Accesses = 0
Layout Run Pause Stop

Location: http://www.dcs.ed.ac.uk/dept_local/ilse-C/Java/cache-start.html

Cache Memory Simulation

This animated simulation demonstrates cache associativity. The diagram shows a simple cache, with the CPU at the top, the cache in the middle and the main memory at the bottom. The cache consists of a controller and eight cache lines. Each frame can hold a piece of data from memory, with an associated address tag.

When the CPU needs a data item from memory, the corresponding address is sent to the cache controller. The controller then probes its lines, in parallel, to see if any of their tags match the address. If a match is found then the data in that frame is sent back to the CPU, otherwise the data has to be fetched from main memory. When an access to main memory occurs, the cache is updated with the value, using an empty frame if possible. If there are no empty lines in the cache one is picked from those probed using a *replacement algorithm* (random in this simulation).

The degree of associativity controls how the cache controller probes the cache lines. A *fully associative* cache, in which **all** the lines are built as a single piece of hardware, is both expensive and slow and set assocative caches are frequently used instead. *direct mapped* caches in which only **one** line is probed using an index (`memory_address_requested % number_of_lines`), is the cheapest form of cache to build but can suffer from performance problems.

The default degree of associativity in the model shown here is

NAVIGATION:

- Return to Storage Hierarchies
- **Return to Contents**

Memory requests: 5

Cache associativity: 4 way

Initialising

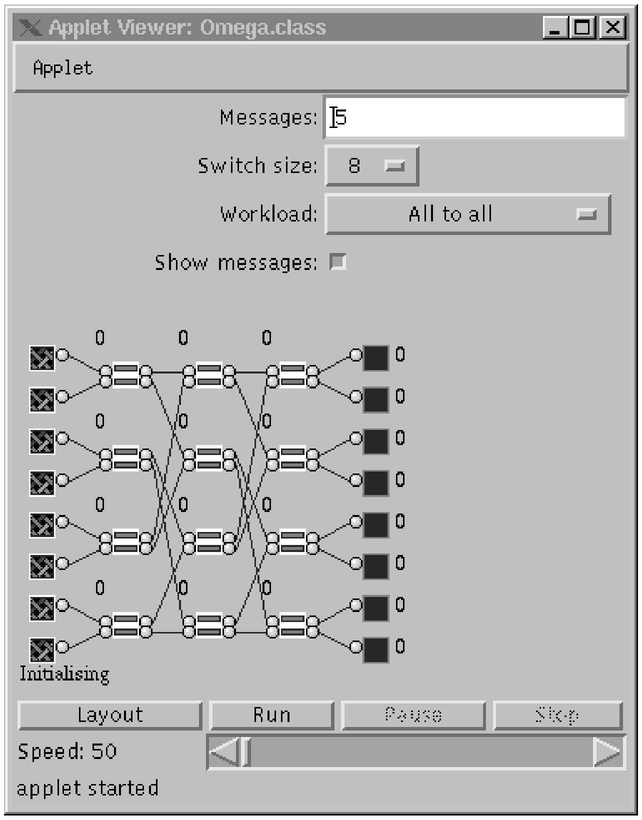
Speed: 50

Figure 2: An example cache simulation

from .gif files. The icons can be changed to represent the current state of the entity, and other entity parameters can be displayed as text. Messages passing between entities are displayed as squares which travel along the connecting lines, the number attached to the square is the message tag.

A larger scale simulation is shown in figure 3 which shows an omega network. The size of the network can be set using the controls at the top, and the workload can also be varied.

These examples were constructed using a general purpose simulation toolkit based on the `simjava` simulation library extensions to Java and the `simanim` animation package.



Messages: 5

Switch size: 8

Workload: All to all

Show messages:

Initialising

Speed: 50

applet started

Figure 3: An example network simulation

4.1 `simjava` - The Simulation Library

Java incorporates the language features necessary for simulation, notably objects and threads. Current Java implementations compile down to an intermediate byte code, which is interpreted. Thus the main disadvantage of using Java (as opposed to a C++ based simulator) is the longer simulation run times. This penalty is quantified in section 4.3.

SIM++, a discrete event simulation library for C++ written by Jade Simulations Inc [Jad92] has been used for computer architecture simulations as part of the HASE project for several years. To allow simulations to be run on architectures not supported by Jade (such as Linux and the Cray T3D), the library was reimplemented using C++ and standard threading libraries, to produce a new library called HASE++ [How96]. HASE++ was used as the basis for the Java simulator [McN96].

A `simjava` simulation is a collection of Java objects (*Sim_entities*) each of which runs in parallel in its own lightweight thread. The *Sim_system* object controls all the threads, advances the simulation time and maintains the event queues.

Entities communicate and synchronise by passing *Sim_event* objects. The primitives of the language are:

- `sim_schedule(Sim_port port, double delay, int tag, Object data)` sends a message to the entity connected to the port after simulation time delay with the given tag and data.
- `sim_wait(Sim_event ev)` waits for an event sent using `sim_schedule`.
- `sim_hold(double t)` blocks for t simulation time units.
- `sim_trace(int level, String msg)` adds a line to the trace file.

This simulation model has been used for modelling hardware and for modelling parallel software.

Figure 4 shows how the Java simulator may be linked with existing C++ based tools.

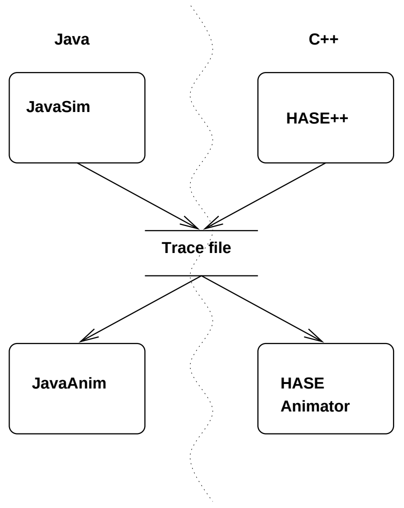


Figure 4: Linking Java simulations with HASE++ simulations

Because there is a standard trace file format, traces produced from a HASE++ simulation may be read into a Java display, and vice-versa. Thus the Java simulation environment may be used for displaying results from a long running HASE++ simulation.

4.2 `simanim` - The Animation Package

`simjava` itself is a simulation library; it produces a trace file as its output. User animations are constructed using the `simanim` package, a toolkit for building animations. `simanim` provides two methods which are designed to be overridden by the user to build the animation display for the application:

- **anim_init()** is extended to add buttons, controls and parameter fields to the display.
- **anim_layout()** is extended to position the entity icons and join ports.

The following example shows how a simple animated entity is created. The example is an entity which keeps a count of the number of messages which arrive at its port, and displays the count on screen. Its icon is provided by a file `bucket.gif` painted using a bitmap editor. The constructor (`public Bucket(...)`) sets the (x,y) position of the icon, adds the port to the side, and adds the displayed parameter `Count`. The simulation behaviour is defined by the `body()` method. This is an infinite loop which waits for an event, updates the count and writes the trace line. The result is shown in figure 5.

```
class Bucket extends Sim_entity {
  int count = 0;
  Sim_port in;
  public Bucket(String name, int x, int y, int side) {
    super(name, "bucket", x, y);
    count = 0;
    in = new Sim_port("in", "port", side, 10); add_port(in);
    add_param(new Anim_param("Count", Anim_param.NAME_VALUE,
              "0", -10, -5));
  }
  public void body() {
    Sim_event ev=null;
    while(true) {
      sim_wait(ev);
      sim_hold(0.01);
      count++;
      sim_trace(1, "P "+count);
    }
  }
}
```

As well as displaying the values of entity states as text on screen, it is possible to use the state to select different icons, as in figure 6, which shows the two possible states of the entities used in the omega network shown in figure 3.

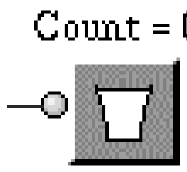


Figure 5: A simple animated entity

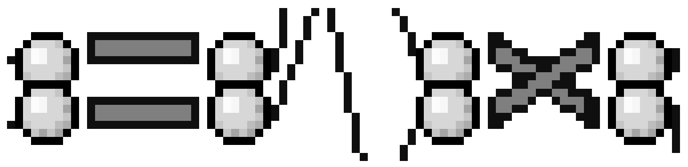


Figure 6: Icons representing the states of network switching elements.

4.3 Performance of `simjava`

To compare the performance of the Java and C++ versions of the library, a simple simulation was written in both languages and the execution time measured. The Java version was run as a stand-alone application, as a Netscape applet on a Sun SPARCstation 5 under Solaris, and as a Netscape applet on a Pentium 133 under Windows NT. The simulation contains two entities which pass 200 messages between them, a simple example for comparative purposes. The results are shown in table 1.

Platform	Average execution time over 5 runs
Solaris C++	1538 ms
Solaris Stand-alone Java	12910 ms
Solaris Netscape	11214 ms
Windows NT Netscape	9341 ms

Table 1: Simulation execution times

The results show that the simulation ran around ten times faster under C++, and the stand-alone Java and Netscape were roughly equal. In practice performance of the Netscape versions appears to be acceptable to users.

5 Conclusion

Taking a book from the printed page to the screen is not trivial, and many issues arise from the change of format. The first is the problem of moving about the text; this has to be designed carefully in order to avoid the spaghetti links of unstructured text on the Web.

`simjava`, a java based simulation package was used for transforming diagrams from static pictures into working models. Although it has been used here purely for demonstrating computer architectures, it is a powerful and complete simulation language in its own right, useful for communicating complex ideas between designers.

Future plans for `simjava` include implementing more Java modules for displaying graphs and timing diagrams, and providing 3D models using VRML-2.

Currently the ILSE is being used by MSc students studying Concurrent Computer Architecture at the University of Edinburgh, and their feedback will be evaluated early in 1997. Access to the ILSE is currently restricted to our own departmental users, though a sampler can be viewed at `http://www.dcs.ed.ac.uk/rni/ilse-index.html`.

Acknowledgements

The HASE project is supported by the UK EPSRC as part of grant GR/J43295.

References

[How96] F.W. Howell. HASE++: a discrete event simulation library. `http://www.dcs.ed.ac.uk/home/fwh/hase++/hase++.html`, Feb 1996.

[IHH95] R.N. Ibbett, P.E. Heywood, and F.W. Howell. HASE: A Flexible Toolset for Computer Architects. *The Computer Journal*, 38(10):755–764, 1995.

[Jad92] Jade Simulations International Corp., Calgary, Canada. *SIM++ User Manual*, 1992.

[Mac92] A.J.G. MacFarlane. Working party report: Teaching and learning in an expanding higher education system. Available from COSHEP, St Andrew House, 141 West Nile Street, Glasgow, UK, 1992.

[McN96] R. McNab. SimJava: a discrete event simulation library for Java. `http://www.dcs.ed.ac.uk/home/hase/simjava/simjava-1.0`, July 1996.

[Rag96] Dave Raggert. HTML 3.2: reference specification. `http://www.w3.org/pub/WWW/TR/PR-html32-961105`, Nov 1996.

[Top89] R.N. Ibbett & N.P. Topham. *Architecture of High Performance Computers.* Macmillan Educational, UK, 1989.