JavaHase: Automatic Generation of Applets from Hase Simulation Models

F. Mallet and R.N. Ibbett

Institute for Computing Systems Architecture
School of Informatics - University of Edinburgh

Keywords: Simulation Applets, Computer Architectures, Object-oriented Modelling.

Abstract

Hase is a design and simulation environment that allows for rapid development and exploration of computer architectures at multiple levels of abstraction. The great flexibility of the graphical display has enabled the creation of models (Tomasulo's algorithm, DLX architecture, *etc.*) which have proved to be useful in their own right, particularly for teaching and demonstration purposes. In order to make the models widely accessible, two different ways of exporting them via the WWW have been investigated, WebHase and JavaHase. WebHase uses a viewer applet to visualise pre-run Hase simulations whilst JavaHase allows existing simulation models to be translated into fully interactive simulation applets.

INTRODUCTION

Hase is a Hierarchical computer Architecture design and Simulation Environment [1]. It allows for rapid development and exploration of computer architectures at multiple levels of abstraction, encompassing both hardware and software. The architecture structural description (and where appropriate the instruction-set) are specified using the Entity Description Language (EDL), an architecture description language [2] designed as part of the Hase project. An EDL file contains the description of entities' general properties, instances, connections and the types of data to be exchanged by entities through the connecting links. The screen layout is described by an Entity Layout File (ELF) and the behaviour is implemented in Hase++, a C++-like programming language with primitives to manage discrete-event simulations.

Once models are built, they may be used either to evaluate the performance of the architecture in response to specific inputs (data and programs) or for educational or training purposes, *e.g.* to demonstrate the internal mechanisms of the architecture. In this context, distributing the models widely is important, so that they can be extensively used, validated and extended. One way to do this would be to distribute Hase. The Hase design and simulation environment currently runs on Linux and earlier versions also ran on Solaris and Windows NT; some simulations were also successfully run on a Cray T3D. However, installation is not always straightforward and updating with new releases implies the use of a versioning system by the user. In any case, the design environment is not really required by the user who only really needs an executable simulation model and the animator. More importantly, designers may not want to release the source code of their models.

Currently, the most efficient distribution medium is the World Wide Web. Markup Languages (HTML, XML) are an easy way to provide up-to-date documentation and explanations, whilst Java applets are an efficient and secure way to provide interactive executable models. This explains the huge number of simulation packages available over the web [3, 4, 5, 6, 7]. We therefore decided to investigate systems which would allow Hase models to be displayed in a browser. This paper presents two solutions: WebHase and JavaHase.

Neither WebHase nor JavaHase is itself a Web simulation environment. WebHase allows existing, pre-run Hase simulations to be viewed with a Web browser whilst JavaHase is a process that allows existing simulation models to be translated into active simulation applets. As a consequence of this translation process, JavaHase is virtually independent of the simulation engine and so allows submodules written in different languages (C++, Java), with different underlying simulation engines, to be integrated into the same model. This process has been successfully applied to a number of existing HASE models. JAVAHASE offers the users the advantage of being able to create models using a well developed support environment (HASE) and the choice of running the same model as either a Linux application or as an applet.

HASE, HASE++ AND WEBHASE

HASE has four modes of operation, each one corresponding to a phase in the development life-cycle of a project: Design, Build, Simulate and Experiment (Figure 1). Each mode provides the designer with a different set of menu options allowing different operations to be performed. The operations available to the designer in Design mode include add/remove component, adjust connections between components and add/remove parameters/ports from individual components.

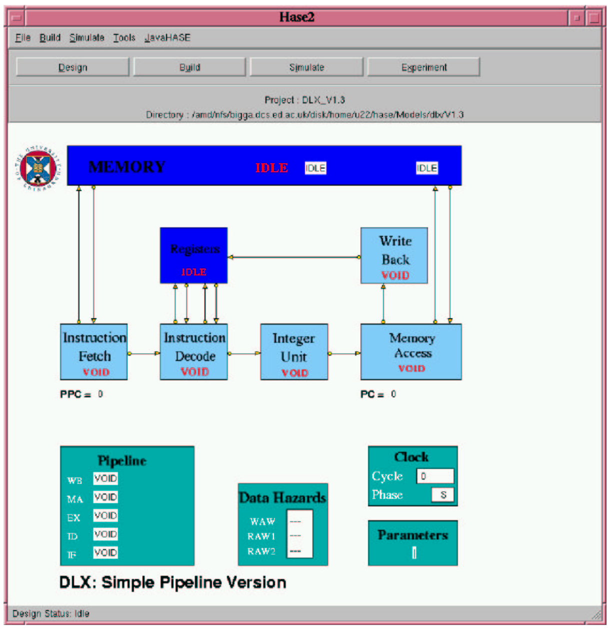


Figure 1: A typical HASE project

Build mode is used to create an executable simulation of the architecture. Simulate mode allows the simulation to be run and the graphical display of the design to be animated. The animator shows packets moving between entities, entity state changes and, in separate pop-up windows, changes to the contents of arrays (*e.g.* registers and memories). Simulate Mode also allows system parameters to be changed and timing diagrams for system components to be viewed. Experiment mode allows automatic multiple executions of the simulation to be performed, with different parameter settings used in each execution.

After defining the properties of the entities in EDL, the designer defines their behaviour in Hase++. Hase++ is a library for C++ which provides discrete event simulations, with entities running in separate threads. It is part of the HASE simulation environment, but may be used by itself. The interface is based on that of Jade's SIM++. Hase++ can be run on Sun Solaris running Solaris threads, on Linux using g++ and the REX threading library and has also been run on a Cray T3D.

EDL and Hase++ information are compiled together to produce an executable file. This file reads inputs from external files, allowing the simulation model to be run with different inputs. Hase++ primitives write events to a trace file which can subsequently be used by the HASE animator to show the results of the simulation in the design window (Figure 2).

Figure 2 also shows how HASE, WEBHASE and JAVAHASE are interrelated. WEBHASE was developed as an undergraduate project which aimed to allow HASE simulations to be viewed with an Internet browser. The WEBHASE converter parses the EDL file, the EL file and the simulation trace file so as to produce a summary file described in XML. This XML file contains all the information required to display and animate a simulation.

Originally, users were allowed to alter model parameters (*e.g.* the instructions in the memory of a computer), then to send these parameters back to a servlet engine which would re-run the simulation using the new parameters. When the simulation was complete, the servlet would create a new XML file which was returned to the viewer applet to be animated. Partly because the servlet engine proved unreliable and partly because of concerns over the potential for overloading the server, the interactive facilities of WEBHASE were disabled and an alternative interactive system, JAVAHASE, developed instead. WEBHASE has nevertheless been retained for use as a demonstration system, allowing pre-run simulations to be viewed over the Web, since it requires less sophisticated browser facilities.

JAVAHASE

JAVAHASE allows HASE projects to be translated into applets containing fully fledged simulation models

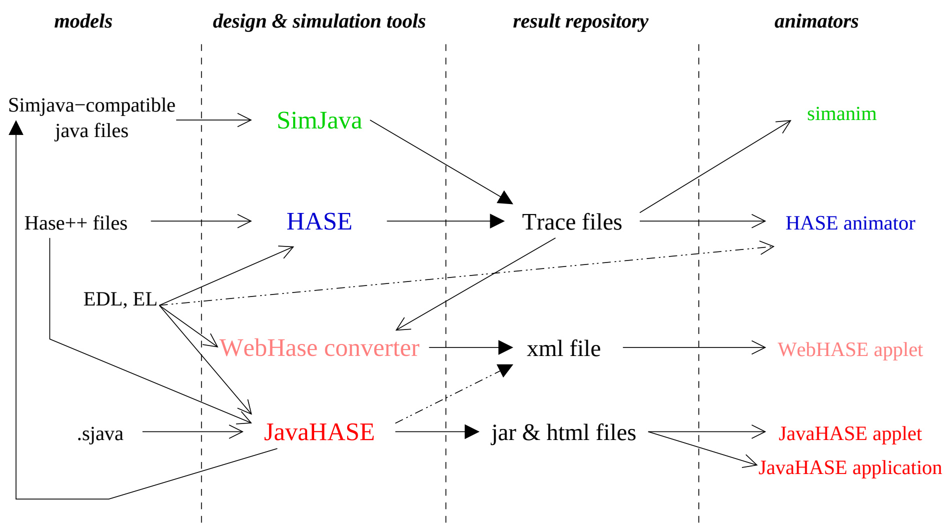


Figure 2: HASE, Simjava, WEBHASE and JAVAHASE

based on `simjava` [3]. `simjava` is a process based discrete event simulation package for Java, derived from Hase++, with animation facilities. Since `simjava` has the same origin as Hase++, it is able to produce compatible trace files (Figure 2) and all Hase++ primitives have a direct equivalent in `simjava`. Consequently, using `simjava` greatly simplifies the translation process. The JAVAHASE system is composed of three packages: javahase.meta, javahase.model, javahase.view.

The javahase.model package is responsible for providing classes allowing for the building of a hierarchical model which interacts with simulation entities. Actually, it conceals the simulation engine internal mechanisms to guarantee the complete independence of JAVAHASE from the underlying discrete-event simulation engine.

The javahase.meta package is used to contain all the information which makes up a HASE project. A part of this package (javahase.meta.generator) is responsible for generating new representations of the project, either an XML file to be given to a WEBHASE applet or Java files to be executed in a JAVAHASE applet or as a JAVAHASE application.

The javahase.view package is composed of two kinds of class. The first kind provides a simple way to display aspects of the entities. The second kind controls the interaction between the classes of the first kind, the model and user requests. This is done according to the Model/View/Controller pattern [8].

The JAVAHASE model and the translation process

When the user decides to generate a fully autonomous JAVAHASE applet, three kinds of java files are generated for each project. These files are generated in the package javahase.<project>, where <project> is the name of the project converted to lowercase and in a format dictated by the Java syntax.

First, for each basic class called EntityClass, the Hase++ description is parsed and translated into a `simjava` compatible Java class. The resulting class is assigned to the javahase.<project>.sim package. Actually, JAVAHASE first looks for an existing `simjava`-compatible description in a .sjava file. If that description is not found, it looks for the Hase++ description in a .hase file. This mechanism allows Java to be used directly to specify the behaviour of all or some of the entities. This can be very useful when the user wants to add some remote or network-oriented features to an existing model. Furthermore, it is not always possible to translate some of the C++ libraries which could be used by specific HASE projects, and in these cases it can be simpler to implement some features directly in Java. This means that some entity descriptions may be in C++ while others are in Java depending on which language is more suitable.

At the end of the translation process, the generated class will inherit directly or indirectly (the inheritance relation is transitive) from the simjava.Entity class, which is the basic class to represent simulation entities in simjava. Such a class will be instantiated automatically for each instance of the corresponding entity, but could also be used as a normal simjava class, in another independent project.

During the translation process, pointers are translated into references and calls to the string.h library are translated into calls of corresponding methods of the java.lang.String class. For each type (structure, instruction-set, link) defined in the EDL description, a Java class is generated and references to these types are translated within the generated classes. Finally, calls to Hase++ discrete-event primitives are replaced by more generic calls which are separately associated with some specific simjava equivalent methods. They could quite easily be associated with primitives of other simulation languages. The only constraint is to generate a compatible trace output.

Second, for each entity, including basic entities, library entities and compound entities, a javahase.model.AEntity is generated and associated with the javahase.<project>.model package. This class conceals the simulation language from the rest of the system. When applicable, the simjava-compatible javahase.<project>.sim is embedded into this entity. This simulation entity could be replaced by another one written in another simulation language. This additional level also allows the hierarchy information to be maintained as well as some information about specific components such as meshes. It is quite rare to find hierarchy management facilities in currently available simulation languages. Neither Hase++ nor simjava contain such information. However, this information can be very useful since it allows a complete part of a system (module) to be reused without rebuilding the hierarchy and the communications within the module.

The model.AEntity classes do not contain any information about the layout. They only provide a transparent way to access a set of parameters shared by a hierarchical system of simulation entities. Each SimjavaEntity instance will be associated with a specific Entity object and will access its parameters on demand. A new class inheriting from the AEntity class can easily be added, allowing a different simulation language to be used.

Third, a controller class is generated for each AEntityClass. These classes deal with the layout attributes and with initial parameters. For each instance, the control class selects a default view according to the entity type. For example, the EDL format provides a macro-instruction to automatically generate n-dimensional meshes of a basic entity. The controller associated with a MeshEntity looks at the entities and connections from which the mesh is constructed and generates drawing properties according to a predefined scheme. The resulting layout allows the dynamic information about each node of the mesh to be accessed during the simulation.

Finally, a main class is generated so as to instantiate the controllers and provide either a Java applet or a Java application, depending on the user's choice.

During the first execution of the main class, the Entity behaviours are executed (the body method) and a trace of events is saved in memory. If the user wants to replay the simulation, the saved events will be animated again. The user may also want to run a new simulation; this can be done by modifying the parameters and then running the new simulation. In this case, the Entities are used again to compute the correct behaviour.

In HASE itself, simulation and animation are separate activities: when a simulation is run, a trace file is produced for subsequent use by the animator. Animation is appropriate during model development and testing or for teaching and demonstration purposes, but not when the user wants to produce performance information as quickly as possible. This effect can also be achieved when running a JAVAHASE model as an application by having two modes of operation: normal and fast. In fast mode, an alternative main class is generated which uses the the simulation-specific classes but not the layout controllers. A trace file is still produced and this can be used either by the JAVAHASE application or by the HASE animator (Figure 2).

JAVAHASE display mechanism

In HASE, an executable file is generated according to the contents of the project's EDL file and the behavioural (.hase) description files of its entities. For performance reasons, this executable file does not contain any information about the graphical on-screen layout. In order to be able to analyse the simulation and the model, the HASE designer must include some dump_state() statements in the .hase files. These statements dump, into a trace file, the current values of an entity's parameters, as declared in the EDL file. When the simulation is finished, the HASE animator reads the trace file and produces an animation. The animation can be run repeatedly to explore different part of the simulation. Alternatively, with large simulations, some data mining techniques can be used on

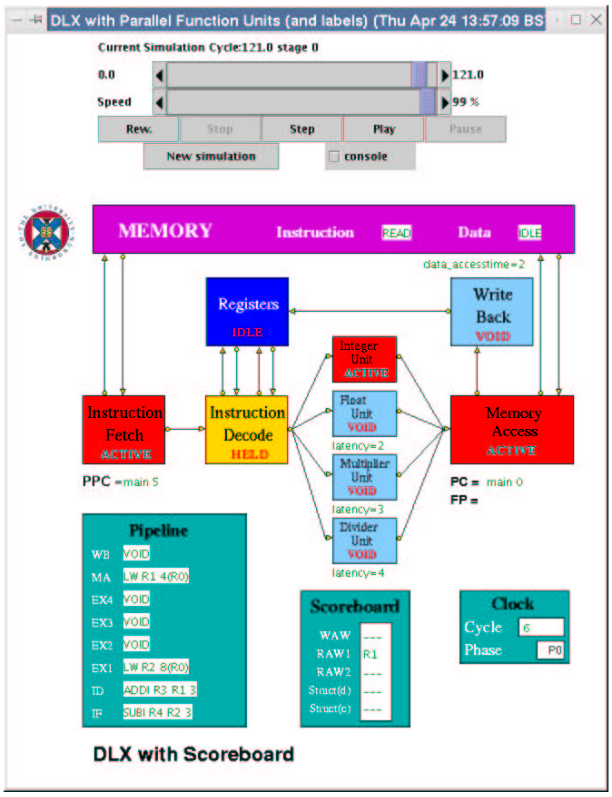
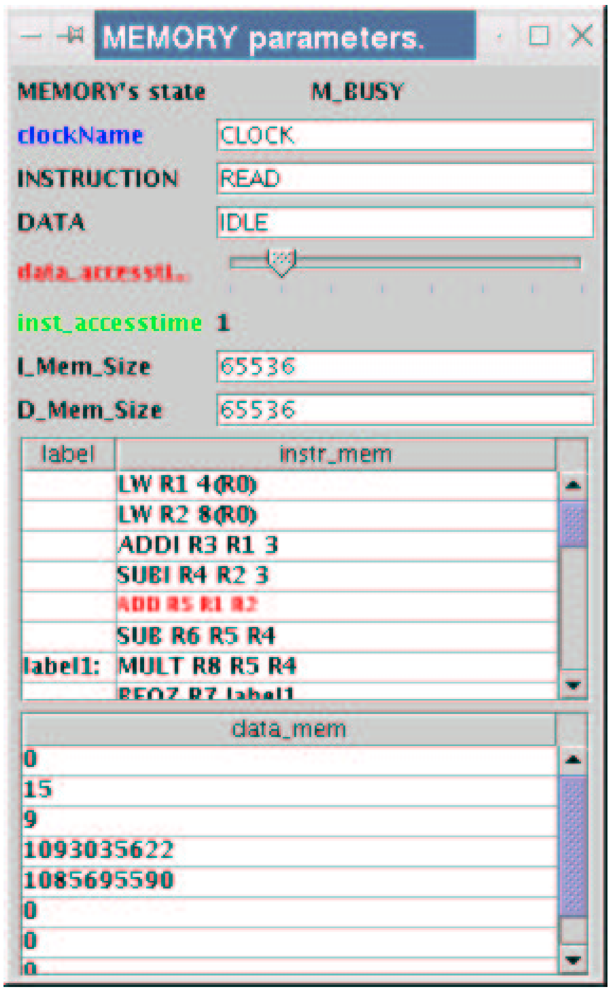


Figure 3: The JAVAHASE applet for the DLX model.

the trace file to extract the right information according to specific criteria.

In JAVAHASE, simjava-compatible files are generated according to the .hase descriptions. The dump_state() statements are converted so as to emit events to the external world. When running a JAVAHASE model as an application, this information is also written into a trace file. When using a JAVAHASE applet, however, these events are caught by graphical components, the layout of which is defined by the ELF file. At each instant, the applet shows the state and the parameters of the entities at the current simulation time. Any time a new simulation step is executed in Java, the graphical layout is updated in the applet (figure 3) while parameters (e.g. memories) are displayed in pop-up windows (figure 4).

A direct benefit of updating the layout after each step is that it allows interactive applets to be built in which parameters, and thus the behaviour, can be modified according to external inputs (*e.g.* mouse clicks). This interactive facility makes JAVAHASE suitable for robotics-oriented applications; this facility is currently being explored.


MEMORY parameters.

MEMORY's state	M_BUSY
clockName	CLOCK
INSTRUCTION	READ
DATA	IDLE
data_accesst...	
inst_accesstime	1
I_Mem_Size	65536
D_Mem_Size	65536

label	instr_mem
	LW R1 4(R0)
	LW R2 8(R0)
	ADDI R3 R1 3
	SUBI R4 R2 3
	ADD R5 R1 R2
	SUB R6 R5 R4
label1:	MULT R8 R5 R4
	BEQZ R7 label1

data_mem
0
15
9
1093035622
1085695590
0
0
0

Figure 4: The memory view with JavaHase.

CONCLUSION

We have presented the JAVAHASE system which allows us to translate HASE projects into Java applets which can be displayed in Java-compatible Internet browsers.

The major restriction of the JAVAHASE approach comes from the translation processes, from Hase++ to `Simjava` and from C++ to Java. Since Hase++ and `simjava` are very similar, however, the correspondence is quite straightforward. But, the Java compiler being more constraining than the GNU g++ compiler, we now have to take great care during the design of new projects to avoid some C-specific mechanisms (assignments between enumerated types, extensive use of pre-processor directives, *etc.*).

So far, we have successfully translated nine of the projects implemented in HASE (*e.g* [9, 10]) and any posterior modifications have been automatically reported to the applets.

We believe that the JAVAHASE approach has advantages over the WEBHASE approach. Even with a reliable servlet engine, the amount of data to be transferred for each new WEBHASE simulation would be prohibitive and the server would certainly be overloaded because it would have to run every simulation for each client. However, WEBHASE is still suitable for demonstrations in lectures, etc, and has the advantage of being less demanding in its browser requirements.

Obviously, HASE projects which make extensive use of external libraries will not be automatically translatable, they will require partial manual recoding. But, since we are able to mix C++ (.hase) and Java (.sjava) descriptions, some parts of the project may be rewritten directly in Java and use equivalent Java libraries.

All JAVAHASE applets are accessible from http://www.icsa.inf.ed.ac.uk/research/groups/hase/ where an access control mechanism allows us to quantify their usage.

Acknowledgments

JAVAHASE is being developed as part of a simulation study of the UKQCD computer architecture supported by the UK EPSRC under grant GR/R27129. WEBHASE was developed by Stuart Buchanan.

Biographies

Frédéric Mallet (fmallet@inf.ed.ac.uk) received his MEng and MSc degrees from the University of Nice in France in 1997 and his PhD in 2000. In 2001, he was Assistant Lecturer at the University of Nice. Since 2002, he has been Research Associate at the ICSA; he develops methods and tools based on the object-oriented paradigm to model and simulate computer systems architecture.

Roland Ibbett (rni@inf.ed.ac.uk) received his BSc and MSc degrees from the University of Manchester and his PhD from the University of Hull. From 1967 to 1985 he was a Lecturer, Senior Lecturer and then Reader at the University of Manchester. Since 1985 he has been a Professor of Computer Science at the University of Edinburgh. He is a Fellow of the Royal Society of Edinburgh and of the British Computer Society. His research interests are in computer architecture and in the simulation and visualisation of those architectures. While at Manchester he was a major contributor to the design of the MU5 computer. His use of the MU5 logic simulator, and involvement in the use of ISPS during a semester at Carnegie-Mellon University, identified a need for a higher level architecture simulator and thus led to the design and development of HASE.

References

[1] P.S. Coe, F.W. Howell, R.N. Ibbett and L.M. Williams: *A Hierarchical computer Architecture design and Simulation Environment*, ACM Transactions on Computer and Modelling Simulation, Vol. 8 No.4, 1998, pp. 431-446.

[2] N. Medvidovic and R.N. Taylor: *A classification and comparison Framework for Software Architecture Description Languages*, IEEE Transactions on Software Engineering, 26(1), Jan. 2000.

[3] F.W. Howell and R. McNab: *Simjava - a discrete event simulation package for Java with applications in Computer Systems Modelling*, Int. Conf. on Web-based Modelling and Simulation, SCS, Jan. 1998. http://www.dcs.ed.ac.uk/home/simjava/

[4] EECS, University of Berkeley: *Ptolemy2 - Heterogeneous Concurrent Modelling and Design in Java*, available at http://ptolemy.eecs.berkeley.edu/ptolemyII/.

[5] Silk inc.: *Silk - a Java-based modelling tool*, http://www.threadtec.com.

[6] A.H. Buss and K.A. Storh: *Discrete-event simulation on the world wide web using Java.*, proc. of the 1996 Winter Simulation Conference, pp. 780-785.

[7] B.A. van Halderen and B.J. Overeinder: *Fornax: Web-based discrete-event simulation in Java*, ACM 98 workshop on Java for High-Performance Network Computing.

[8] G.E. Karsner and S.T. Pope: *A cookbook for using the model view controller user interface paradigm in Smalltalk-80*, Journal of Object-Orientated Programming, 1(3):26-49, Aug./Sep. 1988.

[9] L. Williams and R.N. Ibbett: *Modelling the DASH architecture in* HASE, 29th annual Simulation symposium, New Orleans, July 1996.

[10] R.N. Ibbett: HASE *DLX Simulation Model*, IEEE, Vol 21, pp 24-33, Jan-Mar 1999.