ISHMAEL: AN INTEGRATED SOFTWARE/HARDWARE MAINTENANCE AND EVOLUTION ENVIRONMENT

James O. Coplien

James O. Coplien *is a member of technical staff in the Software Production Research Department at AT&T Bell Laboratories (Indian Hill Park) in Naperville, Illinois. Mr. Coplien does research in large system-development processes, and is also known for his work in object-oriented design. His recent work in system-hardware architectures to support software productivity for large real-time systems led to joint work with CAD organizations on the ISHMAEL environment. He joined the company in 1979 and has both a B.S. in electrical and computer engineering and an M.S. in computer science from the University of Wisconsin at Madison.*

This system-development environment is a companion to SysCAD that permits users to specify, prototype, simulate, develop, and document a complex digital system. ISHMAEL can manage the simultaneous evolution of a system's software and hardware. ISHMAEL users can develop faithful models of system performance that simulate the interaction of hardware and software at any level of detail. They can iterate hardware and software designs during a simulation, which permits realistic evaluation of design tradeoffs and implementation alternatives early in a system's life cycle and, thus, reduces system costs. ISHMAEL's integrated, graphical interface provides several views of the application being developed: a high-level or a detailed (i.e., schematic) hardware view, and the application-user's view. The system was used for prototyping, specification, evaluation, and implementation of a highly distributed, telecommunications system.

Introduction

ISHMAEL is an advanced test bed for life-cycle support of hybrid hardware/software systems. It is a *system*-development environment that can be used to specify, prototype, simulate, develop, and document complex digital systems through a single, uniform interface. ISHMAEL is optimized for system developments where the hardware and software are closely coupled, where new hardware is being developed to support a software application, or where details of system performance and cost must be known before a product is committed to manufacturing. (Panel 1 defines acronyms and terms used in this paper.)

ISHMAEL's first strength is its ability to manage the simultaneous evolution of a product's hardware and software. This unified product management goes all the way from system specification through design, to faithful system simulation and, finally, to preparation for

Panel 1. Acronyms and Terms

ACM	Association for Computing Machinery
C++	an object-oriented descendant of the C programming language
CAD	computer-aided design
CAE	computer-aided engineering
CLOCK	a graphical object used to monitor and control simulation time
EDN	*Electronic Design News*
GIL	graphical interface language; a high-level specification language tuned for graphical programming
glyph	graphical object; the boxes, icons, and other shapes on the screen, and their associated software
HIC	hybrid integrated circuit
hypertext	a multidimensional way of organizing, viewing, and editing on-line computer text
IC	integrated circuit
IEEE	Institute of Electrical and Electronics Engineers
ISHMAEL	integrated software/hardware maintenance and evolution environment
Lisp	a flexible, symbolic programming language
PC	personal computer
SCHEMA	an object that provides a subset of the functionality of the AT&T SysCAD schematic capture tool of the same name
SIGPLAN	ACM special-interest group on programming languages
SIMCED	simulation-circuit-model editor
SN74LS165	a shift-register integrated circuit
SysCAD	a set of common CAE applications for ICs, HICs, and circuit packs
VLSI	very-large-scale integration

system deployment. Such an approach is important in real-time system development to:
- Evaluate the interactions between hardware and software
- Evaluate potential tradeoffs between hardware and software design
- Develop accurate and detailed models of overall system performance.

It breaks from traditional approaches by providing for early testing and tuning of software against hardware designs, and for early adjustment of hardware designs in light of software behavior.

ISHMAEL's second strength is its support for iteration. During a simulation, both hardware and software designs can be iterated to provide a fast, convenient mechanism for realistically evaluating design tradeoffs and implementation alternatives.

The third strength of ISHMAEL is its human interface. Graphical animation is an emerging approach to system development[1] and is well suited to managing the development of large, complex systems. A flexible, integrated, graphical interface empowers developers to work in the context of a total system view.

By evaluating design tradeoffs early in the life cycle, we can reduce system cost. Effective use of ISHMAEL is possible across a broad spectrum of the life cycle. Thus, such tradeoffs can be evaluated by design engineers and programmers, as well as by architects and system engineers. Customer participation in evaluating these tradeoffs ensures a good match between what a customer expects and what is delivered.

ISHMAEL was created as an environment to support applied research on the system architecture for a highly distributed, fault-tolerant, real-time processing system named Logopolis. In order to chart the direction of the Logopolis project, we needed to evaluate high-risk hardware technologies and software approaches. But at the outset, we did not really understand how performance and cost could be best traded off between software and hardware.

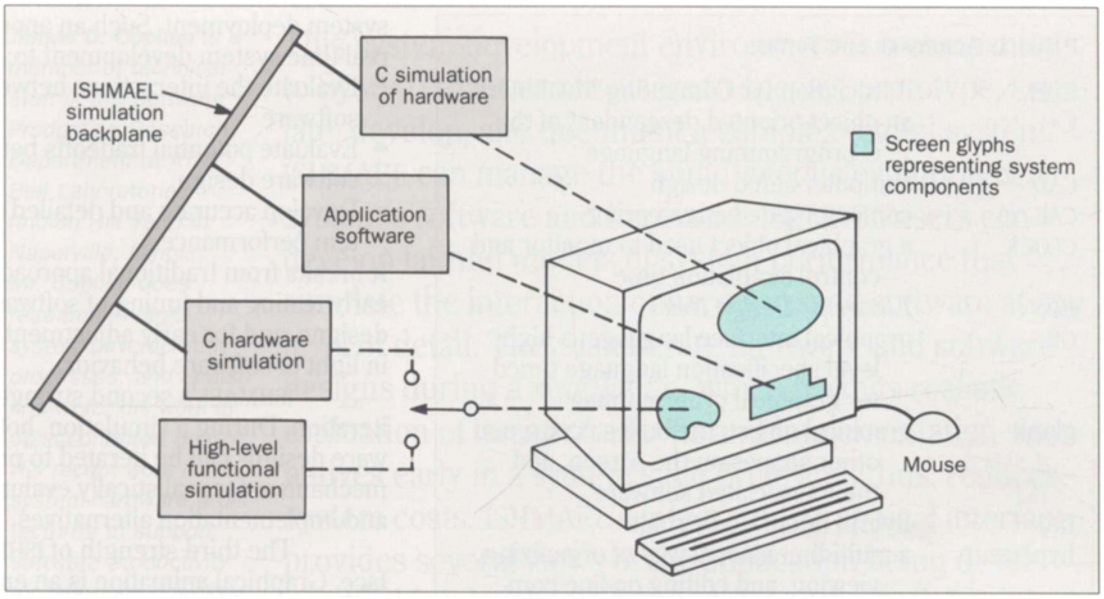

ISHMAEL simulation backplane
C simulation of hardware
Application software
C hardware simulation
High-level functional simulation
Screen glyphs representing system components
Mouse

Figure 1. A high-level view of ISHMAEL. Glyphs on an ISHMAEL design screen are the mechanisms through which users interact with the specification details of individual system components. These specification details may contain code to drive hardware simulation, application code for the target system, or user-defined design aids. The environment is tied together by a "software backplane" that allows hardware and software specifications to communicate flexibly with each other.

We saw a prototyping environment as a necessary risk-management tool. So, the Logopolis project created the ISHMAEL environment to meet its needs. SysCAD engineers collaborated on how to build ISHMAEL as an adjunct to the existing AT&T SysCAD framework, and many of its features are being introduced into forthcoming SysCAD releases. (SysCAD is a set of common computer-aided engineering applications for integrated circuits, hybrid integrated circuits, and circuit packs.)

The ISHMAEL Design Process

The ISHMAEL environment provides a common, graphical, user interface for system design and simulation. Before committing a design, developers can easily create a graphical specification of the system, simulate the operation of the system or of specific hardware or software elements, and determine the effect of design changes.

High-Level Design. To start an ISHMAEL design, the system designer captures the system structure in graphical form, using an interface similar to that of other interactive, picture-drawing programs. However, what the user is really doing is creating the framework of a *graphical specification*, a picture that captures the semantics of the application. For example, the containment of one box within another in the picture corresponds to the logical or physical containment of one part within another in the design. This way, abstraction hierarchies can be built up, either in the sense of a layered design or in the sense of building large components out of smaller ones. The resulting design captures the hierarchy of the major system components.

The boxes, icons, and other shapes on the screen in Figure 1 are called *graphical objects* or *glyphs*. Here, the term *object* means an abstraction that has the "intelligence" to model how its counterpart behaves in the real world. Glyphs aren't just passive shapes. They are active, intelligent agents that are used as building blocks during the architecture and design phases of a project. Glyphs are the software analogy of "plug-compatible parts," which behave like finite-state machines with well-defined input and output interfaces.

These diagrams aren't created just for the sake of drawing pretty pictures, and design is not just the act of making diagrams.[2] The picture encodes the system structure and drives downstream development. Each glyph can be tied to a low-level design or implementation construct. Thus, using a mouse to select glyphs is one way a user can navigate lower levels of the system. To reciprocate, the low-level constructs may use their glyphs to display their status, or to gather interactive input from a user during system simulation.

Capturing the Design. After the major structure is in place, a system designer may say more about how the pieces behave individually and how they interact with each other. There are two ISHMAEL environments for doing this:

- A set of extended *SysCAD tools* that are based on the standard AT&T hardware computer-aided design (CAD) platform
- GIL (for *graphical interface language*), a high-level specification language that is tuned for graphical programming.

The SysCAD tools. Most system components selected for hardware implementation are specified in detail using the SysCAD tools. SysCAD provides a rich tool set for schematic capture, design verification, and circuit manufacturing. It has an extensive catalog of both standard-handbook hardware components and custom integrated circuits.

As with architectural specification, most ISHMAEL hardware designs make extensive use of hierarchies. At the highest level of hardware design, designers can envision an integrated circuit or "chip" that performs some function and can specify the chip's pin-out and external interfaces. Also, timing requirements might be captured in these components' design specifications at this level.

The same technique can be applied inside each chip until a low enough level of detail is reached that permits fully specifying the implementation in terms of existing hardware building blocks (i.e., gates, polycells, or other components, depending on the technology being used). Such a design can be specified with components from the SysCAD catalogue of logic devices, using traditional schematic-capture techniques. Alternatively, any chip's behavior can be fully specified in a low-level, C-programming language, software model (which we call the *C-language model*). This capability is particularly useful for rough characterizations of hardware devices whose external behavior is intuitive, but whose internal behavior is tedious at the gate level (for example, memory devices).

GIL—a high-level specification language. GIL[3] is a powerful specification language that has the flexibility of the Lisp language and many features of the Smalltalk-80® language to support object-oriented programming. (Lisp's main focus is symbolic processing. Besides flexibility, the language is known for the ability of its programs to modify themselves at run time. Smalltalk-80 is a registered trademark of Xerox Corporation.)

GIL is used in three different ways in ISHMAEL:

- GIL can be viewed as a system-application language in its own right, e.g., as the language in which the product will be programmed. It is a high-level programming language that runs under an interpreter, but can be transformed into compiled code with good run-time performance.
- Plug-compatible parts are the major abstractions in

GIL, where functions behave like combinatorial logic gates, and where the interactions between functions behave like signals on a wire. This makes GIL a natural language to model a complex, combinatorial logic function inside a new chip. Idioms in the language can also be used to simulate devices that have states, so almost any hardware function can be conveniently coded in GIL.
- Because GIL is easily interfaced with detailed software and hardware specifications and is rich in graphics primitives, it is the language that holds the entire environment together. GIL's flexibility makes it possible for users to extend, customize, or tune the environment in any dimension they wish.

The software backplane. The hardware and software designs are tied together by a *software backplane.* The job of this backplane is to give software the appearance of talking to real hardware, and to give the hardware being simulated the appearance of interacting with software.

This backplane is not part of the application design, but is an abstraction that is an artifice of the ISHMAEL simulation-support environment. The backplane defines a protocol, and is implemented through a set of library routines that are accessible from both hardware and software design entities. These routines can be bound with code in popular programming languages, such as C. It is through these mechanisms that C-application code or C-language models of hardware logic operate.

ISHMAEL Design and Simulation Environment

ISHMAEL provides a highly interactive environment for simulating system hardware, application software, and the interactions between them. Users interact with the system by using a mouse and the keyboard. The mouse can be used to point to glyphs of interest, and the keyboard to enter data. All user interaction with the application, simulation, and design environments for both hardware and software development takes place through a single, uniform interface.

Each glyph on the screen can map onto one or more components in the system design. For example, a hardware register may be specified as part of the high-level design, and may be represented in the implementation as a SN74LS165 integrated circuit. The screen glyph of such a register might display the register's internal bit pattern at any given time.

User-defined glyphs may have windows, called *scrolls*, for data entry. A user can position the cursor on a scroll with the mouse and, then, enter data on the keyboard. The keyboard data are gathered in the scroll and echoed as they are typed. Glyphs can use these scroll data to inject logic signals into the hardware simulation, change data in the application software, inquire about system status, and so on.

The appearance of the glyphs can change as a function of hardware or software states. Thus, from a glance at the display, an ISHMAEL user can tell the overall state of the application system. For example, a line that represents a circuit net (i.e., a wire) might display the net's current logic value by coloring itself green for logic true, red for logic false, and yellow if the net is electronically isolated. A software variable can display its value in a glyph, and the environment will automatically update the display whenever the variable's value changes.

Besides these general facilities for user-defined components, the environment has some "stock" tools as well:
- A CLOCK glyph
- A log for hardware events
- The SCHEMA object, which provides graphical test access to a hardware-circuit simulation
- A software and hardware logic analyzer
- A run-time, incremental loader for circuit models
- A recording and playback facility that permits reproducing a design or simulation session.

A brief description of these tools follows.

The CLOCK Glyph. Figure 2a shows the CLOCK glyph, which is used to monitor and control simulation time. This glyph shows the current simulation time in nanoseconds, as well as the current time of day. It has *go* and *stop* "buttons" that control the system hardware clock (i.e., the signal that drives all synchronous

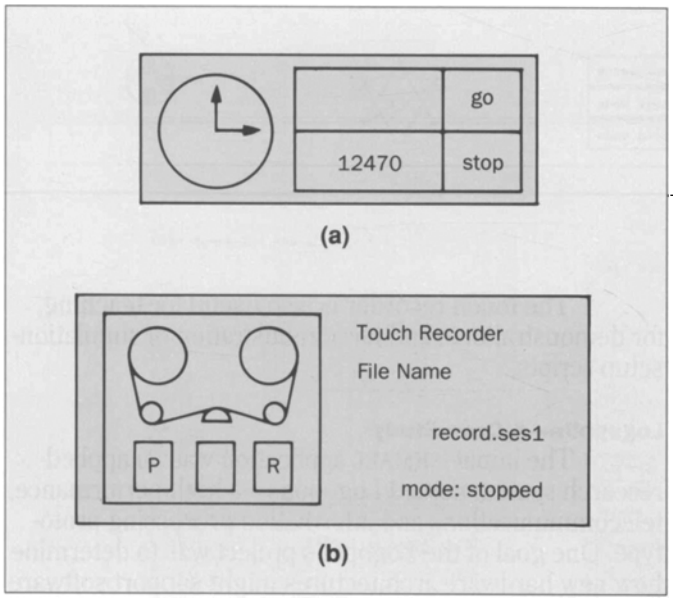


Figure 2. A glyph—an active, intelligent agent—models the behavior of its real-world counterpart. (a) The CLOCK glyph shows the current simulation time in nanoseconds, as well as the current time of day. Its go and stop buttons control the system hardware clock, and can be used to schedule fault events during hardware simulation. (b) The touch-recorder glyph can record all user interactions in a design session or a simulation, and can play them back later. A sequence of recording events can be scheduled using the CLOCK glyph, or the user can interact with the recorder in a live simulation. Thus, a design or simulation problem is easily reproduced.

hardware logic).

The clock can also be used to schedule fault events in the hardware simulation. By stopping the clock, programming a time into the top window, and typing a **1** or a **0** over a logic net, a developer schedules that net to take on the supplied logic value at the specified time.

The Hardware-Event Log. This is a scrolling window that is used to log high-level system events, such as the transfer of data between registers or the arrival of information on a data link. For each event, the window presents relevant data and simulation-clock time.

The SCHEMA Object. This tool provides a subset of the functionality of SCHEMA, the AT&T SysCAD schematic-capture tool. The role of the SCHEMA object is to provide graphical test access to hardware-circuit simulations, a facility not available in the schematic-capture tool. During simulation, a "soft logic analyzer" (which is described below) can monitor circuit-logic values in a SCHEMA circuit diagram.

Most of the SCHEMA object is written in GIL. However, some of the code is in C++ to do format conversion between SysCAD files and the GIL internal format. (C++ is an object-oriented descendant of the C-programming language. Both languages were developed at AT&T Bell Laboratories.) The SCHEMA object was written as an application object, with only loose coupling to the ISHMAEL base. Thus, during a simulation run, the SCHEMA object can be loaded (or unloaded) without stopping or restarting the simulation process.

The rapid prototyping facilities of GIL made it possible for a single engineer to create a working SCHEMA object in less than one working week.

A Soft Logic Analyzer. The software and hardware logic analyzer behaves like a laboratory-bench logic analyzer. That is, it monitors the value changes of the components to which it is connected. We call this a "soft" analyzer because its implementation is purely software.

A developer can use the soft logic analyzer just like a hardware laboratory-bench analyzer to graph digital waveforms of hardware signals. But he or she can also use it to monitor changes in some classes of software values in a simulation.

The analyzer consists of several channels; each channel monitors one hardware or one software value. Channels can be created dynamically as "plug-in" modules during simulation, and can be interactively connected during simulation. Connections can be made to any hardware signal depicted on a SCHEMA diagram, or to any user-defined screen representation of a hardware signal. When the soft logic analyzer is connected to a software object, the object will arrange to redisplay its contents in the analyzer channel whenever its value is overwritten.

Run-Time Incremental Loader. A C-programming language function (i.e., a C-language model) may be used to specify the simulation behavior of a complex hardware component. Models for common catalogue components are supplied in the standard SysCAD library. However, users can define their own chips, too, using a tool called SIMCED (simulation-circuit-model editor).

Under ISHMAEL, a user can invoke SIMCED in the middle of a simulation to replace any C-language model with an updated version. With this facility, a user can change the timing characteristics or logic of any chip for which a C-language model is available.

Doing these modifications incrementally results in greatly compressed turnaround intervals for component changes. It eliminates the overhead of reconfiguring and initializing the simulation, which can take several minutes for large circuits.

This flexible update approach is based on *incremental loading*, a technique used in software-development environments to support rapid iteration of designs. For further examples of how incremental loading can support iterative development, see the paper by Schmidt, Kowalski, and Smull in this issue[4] or by Kowalski, Huang, and Diamantidis.[5]

Session Recording and Playback. The touch-recorder glyph (Figure 2b) can be used to record all user interactions with a design session or a simulation and then play them back at a later time.

Graphical session-recording facilities offer simulation control and monitoring that are more powerful, intuitive, and convenient than textual interfaces can provide.[6] By selecting the touch recorder's RECORD (R) button with the mouse cursor, the user can start the capture of a sequence of stimuli. He or she then can use the CLOCK glyph to set a simulation time, and can use the keyboard and mouse to create events such as stuck-at faults, initiation of monitors, or the transmission of data to a software component.

The touch recorder can capture a live simulation as well. The stimuli can be stored, retrieved, spliced, and played back later, either to reproduce a scenario or to evaluate how modifications to the system under development change its reaction to the same set of events.

Playback can be continuous, where the playback speed is a programmable parameter, or "frame-by-frame," one event at a time.

The touch recorder is also useful for teaching, for demonstrations, and for administration of simulation-setup scripts.

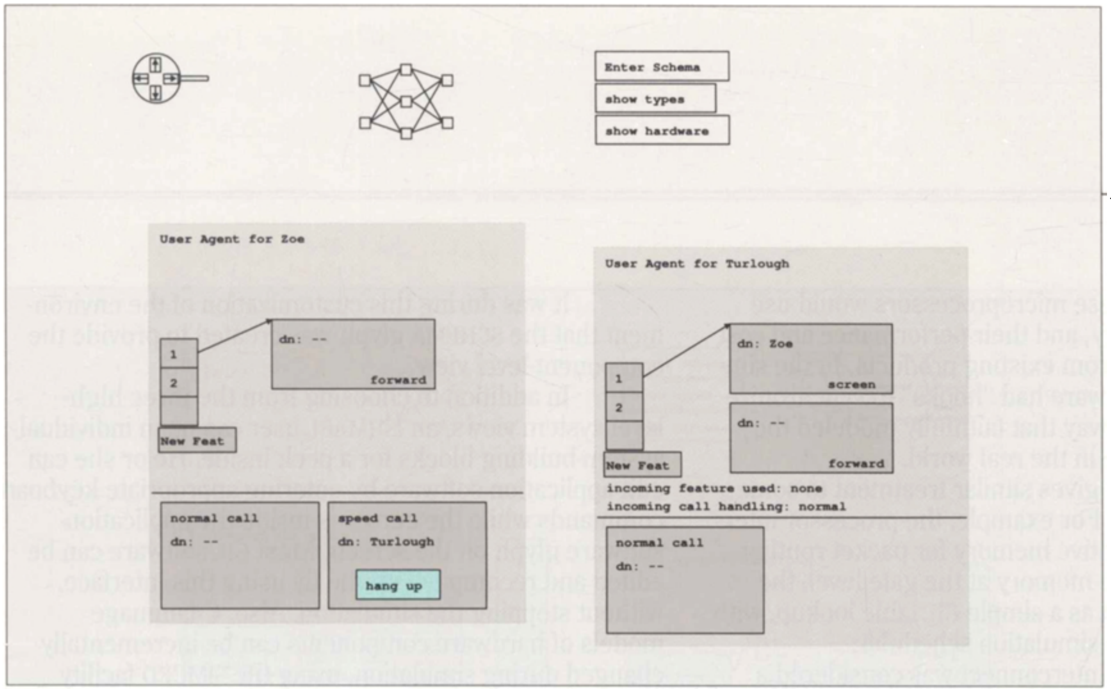
Logopolis—A Case Study

The initial ISHMAEL application was an applied-research system named Logopolis—a high-performance, telecommunications and information-processing prototype. One goal of the Logopolis project was to determine how new hardware architectures might support software development, with particular emphasis on long-term evolution and overall configuration flexibility. The Logopolis system was based on unconventional data-flow hardware, using highly distributed intelligence.

Logopolis was an exploratory development with higher risk technology than is found in current product developments. Uncharted software and hardware approaches were used that, if proven fruitful, would have high payoff. The need to explore and refine these technologies, for which few design rules existed yet, suggested that substantial system-prototyping support was needed. The prototyping needed to be done at a low enough level that we could convince ourselves and our customers that a system built on these principles would perform well and not cost too much. However, the prototyping approach would need to turn designs around faster and cost less than traditional "breadboard" approaches (i.e., where we build a throwaway, hardware prototype).

Another key element in the development was the need for ongoing interaction with customers so they could evaluate the suitability of the system to their needs and evaluate our progress over time.

ISHMAEL was built as the "software breadboarding" environment for the Logopolis effort. With ISHMAEL, we could develop faithful models of system performance at fine degrees of resolution, because we could simulate hardware at any level of detail in the context of the software with which it was interacting. The ability to simulate at gross levels of detail was as important as being able to simulate at the fine levels. It better conveyed the



overall patterns of information flow through the Logopolis system.

All simulations could be controlled and monitored from a graphical control console on a workstation. Different consoles were constructed for different classes of users; e.g., project management, systems engineers, and even the ultimate users of both the application system and the ISHMAEL environment.

High-Level Design of the Application. Initial high-level design of the application was done purely in GIL, with two goals in mind:

- To study user interactions; i.e., to evaluate how telecommunications customers would interact with the product
- To set a framework for the more-refined Logopolis specification that was to follow.

Figure 3 illustrates the human interface to a design. We did the design in Figure 3 without regard to the hardware or software implementation; instead, we worked at the higher level of information flow between objects. Thus, the hardware, the software, and their interactions with each other and with the outside world were specified in GIL. Because the graphical program was executable, a user could make trial calls, develop and exercise new features, etc.

The ability to simulate at this level without any

Figure 3. The user view of the application. The large, gray boxes are *user agents*, the graphical objects that embody the expertise for handling individual user's telephones. Embodied within each user agent are boxes that contain features; features for call origination are in the bottom half of the box, while those used by incoming calls are in the top half. Icons appear on the screen for the hardware-event log, logic analyzer, clock, and SIMCED interface and to depict the states of some internal-routing logic as well. This picture is just one view; other views can be made visible with the three rectangular "buttons" at the top.

substantial commitment to hardware implementation reduced development cost. Iterating simulations up front greatly reduces the need to iterate implementations late in the project, when hardware design changes can be very expensive. Representative customers gave feedback on how the interface was to operate, and on how Logopolis' telephony features were to interact. Changes could be iterated in GIL literally in seconds, so an operational specification of the Logopolis system converged rapidly.

Separating Hardware from Software. The next step was to allocate Logopolis system functionality onto hardware and software. A major organizing principle was to map each telephone set onto a set of microprocessors; each microprocessor provided an individual call-processing feature. These microprocessors would use conventional technology, and their performance and cost could be extrapolated from existing products. In the simulation, application software had "hooks" to synchronize with the hardware in a way that faithfully modeled the software's performance in the real world.

The simulation gives similar treatment to some hardware components. For example, the processor interconnect uses an associative memory for packet routing. Instead of modeling the memory at the gate level, the simulation represents it as a simple GIL table lookup, with timing interfaces to the simulation scheduler.

The processor interconnect was considered a high-risk component from the perspectives of both performance and cost, so detailed design and simulation were in order. The Logopolis project used SysCAD SCHEMA to capture the design as a schematic, using gate-level components from a standard on-line library.

Customizing the Environment. To help designers navigate the complex Logopolis model, which comprised several layers of both hardware and software, the Logopolis project customized the environment to support multiple views of the same design. By touching a glyph in one view, an ISHMAEL user could bring up another window with another view, in a way reminiscent of hypertext systems. Applications engineers programmed this multiview selection, as well as other project-specific design and analysis facilities, into the base ISHMAEL environment. Because ISHMAEL is an open environment with flexible update capabilities, an engineer can—in a matter of days or even hours—create an interface that gives the appearance of being an entirely new environment.

By using the mouse to "push" a button on the screen, an ISHMAEL user can choose from among three views of the system:

- A customer (or user-level) view of the application (Figure 3)
- A high-level (register-transfer level) hardware view (Figure 4)
- A component-level hardware view (Figure 5).

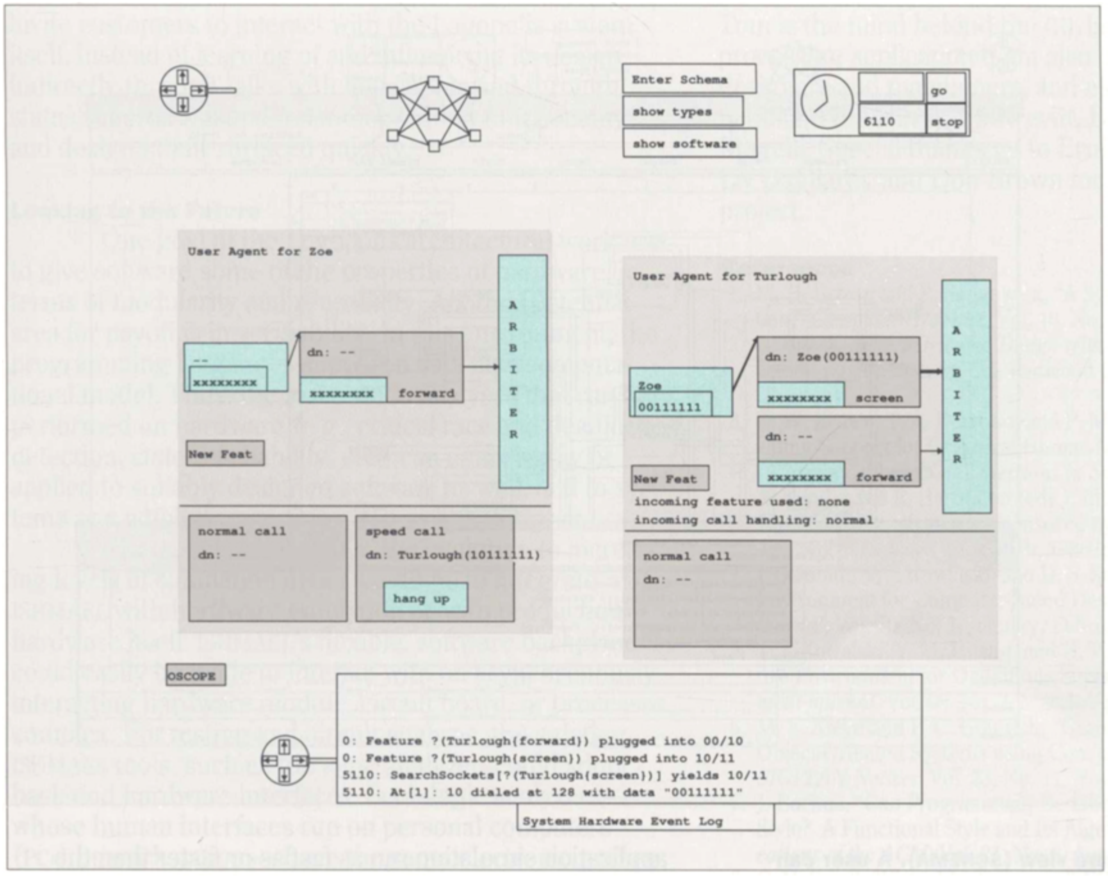
It was during this customization of the environment that the SCHEMA glyph was created to provide the component-level view.

In addition to choosing from the three high-level system views, an ISHMAEL user can open individual system-building blocks for a peek inside. He or she can edit application software by entering appropriate keyboard commands while the cursor is inside the application-software glyph on the screen. Most GIL software can be edited and recompiled on the fly using this interface, without stopping the simulation. Also, C-language models of hardware components can be incrementally changed during simulation, using the SIMCED facility described earlier.

Another characteristic that we put under user control was the level of simulation fidelity. The ISHMAEL user could select either a high-level, software simulation of hardware-data flow or a low-level, nanosecond granularity, hardware simulation. The former was largely used for application-feature evaluation or for call-scenario analyses.

An ISHMAEL user can create custom support tools on the fly. For example, audit objects exist to monitor hardware-signal levels and software-variable values. Such objects might look for any simultaneous set of conditions that violate design criteria on functionality or performance, and respond to such a condition with a user-defined action. These custom tools are easy to write; the condition of interest is encoded in GIL as a Boolean expression, written in terms of simulation values. A monitor object might send a request to the CLOCK object to stop the simulation if it notices any logically inconsistent, concurrent values.

Experience with the Prototype. Logopolis developers who worked with the ISHMAEL prototype had, at their disposal, analogues of most of the tools found in a system lab—but at a fraction of the cost. Designs could be breadboarded with software and "installed" into a running simulation of the entire system, analyzed on the spot, and sometimes modified (i.e., "white-wired") in the

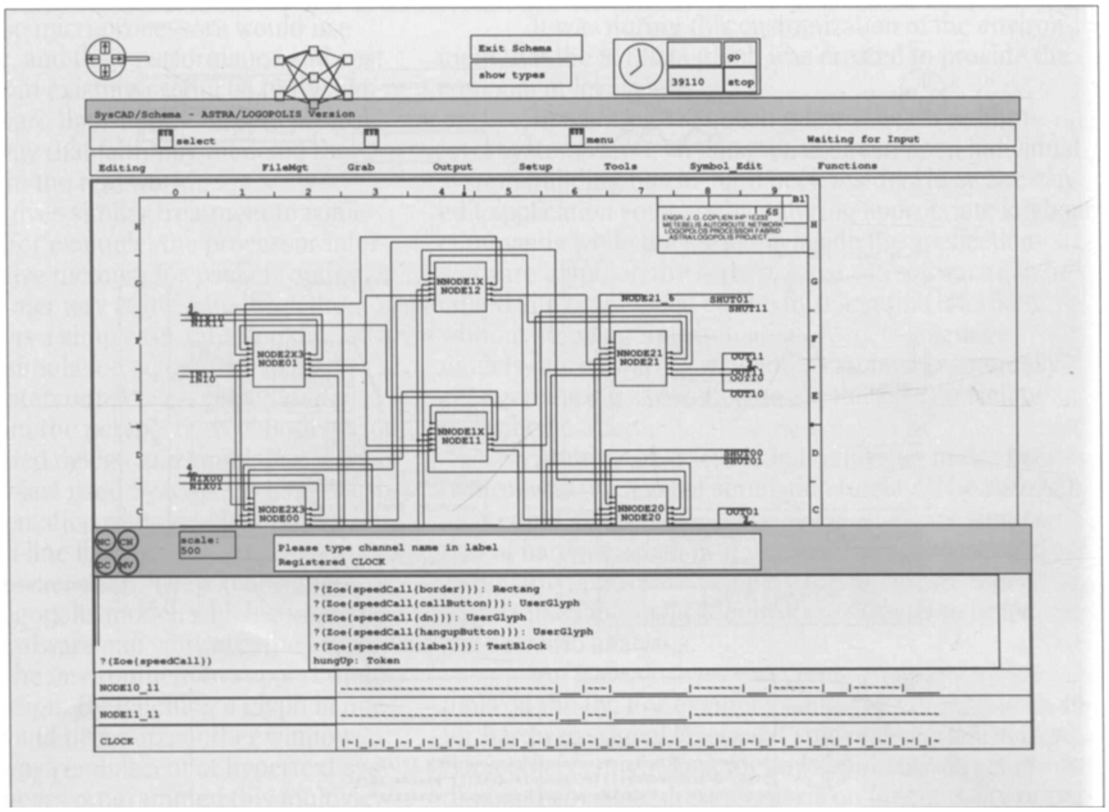


```
0: Feature ?(Turlough(forward)) plugged into 00/10
0: Feature ?(Turlough(screen)) plugged into 10/11
5110: SearchSockets[?(Turlough(screen))] yields 10/11
5110: At[1]: 10 dialed at 128 with data "00111111"
```

Figure 4. High-level hardware view. This screen shows ISHMAEL configured to depict the high-level (register-transfer level) hardware view of the Logopolis prototype. Registers (within the user agents) can be seen displaying their bit-string contents. An icon of the internal network can be seen at the top left; it displays the status of the links between network nodes. A hardware-event log appears at the bottom of the screen. "Wires" that connect registers to processing nodes flash when bits travel over them: red for 1, blue for 0. The logic analyzer can be connected to wires in the network icon or between the processing elements.

middle of a simulation. The simulation faithfully reproduced many problems related to timing and asynchrony that a real prototype would demonstrate. However, ISHMAEL's touch-recording facility made the problems reproducible, and its logic-analyzer facilities gave a complete history of signals leading up to the failure modes.

One drawback was the tradeoff between faithfulness and the speed of the simulation. A faithful simulation with nanosecond granularity ran about a million times slower than real time, so a millisecond of real time could be simulated in about 15 minutes. A high-level



```
Please type channel name in label
Registered CLOCK
```

```
?(Zoe(speedCall(border))): Rectang
?(Zoe(speedCall(callButton))): UserGlyph
?(Zoe(speedCall(dn))): UserGlyph
?(Zoe(speedCall(hangupButton))): UserGlyph
?(Zoe(speedCall(label))): TextBlock
hungUp: Token
?(Zoe(speedCall))
```

```
NODE10_11   ------------------------------|___|---|___________|-------|___|-------|___|-------|_____
NODE11_11   ------------------------------|___|---|___________|------------------------------------
CLOCK       |-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-|_|-
```

Figure 5. Detailed hardware view (SCHEMA). A user can inspect the architecture at its lowest level: gates, wires, and buses. This schematic depicts a subnetwork used in the Logopolis processor interconnect; each of the seven major boxes is a user-defined chip that would be manufactured using VLSI techniques in a product application. (Here, the bottom window covers the seventh chip's image.) Each such subnetwork can be opened up and examined; this circuit is at the top of a hierarchy that is five levels deep. Also shown is the hardware/software logic analyzer with several hardware traces opened up. One software channel has been connected; it depicts the values of a complex data structure in a scrollable window.

application simulation ran as fast as or faster than the application itself would, but provided little insight into system-timing issues.

Overall, the ISHMAEL environment was a boon to productivity. It was "friendlier" than a hardware-system lab and allowed the creation of tools that would be impossible to build outside the domain of a software simulation. The ISHMAEL environment provided a basis for proving in technology without the expense and delay of VLSI fabrication or circuit-board manufacturing, greatly reducing the product's time to market.

The ISHMAEL environment presented the Logopolis system to its users at many levels. Thus, we could invite customers to interact with the Logopolis system itself, instead of learning of and influencing its design indirectly through talks with developers and through status reports. Misunderstandings about functionality and design intent surfaced quickly.

Looking to the Future

One goal of the Logopolis architectural work was to give software some of the properties of hardware, in terms of modularity and reusability. Another potential area for payoff is in verifiability. In this environment, the programming languages support a data-flow computational model. Thus, the same static analyses that can be performed on hardware (e.g., critical race and deadlock detection, state reachability, etc.) can conceivably be applied to suitably designed software as well, and to systems as a whole.[7]

The next logical step in the evolution to increasing levels of simulation detail would be to integrate ISHMAEL with hardware emulation or with production hardware itself. ISHMAEL's flexible, software backplane could easily be made to interact with an asynchronously interacting hardware module, circuit board, or processor complex. For testing and circuit analysis, the existing ISHMAEL tools, such as the logic analyzer, could adopt back-end hardware interfaces. Such logic analyzers—whose human interfaces run on personal computers (PCs) or workstations—have the same electronic probes as conventional logic analyzers and are called *virtual instruments*. Hardware that supports this approach is readily available on the market today, backed by IEEE interface standards.[8]

The output side of ISHMAEL can be extended as well. A straightforward extension to the environment would be to provide a multiscreen interface to a simulation, with different screens to represent different views.

Acknowledgments

I am especially indebted to Thomas A. Burrows for his contributions to the software side of ISHMAEL. Tom is the mind behind the GIL language and the call-processing application. I am also indebted to SysCAD designers and maintainers, and especially to Tom Schultz, Phil Gilles, Jim Krevitt, Bob Michael, and Paul Magelli. Special thanks go to Ernie Frey, Rita Ullrich, Liv Gagliardi, and Don Brown for their support of this project.

References

1. M. H. Brown and R. Sedgewick, "A System for Algorithm Animation," *Computer Graphics*, Vol. 18, No. 3, July 1984, pp. 177-186.
2. G. Booch, *Object-Oriented Design with Applications*, Benjamin-Cummings Publishing Co., Redwood City, California, 1990, Chapter 5.
3. D. W. Brown, T. A. Burrows, and P. M. Zislis, "GOS—A Tool Providing Support for Graphical Human-Machine Interfaces," *Digital Communications: New Directions in Switching and Networks*, A. Kundig and R. Hartmann (eds.), International Zurich Seminar on Digital Communications, sponsored by Swiss Institute of Technology, North-Holland, New York, 1986, pp. 197-204.
4. J. Schmidt, T. J. Kowalski, and D. S. Smull, "An Incremental Environment for Computer-Aided Design Tools," *AT&T Technical Journal*, Vol. 70, No. 1, January/February 1991, pp. 101-110.
5. T. J. Kowalski, Y. M. Huang, and H. V. Diamantidis, "An Interpretive Environment for Operations Support Systems," *AT&T Technical Journal*, Vol. 69, No. 2, March/April 1990, pp. 42-51.
6. M. F. Kleyn and P. C. Gingrich, "GraphTrace—Understanding Object-Oriented Systems using Concurrently Animated Views," *SIGPLAN Notices*, Vol. 23, No. 11, November 1988, pp. 191-205.
7. J. Backus, "Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs." *Communications of the ACM*, Vol. 21, No. 8, August 1978, pp. 613-641.
8. C. H. Small, "Virtual Instruments," *EDN*, Vol. 33, No. 18, September 1, 1988, pp. 120-128.

(Manuscript received October 23, 1990)