

# Integrated Simulation of Performance Models and Behavioral Models

Sanjaya Kumar, Fred Rose  
Honeywell Technology Center  
3660 Technology Drive  
Minneapolis, MN 55418

{skumar, rose}@htc.honeywell.com

## Abstract

*Hybrid modeling is a technique that can be used to integrate performance models and behavioral models within a common simulation. This approach allows behavioral components, containing mixtures of hardware and software, to be evaluated within the context of the system being developed. Hybrid interfaces are required to integrate the behavioral components with the performance models.*

*This paper presents Honeywell's VHDL-based approach to hybrid modeling. The structure of the hybrid interface is described, and a library of hybrid interfaces, the Hybrid Model Library (HML), is presented. The Hybrid Interface Generation (HIG) toolkit, a toolkit to aid in the creation of hybrid interfaces, is also presented. We are in the process of evaluating our hybrid modeling methodology. This work is briefly discussed.*

*Keywords: hybrid modeling; hybrid interfaces; performance evaluation; models; abstractions; multi-level modeling.*

## 1. Introduction

Several problems exist in the design of complex systems. Currently, there is a separation between those who develop the system architecture and those who create the detailed hardware/software implementation for the system. This separation leads to the model continuity problem [1], the inability to refine a system level model into a hardware/software implementation. Another problem is the failure to appreciate the subtleties associated with integrating subsystems. For example, the 90/50 rule [2][3] states that although 90 percent of ASICs work the first time, only 50 percent work properly in the system. Also, the increasing complexity of systems mandates the use of methodologies and techniques which support design and analysis at multiple levels of detail.

Hybrid modeling [4][5], the simulation of performance (uninterpreted) models and behavioral (interpreted) models in a common environment, attempts to address these problems. This modeling technique allows behavioral components, containing mixtures of hardware and software, to be evaluated within the context of the system being developed. Hybrid interfaces are required to integrate the behavioral components with the performance models.

This paper presents Honeywell's VHDL-based approach to hybrid modeling. The structure of the hybrid interface is described. A library of hybrid interfaces, the Hybrid Model Library (HML), is discussed. These architectures are hybrid versions of various performance modeling constructs contained in the Honeywell Performance Model Library (PML), a library of VHDL-based modules used for performance analysis that employ tokens. The Hybrid Interface Generation (HIG) toolkit, a toolkit to aid in the creation of hybrid interfaces, is presented. A hybrid modeling example is provided to illustrate the ideas in the paper.

We are in the process of evaluating our hybrid modeling methodology using an internal Honeywell application. The objectives are to better understand the benefits of hybrid modeling, beyond those discussed earlier, and to evaluate our current approach. This effort is also discussed.

The remaining portions of the paper are organized as follows. Section 2 briefly describes the PML environment. Section 3 provides background material. Section 4 discusses the HML hybrid interface. Section 5 presents the HML. Section 6 describes the hybrid interface generation toolkit. A hybrid modeling example is provided in Section 7. Section 8 summarizes the contributions of the work.

## 2. The Performance Model Library

The Honeywell PML [6][7] was created to fill the analysis needs required for the design of large, distributed, embedded real-time systems. This library is being incorporated into the commercial Performance Modeling Workbench (PMW) [8] product being developed by Omniview, Inc. Design features of this library include processing elements, communication components (routers, crossbars, etc.), system input/output and storage, topology, software partitioning, allocation, and scheduling. Typical design parameters include the entire software design or architecture, system software, and portions of the hardware.

The PML utilizes standard commercial VHDL capabilities. As a result, the library allows a system architect to capture the system under study in a consistent, verifiable form. Standard output routines tabulate and graph performance statistics such as utilization and latency.

Three primary classes of library elements are used in the PML: the leaf cells, the communication cells, and the processor model. All of these modules communicate through the use of tokens [9]. These library elements are described in more detail below.

The leaf cells provide basic token manipulating functionality that can be inserted into performance models quickly and easily. Such functionality includes token creation (indevice), token consumption (outdevice), token delays (pipeline, fifo, iodevice), and token routing (split, join).

The communication cells are used to model particular communication protocols. Currently, the communication cells include a VME component for multi-drop buses, a set of Mercury RACEway [10] components that model circuit switching communications, and a collection of Myrinet [11] components that model packet switching (store and forward) communications.

The processor model facilitates hardware/software codesign and coanalysis [12][13]. It consists of three parts: software models or tasks, a scheduler or thread manager, and the hardware core. The processor model provides a powerful software modeling capability over a wide range of modeling levels. Additional features have been added to handle interrupts, preemptive tasking, task communication, task synchronization, and other services. The scheduling model supports static and dynamic tasking, and even rate monotonic scheduling.

## 3. Hybrid Modeling Background

This section provides some background on hybrid modeling. The first section discusses the different classes of hybrid models. The next section describes some interfacing scenarios, and the last section discusses where hybrid modeling can be used within a design process. A more detailed discussion of these topics, particularly the first two, can be found in [5].

### 3.1 Hybrid Model Classes

A hybrid model consists of both performance (uninterpreted) and behavioral (interpreted) models. In order to integrate these two different types of models within a common simulation, a hybrid interface is required which resolves the discrepancies between these two domains.

The technique for developing hybrid models depends on the class of modeling problems being solved. The classes of hybrid modeling are defined by those model attributes which fundamentally alter the implementation of the hybrid interface. These attributes are described below.

- **Hybrid model objective.** Two major objectives are: 1) performance and timing verification, and 2) functional verification.
- **Timing and synchronization mechanism.** This attribute defines the timing and synchronization mechanism across the hybrid interface. The two types of system models are synchronous and asynchronous.
- **Nature of the interpreted model.** Interpreted elements can be hardware components or possibly mixtures of hardware and software.
- **Data transformation mode.** The mode defines the data transformation/interface mechanism. Specifically, the values for the interpreted signal inputs can be generated using the following techniques (among others): translate, stochastic, external, and files.
- **Data type of the interpreted signals.** The data type defines the interpreted data generated or received in the hybrid interface. Examples include bit vectors, integers, and reals.

### 3.2 Interfacing Scenarios

In addition to the attributes described above, several possible interfacing scenarios exist. The interfacing scenarios define the data flow between the uninterpreted and the interpreted domains. Four interfacing scenarios are possible.

These scenarios are illustrated in Figure 1. Some scenarios transfer data across a single boundary (U/I, I/U), and others move data from one domain and then back (U/I/U, I/U/D). The U/I/U and I/U/I scenarios show the insertion of a model described at a different level of detail into an existing model. In such a case, the inserted model is surrounded by a hybrid interface.

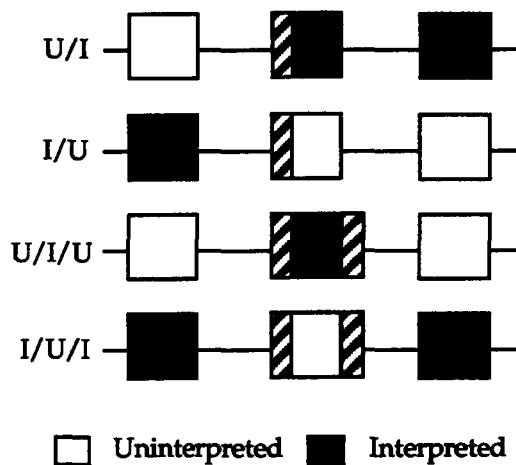


Figure 1. Interfacing Scenarios [5]

During a typical top-down process, the U/I/U scenario is very likely since an uninterpreted model is constructed followed by the insertion of various interpreted elements. This scenario enables one to preserve token information across the hybrid interface. Whenever the interpreted model is surrounded by uninterpreted elements, it is regarded as a U/I/U interfacing scenario.

### 3.3 Relation to Design Process

Figure 2 shows the Rapid Prototyping of Application Specific Signal Processors (RASSP) design process [14][15], as defined by Lockheed Martin Advanced Technology Laboratories. The bold, solid box indicates where performance modeling, also called uninterpreted modeling, may occur. Uninterpreted modeling is used whenever the designer is interested in analyzing the performance of the design. This typically occurs when the architecture or design is first created. Using representative loading, performance simulation and analysis are done to verify that the proposed architecture will meet performance requirements. For different parts of the design, this analysis will be done at different times. Within the overall system architecture, this analysis could

occur during the system definition phase. For smaller sections of the architecture, this analysis will occur during the architecture selection and verification phase.

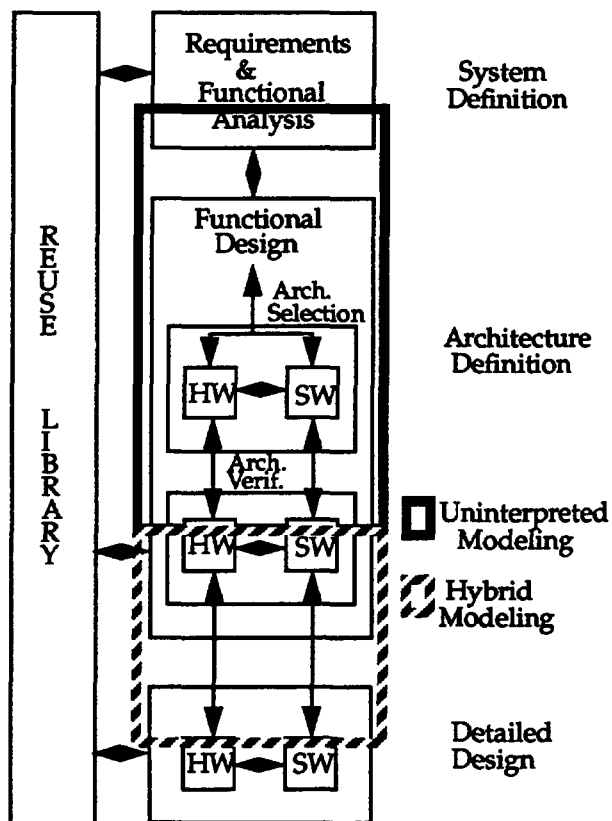


Figure 2. RASSP Design Process

As the design evolves, it may be desirable or perhaps even necessary to consider detailed design alternatives in order to minimize risk. Following the spiral model of development [16], this results in the spawning of mini-spirals to develop critical or high risk items. Based upon risk, pieces of the design may be at differing maturity levels. This is where the concept of hybrid modeling (bold dashed box) becomes important. As the overall system model is developed, certain high risk areas need further development. From a modeling perspective, this will be accomplished by including more detailed models. Specifically, the system level uninterpreted model will have certain components replaced with more detailed, interpreted models. To continue the RASSP philosophy of virtual prototypes, methods must support models at different levels of detail.

#### 4. HML Hybrid Interface

To demonstrate the use of hybrid interfaces, one possible application of these interfaces is illustrated in Figure 3. In this example, two performance models (white boxes) are interacting with two behavioral models (black boxes) using hybrid interfaces (striped boxes). The hybrid interface to the bus is a token-based interface, and the hybrid interface to the behavioral model is a valued interface, consisting of, for example, bits. Although not explicit, in this example, the bus is also described as a performance model.

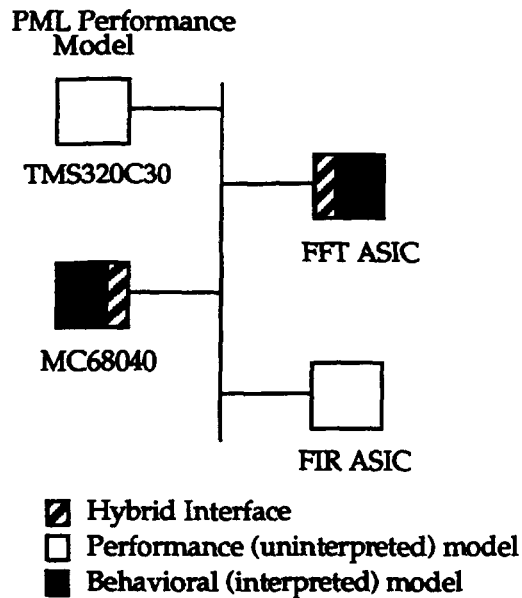


Figure 3. Hybrid Interface Example

Before discussing the actual implementation of such interfaces, the logical components of the interface are described. In the HML, a hybrid interface consists of the three components described below (see Figure 4).

- **Uninterpreted Hybrid Interface (UHI).** This component interfaces to the uninterpreted domain. This part of the hybrid interface interacts with tokens and returns data back to the performance model.
- **Hybrid Interface Core (HIC).** The HIC handles data translation (for example, from "tokens" to bit vectors), abstraction, and generation, as well as timing and synchronization between the UHI and IHI. It accepts data from the IHI and gives it to the UHI.

- **Interpreted Hybrid Interface (IHI).** This component interfaces to the interpreted domain. The IHI is the most complicated component since the interpreted domain must potentially represent many different abstractions, forms, and types. The IHI also handles timing and synchronization specific to the behavioral component. It is responsible for applying an appropriate stimulus to the behavioral component, determining when the behavioral component has completed its operation, and returning any data from the behavioral component to the HIC. In some cases, it performs clock generation if the behavioral component requires a clock.

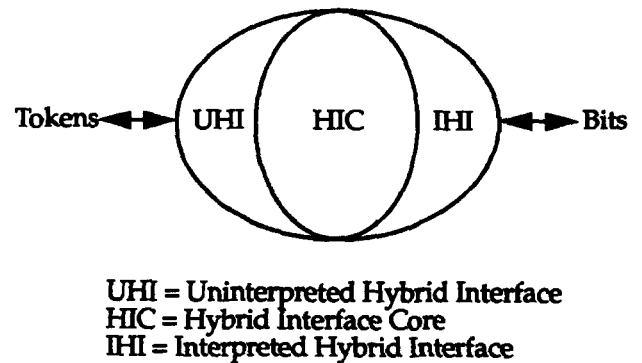


Figure 4. Components of a Hybrid Interface

Hybrid interfaces have standard port and generic interfaces. This port interface is the existing PML entity so that a PML element will have one entity, and at least two architectures: performance and hybrid. This allows easy insertion of hybrid elements into a PML-based design.

The hybrid interface architectures within the HML typically only contain the UHI and HIC. The design of the IHI is specific to the behavioral component. While the interface to the uninterpreted domain is relatively standard and straightforward, the interpreted domain is far broader.

The implementation of the hybrid interface is shown in Figure 5. It contains a process and three components: a queue, the IHI, and a global data store. The process contains the UHI code and the code comprising the HIC. The queue is used to store output tokens. The IHI is contained within a separate component. This allows the reasonably

generic UHI and HIC code to reside within the main process, while the more design dependent IHI is controlled via configurations. The IHI contains an Interpreted Component Interface (ICI) which interacts with the interpreted model. A global data store, a PML component used to store model-specific data, provides a consistent communication mechanism between the HIC and the IHI.

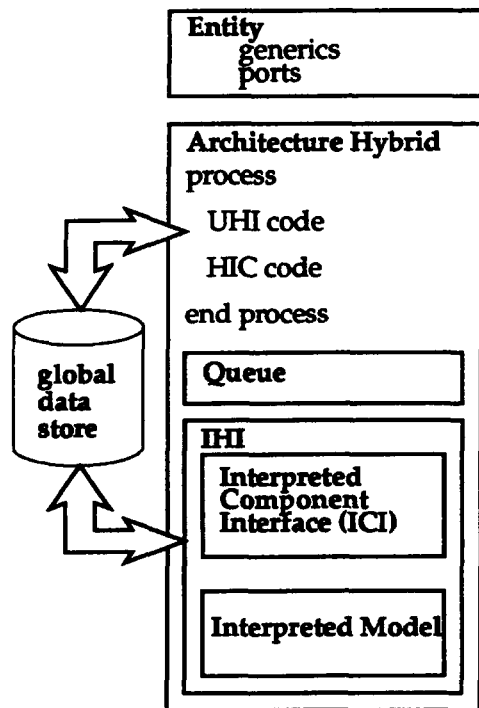


Figure 5. Hybrid Interface Implementation

### 5. The Hybrid Model Library

The Honeywell HML contains several hybrid interfaces. This library is based on the PML and provides hybrid architectures for the PML component entities. Like the PML, the HML is constructed in a generic fashion to support a wide range of applications. A variety of behavioral components have been used with these interfaces. This section discusses the hybrid interfaces that have been constructed for the leaf cells, the communication cells, and the processor model.

#### 5.1 Leaf Cells

Hybrid architectures have been constructed for the following basic leaf cells: indevice, outdevice, iodevice, and pipeline. The discussion in this section focuses on the iodevice.

The iodevice module can be used to model the performance aspects of a memory. The basic structure of the hybrid iodevice architecture follows Figure 5. The UHI and HIC are contained within the hybrid architecture of the iodevice. The interface between the HIC and the IHI is shown in more detail in Figure 6. The HIC, global data store, and queue components are taken from the PML and the HML. In this example, a RAM is used as the interpreted component.

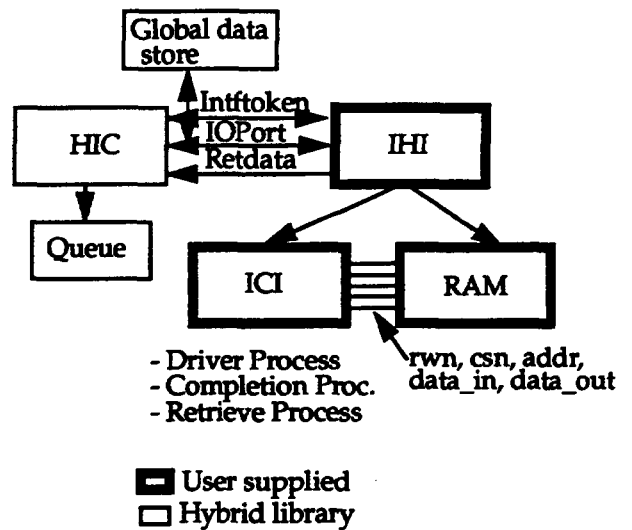


Figure 6. Interfaces within a Hybrid Architecture

Three "standard" signals are utilized to interface the HIC with the IHI. The *Intftoken* signal carries a request token that contains the operation to be performed by the interpreted component. In the case of the RAM, the operation would be either a read or a write. The *IOPort* signal connects to the global data store. The HIC writes translated (or generated) data values, such as address and data in the case of the RAM, into the global data store. These values are read by the ICI, converted into a format appropriate for the RAM, and applied to the RAM. The *Retdata* signal is used to return information back to the performance model, for example, during a read operation.

The ICI is an important element of this interface. The ICI consists of the driver process, the completion process, and the retrieve process, which apply the appropriate values to the interpreted component, check to see if the operation is complete, and return any data back to the HIC, respectively. The components shown in bold will differ between memories and are user-specified.

## 5.2 Myrinet Network Interface

As an example of a communication cell, a hybrid Myrinet Network Interface (NIF) has been developed. The hybrid Myrinet NIF is similar in structure to the hybrid models described earlier with one exception. Two sets of standard interface signals are employed. One set of interface signals (suffixed by "L" for Local) is used during the transfer of information from the host to the network, and the other set (suffixed by "G" for Global) is utilized for information flow from the network to the host.

As shown in Figure 7, the hybrid Myrinet NIF architecture, consisting of the UHI and the HIC, "encapsulates" the Myrinet NIF IHI description. The interface supports the sending and receiving of messages. Because two sets of standard interfaces are necessary, two global data stores are instantiated to store translated data going in each direction. Also, returned data can be sent to either the host or the network. In the figure, the interaction between the ICI and the interpreted component has been simplified.

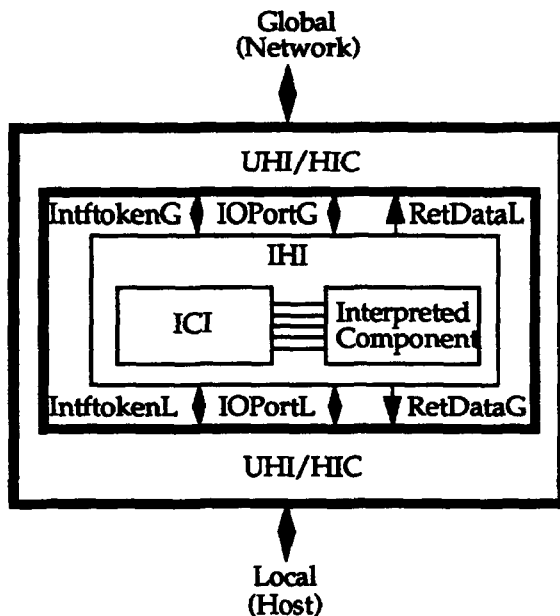


Figure 7. Hybrid Myrinet NIF

This hybrid architecture utilizes an interpreted component which is an abstraction of the actual Myrinet network interface. A more detailed discussion of this component is provided as part of an example in Section 7.

## 5.3 Processor

The hybrid processor interface may be used with either a processor element containing memory and even possibly a switch device, or a custom computing device. The current hybrid architecture was developed with a simple interpreted component consisting of an instruction set level cpu, a memory, and a bus communication element.

The hybrid processor architecture is illustrated in Figure 8. A hybrid token watcher, not part of the hybrid interface, accepts tokens from *HWport* (an interface to a bus). The UHI and HIC sections are similar to the corresponding sections in other hybrid models. The hybrid processor interface accepts read requests, write requests, and requests for executing programs in memory. The primary difference between this model compared to others is that the interface has requests coming from both the uninterpreted and interpreted domains. For example, the interpreted component may want to send a message to another processor. As a result, the additional signal *Extrequest* is required to support a request for generating a token to the uninterpreted domain.

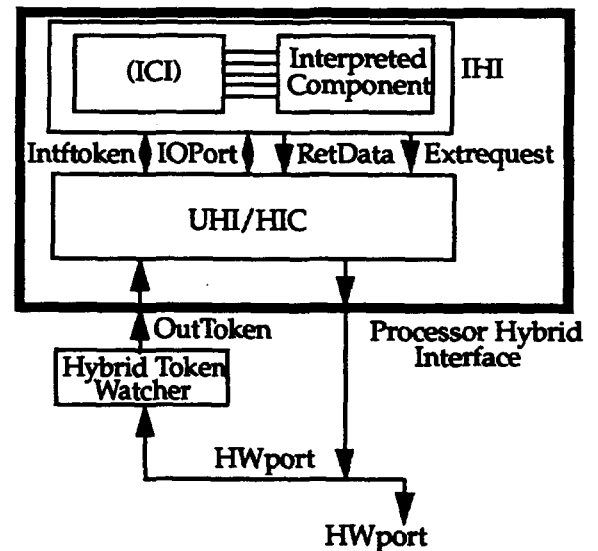


Figure 8. Hybrid Processor Architecture

The "watch-and-react" interface [17] is an alternative hybrid processor interface consisting of triggers and drivers. The trigger is used to detect events or changes in primary variables, and the driver is used to apply values to its probes, which represent signals of interest within the interpreted model.

## 6. Hybrid Interface Generation Toolkit

This section describes the Hybrid Interface Generation (HIG) toolkit which aids in the development of hybrid models. This toolkit consists of PERL scripts used to generate various portions of the hybrid interface. The first section provides the motivation for developing the toolkit. The next section provides an overview of how the toolkit can be used.

### 6.1 Motivation

The intent of the HIG toolkit is to provide an easy mechanism for the user to construct VHDL structures which will implement the hybrid interface. This means using the appropriate UHI and HIC process code, providing the proper interface signals, and developing the ICI. The original intent of the HML was to provide templates for these user-defined parts of the hybrid architecture. However, to ease the development of hybrid interfaces and to facilitate interoperability, it is believed that a toolkit approach is better. This will also ease integration with the Performance Modeling Workbench (PMW).

### 6.2 Process Flow

As shown in Figure 9, a bottom-up approach is best suited for creating a hybrid interface. Given an entity for an interpreted component, this approach consists of first creating an ICI. Once the ICI has been successfully created, the entities for the ICI and the interpreted component can be used to create an IHI. Upon generating an IHI, the IHI component can be instantiated within the hybrid architecture. This approach utilizes three primary modules, Genhybrid\_ici, Genhybrid\_ihi, and Genhybrid\_intf, which generate the ICI, the IHI, and the hybrid interface, respectively. The Genhybrid\_intf module utilizes a library of hybrid templates supplied with the HML. This library consists of "generic" hybrid architectures for the indevice, outdevice, iodevice, pipeline, processor, and the Myrinet network interface.

One of two paths can be employed to supply the interpreted component entity. The first path utilizes an interpreted component entity created by the user. This entity could have been constructed with the aid of an editor or obtained from an existing library. The second path utilizes Omniview's ALCHEMIST tool. Using this tool, a VHDL testbench file can be generated which contains the interpreted component entity and VHDL code that provides stimulus for the

interpreted component. Note that this second path simplifies the creation of the ICI since the stimulus can be used directly within the driver process of the ICI. Comparing Figure 9 with Figure 5, it can be seen that a layered approach is adopted by the toolkit, starting with an interpreted component and gradually adding interfacing layers on top of it.

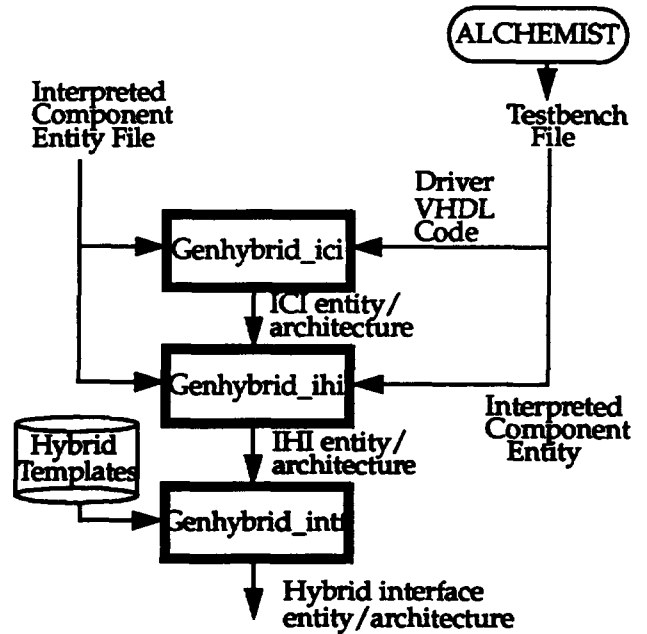


Figure 9. Process Flow for HIG Toolkit

## 7. Examples

Some examples are presented in this section to demonstrate the concepts and ideas developed earlier.

### 7.1 Hybrid Myrinet NIF

To illustrate the hybrid modeling capability, a simple Myrinet example is presented. As shown in Figure 10, the model for this example consists of an indevice, two Myrinet network interfaces (NIFs), a four port Myrinet switch (only two are used in this example), and an outdevice. The Myrinet NIF indicated in bold is a hybrid element. Messages are sent by the indevice and received by the outdevice. The indevice and outdevice models have been used for simplicity, although these would most likely correspond to processing elements.

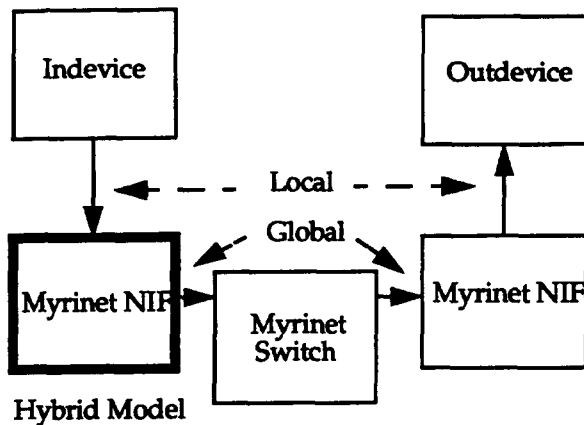


Figure 10. A Simple Myrinet Example

The PML versions of the Myrinet NIF and switch support the capability of either routing messages or splitting a message into packets and then routing the packets. If a message is split into packets, the packets can be combined into a single message at a destination NIF. This approach allows the user to control the level of message granularity, and thus, trade-off amount of detail versus simulation time.

The arrival of a token from the indevice on the local port triggers the execution of the hybrid Myrinet NIF architecture (see Figure 7). In this example, the HIC in the hybrid model uses a file-based data transformation mode for performing data generation. The generated address and data are written into a global data store by the HIC. The actual sending and receiving of messages is performed by the interpreted component upon the arrival of a request token (containing the operation to be performed) on the *Intftoken* signal within the IHI (see Figure 6).

As illustrated in Figure 11, the interpreted component in this example uses an abstraction of the actual Myrinet NIF device. It consists of three primary elements: an abstract Myrinet NIF device, an SRAM, and a clock generator. The NIF device reads and writes packets into the SRAM, and sends packets to other NIFs. The SRAM is used to store packets and can be accessed by the host or the NIF. The clock generator is used for sending the data bytes within a packet to other network elements.

The arrival of a request token within the IHI initiates the execution of a driver process within the ICI. The driver determines which operation is to be performed, reads data from the global data store, and applies the correct values to the interpreted

component through the interfacing signals (address, data lines). For example, to perform a message send, the host may provide a start address and write the data to be sent into the SRAM.

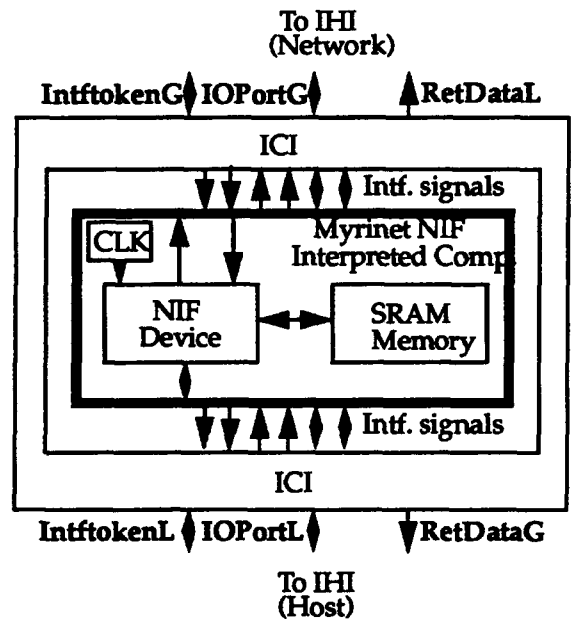


Figure 11. Interpreted Component

The ICI interfaces directly to the interpreted component and responds to two types of requests from the performance model. A *WRITETOKEN* request indicates that information is to be sent to another NIF. A *READTOKEN* request is used to indicate that a packet of information has arrived from the network. Once such a request is received, the appropriate signal values are applied to the interpreted component. Different completion criteria are used for the two requests. For example, the end of a packet send is assumed when a "GAP" symbol in the byte stream is encountered. It is also assumed that the ICI populates the size field of the token to be returned to the performance model. This information can be used to influence the delay of "downstream" modules and thus, the overall performance of the system model.

This hybrid Myrinet NIF is an example of a U/I scenario. To utilize this architecture for a U/I scenario would involve simply stripping out the appropriate uninterpreted half (in effect cutting Figure 7 in half horizontally). A likely use of the Myrinet NIF, or other communication element, would be to integrate an interpreted module with an uninterpreted communications bus. This type of scenario would require a U/I hybrid architecture.

## 7.2 Methodology Verification

We have developed a relatively generic structure to support hybrid modeling, which was one of the primary objectives of this project. The second objective is to verify the hybrid model concept in a design project. That phase of the project is still ongoing. We are participating in the design of a large system in a Honeywell product division, and have created a performance level model of the architecture. A high risk area in the architecture has been identified. This potential bottleneck needs more analysis than a performance level model can provide. We are in the process of developing a finer grained performance model of that bottleneck. That performance architecture will then be replaced with a hybrid architecture containing a behavioral model. The intent of this process is to evaluate the use of a hybrid architecture within the performance model versus off-line simulation of the behavioral model and back annotation of the performance results. Results will be discussed in future publications.

## 8. Summary

The contributions of this work focus on providing a capability that supports the hybrid modeling of complex systems. Specifically, we have developed a modeling structure to serve as a basis for constructing hybrid interfaces. Also, we have developed a VHDL-based library of hybrid interfaces to support the integrated simulation of performance models with behavioral models. Finally, we have created a toolkit to aid in the generation of hybrid interfaces.

Although qualitative arguments can be made regarding the use of hybrid modeling, more quantitative metrics are required to evaluate the benefit of hybrid modeling. We are in the process of further evaluating our hybrid modeling methodology.

## Acknowledgments

The HML was developed under contract # F33615-94-C-1495 from Wright Labs, Dayton OH.

The PML was originally developed under contract # F33615-92-C-3802 from Wright Labs and has been significantly enhanced under the RASSP program. The RASSP program is a four and one-half year, \$150 million Defense Advanced Research

Projects Agency (DARPA)/Tri-Service initiative intended to dramatically improve the process by which complex digital systems, particularly embedded digital signal processors, are designed, manufactured, upgraded, and supported. Under RASSP, PML work has been done under the following contracts: DAAL01-93-C-3380 and F33615-94-C-1495. HML development was also sponsored by the RASSP program.

Todd Carpenter, John Shackleton, and Todd Steeves, in addition to playing key roles on the PML development, have contributed ideas to the hybrid model library. Sabera Kazi has contributed to the hybrid model library development and verification. Minesh Amin was the principal author of the Hybrid Interface Generation toolkit.

## References

- [1] Franke, D. W., M. K. Purvis, "Hardware/Software Codesign: A Perspective," *Proceedings of the 13th International Conference on Software Engineering*, May 13-16, 1991, pp. 344-352.
- [2] Bourbon, B., "On System Level Design," *Computer Design*, December 1990.
- [3] Schultz, S. E., "An Overview of System Design," *ASIC & EDA*, January 1993, pp. 12-21.
- [4] Aylor, J. H., R. Waxman, B. W. Johnson, R. D. Williams, "The Integration of Performance and Functional Modeling in VHDL" in *Performance and Fault Modeling with VHDL*, J. Schoen, ed., Prentice-Hall, Englewood Cliffs, NJ, 1992.
- [5] Meyassed, M., R. McGraw, J. Aylor, R. Klenke, R. Williams, F. Rose, and J. Shackleton, "A Framework for the Development of Hybrid Models," *Proceedings 2nd Annual RASSP Conference*, Arlington, VA, July, 1995, pp. 147-154.
- [6] Rose, F., T. Steeves, and T. Carpenter, "VHDL Performance Models," *Proceedings 1st Annual RASSP Conference*, Arlington, VA, August, 1994, pp. 60-70.
- [7] Steeves, T., F. Rose, T. Carpenter, J. Shackleton, O. von der Hoff, "Evaluating Distributed Multiprocessor Designs," *Proceedings 2nd Annual RASSP Conference*, Arlington, VA, July, 1995, pp. 95-102.
- [8] Performance Modeling Workbench User's Manual, Version 0.9, Omniview, Inc., Pittsburgh, PA, 1996.
- [9] Honeywell Technology Center, VHDL Performance Modeling Interoperability Guideline, Version 1.6, November, 1995.
- [10] "The RACE Architecture for Real-Time Multicomputers," Mercury Computer Systems, Inc., 1993.
- [11] Boden, N. J., et al., "Myrinet: A Gigabit-per-Second Local Area Network," *IEEE Micro*, February 1995, pp. 29-36.
- [12] Rose, F., T. Carpenter, S. Kumar, J. Shackleton, and T. Steeves, "A Model for the Coanalysis of Hardware and Software Architectures," *Proceedings of the 4th International Workshop on Hardware/Software Codesign*, March 18-20, 1996, Pittsburgh, Pennsylvania, pp. 94-103.
- [13] Kumar, S., J. H. Aylor, B. W. Johnson, W. A. Wulf, *The Codesign of Embedded Systems - A Unified Hardware/Software Representation*, Kluwer Academic Publishers, Boston, Massachusetts, 1996.
- [14] Richards, M., "The RASSP Program Overview and Accomplishments," *Proceedings 1st Annual RASSP Conference*, Arlington, VA., August 1994, pp. 1-8.
- [15] Saultz, J., "Lockheed Martin Advanced Technology Laboratories RASSP Second Year Overview," *Proceedings 2nd Annual RASSP Conference*, Arlington, VA, July, 1995, pp 19-31.
- [16] Boehm, B. W., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, May 1988, pp. 61-72.
- [17] Dungan, W. W., R. H. Klenke, J. H. Aylor, "A Watch-and-React Interface for Hybrid Modeling," Department of Electrical Engineering, University of Virginia, Technical Report No. 960531.0.
- [18] IEEE, "IEEE Standard VHDL Language Reference Manual," New York, NY, IEEE Standard 1076-1987, March 31, 1988.