Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

ARENA^{TM}: A SIMAN/CINEMA-BASED HIERARCHICAL MODELING SYSTEM

C. Dennis Pegden
Deborah A. Davis

Systems Modeling Corporation
504 Beaver Street
Sewickley, PA 15143, U.S.A.

ABSTRACT

This paper presents an overview of Arena--an extendible modeling system that is built on SIMAN/Cinema. A key idea behind Arena is the concept of tailorability to a specific application domain through the use of a template.

1 INTRODUCTION

During the past decade, there has been tremendous progress in the development of new simulation technology and related software. Most modern simulation tools now provide an interactive graphical interface for model definition as well as real-time graphical animation. These new capabilities represent a significant improvement over earlier nongraphical, batch-oriented modeling tools.

The many recent advances in simulation technology have created a greater awareness and use of simulation by industry. Managers are now more aware of the potential benefits of simulation. However, even with the many important advances that have been made over the past several years, there are still many cases where complex systems are being designed and implemented without the benefit of simulation. In our view, a very small percentage of systems that could benefit from simulation are actually simulated, and the primary reason for this is the high level of effort required to employ simulation technology successfully. We believe that the key to making simulation technology more widely used is to make the tools significantly easier to learn and use.

A number of authors have discussed the importance of the complexity of simulation as an impediment to its widespread acceptance. There are many potential users who believe that they cannot successfully apply simulation without employing an outside consultant [O'Loughlin, et. al, 1988]. Even managers who realize the benefits of simulation often abandon simulation attempts because models are too time-consuming and labor intensive to write, test, debug and analyze [Thomasma and Ulgen, 1988]. Resource requirements and scheduling constraints often leave little or no time to put together a detailed simulation analysis [Suri, 1988].

The desire for ease-of-use by practitioners is highlighted by the growth in popularity of domain-restricted simulation packages such as Witness and ProModel for modeling manufacturing systems. Although these packages lack the flexibility of a full-capability simulation modeling language such as SIMAN, they offer a more focused modeling framework within a limited application domain. These tools are often cited as being effective for use in rough-cut modeling of simple manufacturing systems, but limited in terms of general simulation modeling features [Fegan, et. al, 1991]. For this reason, the usefulness of these packages for modeling complex systems is somewhat limited [Ulgen and Thomasma, 1990].

With the present state of simulation technology, users are forced to make a choice between the ease-of-use of the domain-restricted packages, and the flexibility of languages [Law and McComas, 1990]. Although over time the packages are becoming more flexible and the languages are becoming easier to use, there remains a significant void between these product categories. If the user elects to employ one of the packages (if one exists for his/her domain of interest), the limited flexibility provided by these packages may result in an inaccurate representation of the system--resulting in false conclusions from the model. On the other hand, by electing to employ one of the languages, the user is required to make a greater commitment in time and effort to master the capabilities of the language fully.

This paper presents a new SIMAN/Cinema-based modeling/animation system, named Arena, developed by Systems Modeling Corporation. In our view, this new system represents a major advance in simulation technology by combining the modeling power and flexibility of the SIMAN/Cinema system with the ease-of-use of application-focused packages. Arena offers a high level of modeling flexibility across a wide range of problem domains, yet is very simple to learn and use.

Arena also provides a complete simulation environment that supports all basic steps in a simulation study. The Arena system includes integrated support for input data analysis, model building, interactive execution, animation, execution tracing and verification, and output analysis. In this paper, we will restrict our focus to the model building functionality of Arena.

The key idea behind Arena is the concept of tailorability to a specific application area. The Arena system is not restricted to a specific set of predefined modeling primitives, but can be easily tailored to a domain-specific application area by means of an application template. Hence a user of Arena who is modeling health care systems could employ a health care template that would contain modeling primitives focused on health care systems (doctors, nurses, beds, X-ray, etc.). Likewise, a person modeling traffic flow in a city could employ Arena with a traffic flow template that would contain modeling primitives focused on traffic flow (roads, exit ramps, cloverleafs, stoplights, etc.). Unlike conventional simulation systems that have their modeling primitives "hard-coded" into the software by the vendor, Arena allows the modeling primitives to be "soft-coded" by the end user by means of the template.

The application template concept is fundamental to the flexibility and ease-of-use provided by Arena. This mechanism makes it possible to provide the end user of the product with a tool that closely matches the real system being modeled--hence the user is presented with concepts and terminology that are focused on his or her problem. This dramatically reduces the level of modeling abstraction required by the user. However, the user is not limited to the primitives provided by a single domain-restricted primitive set. The user can combine domain-restricted primitives from one or more application-focused templates with the full modeling power of the SIMAN simulation language and thereby avoid the modeling "brick wall" encountered with traditional hard-coded, domain-restricted packages. The modeling power of SIMAN is made available to the user as simply one additional template.

2 BACKGROUND

The Arena system is an extendible modeling environment that is built upon the SIMAN/Cinema simulation/animation software. Arena is a graphical modeling/animation system that is based on concepts from object-oriented programming and hierarchical modeling. Since these two concepts represent the foundation for the Arena product, we will briefly review these concepts and discuss their significance to Arena. As we will discuss, there is substantial overlap between the concepts of object-oriented programming and hierarchical modeling.

2.1 Object-Oriented Simulation Languages

Object-oriented programming is a powerful paradigm gaining widespread use in software development. Major software developers including IBM, Borland, Microsoft, Next, and so on, have made strategic commitments to the object-oriented approach to software development. The Arena software was also developed using an object-oriented approach--and this proved to be a highly effective paradigm for this development project. Although this paradigm is now being widely embraced throughout the software development community, it is interesting to note that the first general language to promote an object orientation was the simulation language Simula (Birtwistle, et al., 1979). The concepts in modern object-oriented languages such as C++ and Smalltalk are directly based on Simula.

Simulation languages tend to have an object focus since they are used to represent physical objects in the system being modeled--and it is therefore natural that the object-oriented paradigm was first developed within the simulation community. The entities, queues, resources, transport devices, conveyors, etc., in languages like SIMAN are used to represent physical objects in the systems that are modeled. However, languages such as SIMAN are not true object-oriented languages. An object-oriented simulation language can be distinguished from other simulation languages by its support for the creation and description of new objects [Bishack and Roberts, 1991]. Although traditional languages such as SIMAN and GPSS incorporate objects such as entities, queues, resources, etc., they do not allow the user to create their own objects easily. In an object-oriented approach, the user can create new objects by building on the base objects included in the language.

There have been a number of object-oriented simulation languages developed over the years. A list of 48 simulation models and languages written in object-oriented programming languages is presented in a recent survey [Thomasma, et al., 1990]. In addition to Simula, these include MODSIM [Belanger, 1990], Sim++ (Lomow and Baezner, 1990], and Smalltalk-80 [Goldberg and Robson, 1989]. These languages allow the user to define and create new objects based upon the predefined objects in the language.

Although these languages are relatively new, they have not shared the same widespread use in practice as their more traditional counterparts such as SIMAN, SLAM, and GPSS. One of the reasons for this is the level of expertise required to develop models with these languages. The user is required to program in a C++, Smalltalk, or similar language and design and develop new objects for a particular application using the object-oriented paradigm. Hence model development requires a fairly high level of programming skills as well as experience with the object-oriented paradigm.

2.2 Hierarchical, Modular Modeling

A related development to object-oriented simulation languages has been a focus on hierarchical modular modeling systems. A number of researchers have developed hierarchical modeling systems including the SIMAN Module Processor [Tempelmeier and Endesfelder, 1987], SAM [Concepcion and Schon, 1986], DEVS-Scheme [Zeigler et. al, 1989], SmartSim [Thomasma and Ulgen, 1988], RESQME [Gordon, et al., 1986], and ISI [Lane, et al., 1989]. These systems employ a hierarchical framework to provide an extendible modeling system.

Many of these systems employ object-oriented concepts and are based on hierarchical modeling concepts proposed in the DEVS formalism [Zeigler, 1984]. The basic concept embedded in the DEVS formalism is that a simulation model can be built in a hierarchical fashion. The base primitives of the system can be combined to form new primitives (subsystems), which can themselves be combined in a hierarchical fashion with other primitives. By providing a mechanism for managing the saving/retrieving of subsystems, the concept promotes the idea of application-specific libraries. Once a specific component has been modeled as a subsystem, it can be re-used in other similar applications.

The hierarchical modular modeling systems are very similar in concept to the object-oriented simulation languages. These systems allow the users to create new primitives from the base primitives provided by the system. These systems also include a facility for easily storing and retrieving user-derived subsystems and incorporating these subsystems across multiple models. In some cases, these systems provide a graphical method for defining new primitives from the existing primitives in the language. This approach is simpler and more natural for a modeler than the programming approach adopted by the statement-based, object-oriented programming languages. The graphics-based hierarchical modeling systems encourage a high degree of reusability of subsystem models and promote the concept of application-specific libraries.

The SIMAN Module Processor (SMP) developed by Tempelmeier and Endesfelder provides a single level of hierarchy that allows users to extend the modeling system provided by SIMAN. Although this system does not allow for multiple levels of hierarchy as specified in the DEVS formalism, it does support the concept of extendibility for modeling primitives. The SMP reads modules from a user-written module library, interprets them in relation to interactively specified data (e.g., problem-specific parameters) and produces a syntactically correct SIMAN simulation model. The user can develop new modules and add these to a library. Tempelmeier and Endesfelder have developed module libraries for modeling flexible manufacturing systems, warehouses, material handling systems, etc., and these have been used by SIMAN users in Europe to simplify the modeling process.

SAM was one of the first implementations of a hierarchical modular modeling system based on the DEVS formalism. This system provides an environment for specifying, designing, and analyzing discrete event systems for distributed simulation. DEVS-Scheme is an implementation of the DEVS formalism in PC-Scheme, a dialect of LISP that contains an object-oriented subsystem.

The SmartSim system developed by Thomasma and Ulgen is also based on the DEVS formalism proposed by Zeigler. This package is written in Smalltalk and allows the user to build hierarchical submodels using both primitive and user-created objects. New objects are added to the system by writing the necessary object-oriented code in Smalltalk. Once a new object is created, the user draws an icon to represent the object. The newly created object is then graphically manipulated using this icon. The graphical interface of SmartSim automatically generates a Smalltalk model from a graphical representation of the model for simulating the system. Because of the slow execution speed of Smalltalk, a version of SmartSim was also developed to generate SIMAN models automatically.

The RESQME system provides a graphical method for implementing hierarchical models. The user can create a submodel and draw a user-created icon to represent that submodel. It is important to note that in this system the user does not have to program in an object-oriented language to add a new object to the system--object creation is done graphically by placing and interconnecting icons representing existing objects. One of the interesting features of this system is that the user can explode an icon during model execution to see the operation of the submodel.

The ISI product is a hierarchical, modular simulation system written in a dialect of Lisp. Like RESQME, this product allows the user to define graphically new objects from the base primitives provided and to associate a user-defined icon with the new object. These objects can be stored in libraries and re-used. Models (and new primitives) are developed by selecting icons from the libraries and placing and interconnecting these icons on the screen. One important feature of ISI is that it was designed to make it very easy to modify the primitive objects in the system as well as the model generation function. Separate versions of ISI have been developed for generating SIMAN, SLAM, and GPSS models.

Although hierarchical modeling systems have many important advantages over traditional languages, they have had limited use in industry. There are several factors that have limited the acceptance of these systems. In some cases, the built-in modeling primitives are very limited, and therefore the user must create many new modeling primitives to represent the components in the system. In some cases, object-oriented programming skills are required to add new objects to the system. In addition, in several cases the necessary related functions such as animation, output analysis, interactive model verification tools, etc., are not well supported. The most significant factor, however, is the limitation imposed on developing application-specific libraries for use with these systems. Although the general idea of application-specific modeling libraries is embedded in these hierarchical modeling systems, in our view, the current systems do not provide an adequate framework for building truly flexible and practical application-specific libraries.

3 THE ARENA MODELING FRAMEWORK

The Arena system is SIMAN/Cinema based and builds upon concepts from both the object-oriented languages and the hierarchical modular simulation systems. Arena allows the user to create new modeling constructs from the basic modeling primitives consisting of SIMAN blocks/elements. The user defines a graphic icon to represent the new modeling construct. This icon can be static or can incorporate Cinema animation components.

To illustrate the basic concept, consider the SIMAN block sequence QUEUE-SEIZE-DELAY-RELEASE shown in Figure 1. In Arena, we could build a new construct named SERVER to represent this sequence of four SIMAN blocks. To do this, we would i) define the operands of our new construct SERVER, ii) construct a submodel consisting of QUEUE-SEIZE-DELAY-RELEASE and specify how the operands of SERVER are passed down to these component blocks, and iii) draw an icon to represent the new SERVER primitive. This new SERVER primitive can be employed in model building in exactly the same way as the original SIMAN primitives in Arena. Hence the SERVER primitive becomes an extension to the SIMAN language.

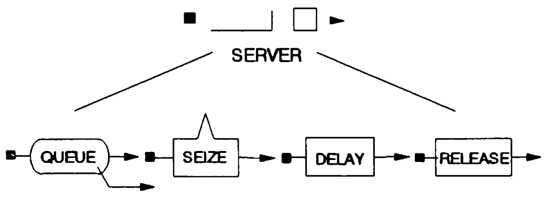

Figure 1: The SERVER Module

Existing SIMAN users who are accustomed to the familiar top-down graphical flowchart representation for SIMAN models will notice the left-right flow convention used for modules in Arena. Hence entry to a SIMAN module is on the left, and exit is on the right. We have found this left-right flow convention to be more natural for most applications than the more traditional top-down format (entry and the top, exit on the bottom) used by SIMAN. However, this convention is arbitrary and can be easily changed from application to application. Other than the location of entry and exit points to the module, the SIMAN module symbols are drawn the same as before.

From the beginning, the driving force behind the design of the Arena product was support for the development of truly flexible and practical application-specific libraries. The functionality needed for building flexible libraries of primitives dictated the hierarchical framework and related functionality incorporated into Arena. Although incorporating basic hierarchy into Arena to accommodate the simple SERVER example discussed above was relatively straightforward, the real challenges were in providing a framework for hierarchy that incorporated the necessary constructs to support flexible application-specific templates fully. One of the important ways that the Arena system differs from previous hierarchical modeling systems is in the flexibility provided by its hierarchical framework.

One of the fundamental requirements that was driven by the need to support flexible application templates was the need to support repeating subcomponents within a module. For example, a derived module representing a workcenter might have a variable number of servers, and each server might have its own set of associated data. To provide the necessary support for application template development, we felt that it was necessary to allow the user who places the module to vary the number of servers easily within the module and to enter unique data for each server.

A second major issue that evolved as a requirement for flexible and practical template support related to multiple variations in the definition of a module. The basic issue here is the need to provide a single module that represents many variations of a basic module definition, based on the input data supplied by the user. For example, a single workcenter module might represent various types of workcenters whose underlying structure (in terms of component modules and associated operands) changes based on the user inputs to the workcenter module. Without this capability, application templates would be required to either i) have an excessive number of modules, with many of these being slight variations of other modules in the template or ii) incorporate routine logic within modules to branch based on cases (thereby slowing execution).

4 BASIC TERMINOLOGY AND CONCEPTS

Since the Arena system blends together concepts from SIMAN/Cinema, object-oriented programming, hierarchical modular modeling, as well as a number of new concepts, it is helpful to define some basic terminology used to discuss the hierarchical modeling concepts in Arena.

4.1 Objects and Properties

The term SIMAN object is used in Arena to denote the constructs used by SIMAN to represent physical objects in the real system. Examples of SIMAN objects include Resources, Conveyors, Transporters, Queues, and Variables. There is a one-to-one correspondence between objects in Arena, the elements in the conventional SIMAN experiment, and the objects being modeled in the real system. Everything that is defined in the SIMAN experiment is defined to be an object in Arena.

The characteristics of an object are referred to as properties. For example, the speed, acceleration/deceleration, and turning velocity factor are all properties of a transporter object. The properties of objects correspond to the fields defining each object in the SIMAN experiment.

4.2 Base and Derived Modules

The term module is used in Arena to denote the modeling components used to construct a model of a system. There are two basic types of modules; these are called base modules and derived modules. Base modules are the lowest level modules in Arena and correspond directly to the SIMAN blocks. Hence a QUEUE module in Arena corresponds directly to the QUEUE block in SIMAN. Derived modules are built up from one or more base and/or derived modules. For example, the SERVER construct discussed earlier is an example of a derived module built from the QUEUE, SEIZE, DELAY, and RELEASE base modules. Note that the SERVER module can be used in building additional higher-level modules.

The user of Arena builds a model by placing and interconnecting modules in a layout. For example, a model of a simple service system might be built by interconnecting modules from a general-purpose modeling template representing the arrival process, service process, and departure process. A model of an emergency room might be built using modules from a health care template representing the reception/admission process, X-ray, and so forth. The modeler may freely mix modules from multiple application templates--including the base SIMAN template.

4.3 Operands

A module may have operands that define values associated with the module. For example, the SERVER module might have operands defining the queuing discipline, server name, and processing time. The module creator defines the characteristics of each operand including the positioning in the dialog box, user prompt, default value, permissible values, and user interface method (buttons, list box, toggles, etc.). The user may view/edit the operands of a module by double-clicking the mouse on the module in the layout--this causes the dialog box for the module to appear.

The operands of a derived module may supply values for operands of its component modules from which it is constructed. For example, the string supplied for the server name could be used to supply part or all of the queue identifier in the QUEUE module and/or the resource identifier in the SEIZE and RELEASE modules. Note that operand values are not inherited up from below--but are instead defined at higher-level modules and then passed down to lower-level modules. This is analogous to the passing of arguments from a main program down to a function in normal programming.

Module operands may either create or reference SIMAN objects and their properties. Operands that define the name of a SIMAN object (e.g., a resource) are called namerands, and operands that reference properties of objects are called properands. For example, the operand defining the queue identifier for the QUEUE module is a namerand and may either create a new instance of the queue object with the specified name or reference an existing queue object. The operand defining the queue ranking rule is a properand since the ranking rule is a property of the queue object.

4.4 Entry/Exit Points and Connectors

The entities in the model move from module to module and activate functions that act upon the SIMAN objects and thereby change the state of the system. For example, an entity moving through the SERVER module may change the state of the queue and resource objects referenced by the module. Entities enter modules at locations called entry points and exit modules at locations called exit points. The exit point of one module may be connected to the entry point of a second module by means of a graphical connector. An entity passes through a connector in zero simulated time.

4.5 User View

As discussed earlier, a derived module is created by i) defining the operands for the new module, ii) building a submodel from existing base/derived modules to represent the logic of the module, and iii) drawing an icon to represent the new module graphically. This icon is referred to as the user view of the Module because it is the view of the Module seen by the end user who places the module in his/her model.

The user view of a module is drawn by the module creator to represent the new module graphically. The user view can be a static icon or it can contain one or more animation components. For example, the user view for the simple SERVER module could be either a static module symbol or an animated queue and resource. It is also possible to provide the end user with both alternate user views for this module--and thus allow for both an animated and nonanimated representation of the module.

The user view may also contain one or more entry/exit points. These entry/exit points are used by the end user of the module to interconnect this module with other modules in the layout. They may be placed in any position in the user view by the module creator.

4.6 Animation Objects

One of the important features of the Arena system is the full integration of animation as a fundamental element of the system. Arena contains Cinema-based animation objects including queues, levels, variables, resources, plots, histograms, transporters, conveyors, and paths. These objects dynamically change position, shape, or color during the execution of the model.

Animation can be incorporated into an Arena model in several different ways. If Cinema animation primitives are included in the user view of a derived module, then these animation objects come along with the module when it is placed in the layout. Animation objects can also be freely added anywhere in the model layout to embellish the animation that is included in the modules' user view.

Note that when animation is included in the user view of the modules, the building of the model and the animation occurs simultaneously--i.e., the process of building the model also constructs the animation. This integration of model building and animation drawing is a useful feature for many applications. However, in some cases it is convenient to be able to separate the development of the animation layout from the development of the model. This provides greater freedom for drawing the animation layout and also supports the concept of multiple animation layouts for the same model. Arena is flexible in this regard and supports independent animation layouts as well as the integrated approach of simultaneously building the model and the animation.

Animation objects can also be temporarily added to the layout during execution of the model as an aid to model verification/validation. For example, the simulation can be interrupted and an animated queue can be added to view the current entities in the queue. This queue could then be deleted and the run continued.

The Arena system also supports both concurrent and play back animation of the model. In concurrent animation, the animation objects are updated on a real-time basis as the simulation is executing. This is an essential feature for effective model verification and validation activities. In playback mode, a trace file is generated during the simulation run that contains a time-ordered history of the significant events that occurred during the simulation. This trace file can then be use to playback an animation of the simulation. Although the playback mode is restrictive and not nearly as effective as concurrent animation for model verification and validation activities, it is sometimes convenient for repeated replays of an animation.

The Arena system contains many new and advanced Cinema-based animation features. These new features include a virtual drawing area with pan and zoom, rotation of symbols along paths, direct editing features such as 'drag-and-drop,' simplified symbol management, and much more [Conrad, 1992].

5 OBJECT-ORIENTED CONCEPTS IN ARENA

Although Arena is motivated by object-oriented concepts, its approach to hierarchy is different from most other object-oriented simulation languages. In Arena, modules are defined hierarchically, but SIMAN objects (i.e., resources, transporters, etc.) are not. New SIMAN object types cannot be created as is done in object-oriented languages. The available SIMAN object types are built into Arena and cannot be changed by the user. Hence the object-oriented concept of extendibility applies to the modules and not the SIMAN objects. In this sense, Arena might best be described as module oriented as opposed to object oriented.

At first glance, the inability of the user to create new object types within Arena may seem like a limitation, but this is actually not the case. Since an operand of a module may define the instance of an object or its properties, objects can directly share in the benefits of module hierarchy--even though they are not themselves hierarchically defined. For example, we could define a derived module named WORKCENTER, that is built on top of the derived module named SERVER, and supplies properties for one or more standard SIMAN objects (queues, resources, conveyors, etc.) at multiple levels of the hierarchy. In effect, objects can piggyback onto the hierarchical definition of modules to gain the benefits of hierarchy for themselves.

A key advantage of the approach employed by Arena is that the derived modules are defined graphically, and therefore no programming is required to build application-specific libraries. Hence the development of application-specific libraries is a modeling task and not a programming task. This is not generally the case in object-oriented simulation languages--these systems typically provide extendibility for objects, but require the user to program in an object-oriented language (C++, Smalltalk, etc.) to add new modeling primitives.

6 TEMPLATES

The real power of the Arena product is realized through its support for application-focused templates. Numerous application areas have been discussed for possible templates including computer systems, communication systems, traffic flow, airport design, fast food restaurants, high-speed packaging, health care, process industry, package sorting, warehousing, and many more. Each new application template brings simulation technology much closer to a large class of potential users.

In addition to general application areas, the opportunity exists for developing company-specific templates that contain primitives focused at a specific company--thereby sharing the effort and expertise of a few people. For example, an automotive manufacturer could develop a template containing manufacturing equipment and/or workcenter designs specific to automotive assembly within the company. By tailoring the modeling primitives to a specific company, such templates can help make simulation technology a much simpler and more widely used methodology throughout the company.

In this section, we will briefly discuss four examples of templates for use with Arena. The first is the basic SIMAN template for building traditional SIMAN models. The second is the Arena template, which is a general modeling template that builds on SIMAN and provides useful higher-level primitives (e.g., SERVER) across a broad range of applications. The third is a template for modeling general manufacturing processes. The final example is a template for modeling semiconductor wafer fabrication.

6.1 SIMAN Template

The SIMAN template contains base modules that correspond directly to the SIMAN blocks/elements. By selecting and interconnecting modules from the SIMAN template, the user can build what is referred to as a flat SIMAN model.

In its most basic use, Arena can be employed with the SIMAN template to provide a graphical model builder for SIMAN. Hence Arena and the SIMAN template replace and expand on the functionality of the original Blocks and Elements editors.

6.2 Arena Modeling Template

The Arena template contains derived modules that provide general-purpose modeling primitives that are at a higher modeling level than the standard SIMAN blocks/elements. The objective of the Arena template is to provide a comprehensive set of high-level modeling primitives for modeling simple systems.

The basic idea behind this template is that SIMAN models frequently contain a series of standard block sequences such as QUEUE-SEIZE-DELAY-RELEASE or QUEUE-ACCESS-DELAY-CONVEY. The Arena template contains a collection of useful SIMAN block combinations for general-purpose modeling. The Arena template modules typically correspond to combinations of several base modules from the SIMAN template.

For the new user, the Arena template provides an easier entree into modeling using Arena. By providing the modeler with higher-level primitives such as

Arrival, Service, and Departure, the new user can build simple models with less modeling abstraction--and therefore, with greater ease and less learning. The new user can then transition into using the base SIMAN primitives as necessary to expand on the modeling flexibility of the Arena template.

For the experienced user, the Arena template allows the user to leverage the predefined modules in the Arena template to develop large models more rapidly. Since a single module in Arena typically corresponds to several base SIMAN modules, large models can be built more rapidly and with fewer errors. The modules in the Arena template can be mixed with standard SIMAN modules to provide a detailed modeling capability for complex systems.

6.3 Manufacturing Template

The Manufacturing Template is a collection of modules that may be combined to describe the process flow of a manufacturing system. The template was designed to support the majority of discrete manufacturing applications; however, it is not limited to that application domain. Each of the modules within the template is made up of one or more SIMAN blocks and/or elements. The modules have been designed to allow for flexibility in modeling, while ensuring a user-friendly modeling environment.

The Manufacturing Template supports various types of process flow including unconstrained push, constrained push and pull. Unconstrained push may be utilized if there are no work-in-process (WIP) constraints. Constrained push is used when blocking occurs due to either physical space or WIP constraints. The use of production and transfer authorizations enable the template to provide just-in-time (JIT) type pulling capability. Any of these process flow methods or a combination of them may be used in a single simulation model.

Various types of workcenter processing activities are supported within the template. The processes supported include single part, production, assembly and batch processing. Single-part processing implies that one part enters a workcenter and the same part exits the workcenter upon completion of processing. Production-type processing enables one part to enter a workcenter and multiple parts of the same or different part types to exit the workcenter. Assembly operations are accomplished by combining all specified component parts into an assembled part. Finally, both temporary and permanent batching is available at the workcenters. If parts require batch processing, a number of parts can be grouped for the processing activities and unbatched following completion.

In incorporating the features of the SIMAN language into the template, material handling capability is strongly supported. Both free path and guided transporters, as well as accumulating and nonaccumulating conveyors, can be used to move parts from one workcenter to the next. For less detailed control of material handling, operator resources or simple delays may be used for transfers.

The Manufacturing Template consists of three types of modules: workcenter modules, component modules, and data modules. Both the workcenter and component modules consist of the logic portion of a manufacturing system, including such functions as entering a workcenter, exiting material handling, processing and moving to the next workcenter. Workcenter and component modules support modeling at multiple levels of detail. Workcenter modules include sufficient functionality and flexibility to describe many manufacturing systems. Component modules are simply components of workcenter modules that allow a user to incorporate detailed logic to represent a system. Data modules allow the definition of information related to objects used in the workcenter and component modules, such as operators and parts.

Given that the Manufacturing Template consists of two levels of logic modules (workcenter and component modules), it supports both machine-based and jobstep-based modeling orientations. In a machine-based orientation, the logic defining what is to occur, though not necessarily all of the data, is specified in conjunction with the machine (or workcenter). Workcenter modules may be used to define processing logic and the default data associated with each workcenter. In a jobstep-based orientation, the logic defining what is to occur is specified with the jobstep itself. By utilizing component modules, the processing logic and data may be incorporated directly into the process plan for a part. Combinations of the two modeling orientations can be achieved by intermixing the levels of logic modules.

6.4 Semiconductor Wafer Fabrication Template

The wafer fabrication template is designed to support modeling of wafer fabrication operations in the semiconductor industry. (This template is based on prototype research performed by D. Phillips, G. Curry, and B. Deuermeyer at Texas A&M University, and it was developed with partial funding from SEMATECH. John Fowler managed the development of this template for SEMATECH and played a key role in the design of the template.) While not restricted to this application domain, the model structure and terminology used in this template are consistent with that found in wafer fabrication applications.

Models that are built using this template consists of two types of user input, model data and model rules. Model data definitions include products, technologies and jobsteps (process plans), recipes, workcenters, resources, operators, and production schedules. Model rules define the model logic to be used by a particular workcenter or recipe--i.e., model rules define the logic by which manufacturing lots seize and release sets of resources, are batched and split apart, and undergo processing delays. Given the large number of jobsteps in a typical wafer fabrication model, model rules are defined separately from the model data so that they may be re-used.

A unique aspect of the wafer fabrication template is that is it can generate either a standard simulation model or a special model that is used to perform analytical flow and queue analysis of the system. Either of these models can be generated from the same user description of the system--i.e., the user builds a single model of the system and then selects which form of the model (standard or flow and queue) they would like to execute. Flow and queue analysis, as the name implies, involves performing two types of analysis: a flow analysis and a queuing analysis. First, flow analysis is performed to determine the rate at which products (manufacturing lots of wafers) flow through individual components of the system. Second, based on the calculated flow rates, an interactive queuing analysis is used to calculate resource utilizations and queue times.

The flow and queue analysis provides the same basic performance measures for the system as does the standard simulation model, but uses queuing approximation formulas to obtain the results. The advantage of the flow and queue analysis is that it executes much faster than a simulation model of the same system. The advantage of the simulation model is that it provides more accurate results.

7 SUMMARY

Arena is a new hierarchical modeling system based on SIMAN and Cinema. The key feature of this system is that it allows users to define new modeling primitives (modules) that can be tailored to a specific problem domain. These domain-focused primitives can be placed into libraries called templates. The development of application-specific templates for use with Arena creates the exciting opportunity to bring simulation technology to a large cross section of people who currently do not benefit from this technology. Arena brings simulation technology much closer to the application specialists--and does so across an unlimited range of problem domains.

ACKNOWLEDGMENTS

The Arena product is the result of the efforts of a large group of talented and dedicated individuals. People have contributed to the development of Arena in conceptual design, software design and implementation, application template development, and testing and evaluation. Although a large number of people have contributed ideas to Arena over time, we would like to acknowledge several key contributors to this project specifically. The hierarchical framework of Arena was designed and implemented by Steve Frank and Dave Takus. SIMAN V and Cinema V development efforts were led by Dave Sturrock and Jack Poorte whose team consisted of Mark Glavach, Jim Higley, Robb Kipp, and Ray Wong. Gary Nordlund and Chris Watson designed the Arena, Manufacturing, and Wafer Fabrication templates and defined product requirements to support these efforts. The testing, evaluation, and documentation of the software was done by Lynn Barrett, Norene Collins, and Sherri Conrad. Cheryl Vicinanza designed the object-oriented implementation of the software and also designed and implemented major components of the user-interface. We would also like to acknowledge Gearoid Lane of Extech Ltd. who participated in early conceptual design discussions and shared his ideas on hierarchical modeling.

REFERENCES

Belanger. R. and A. Mullarney (1990), MODSIM II Tutorial, CACI Products Company, La Jolla, California.

Birtwistle, G. M., O. J. Dahl, B. Myhrhaug, and K. Nygaard (1979), Simula begin, Studentlitteratur, Lund, Sweden.

Bishack, D. and S. Roberts (1991), "Object-Oriented Simulation." In Proceedings of the 1991 Winter Simulation Conference, IEEE, Piscataway, NJ.

Concepcion, A. and S. Schon (1986), "SAM - A Computer-aided Design Tool for Specifying and Analyzing Modular, Hierarchical Systems." In Proceeding of the 1986 Winter Simulation Conference, IEEE, Piscataway, NJ.

Conrad, S., D. Sturrock, and J. Poorte (1992) "Introduction to SIMAN/Cinema." In Proceedings of the 1992 Winter Simulation Conference, IEEE, Piscataway, NJ.

Fegan, J. M., G. M. Lane, and P. J. Nolan (1991), "Introduction to Simulation Using Intelligent Simulation Interface (ISI)." In Proceedings of the

1991 Winter Simulation Conference, IEEE, Piscataway, NJ.

Goldberg, A., and D. Robson (1989), Smalltalk 80: the language., Addison Wesley, Reading, Mass.

Gordon, R.F., E. A. MacNair, P. D. Welch, K. J. Gordon, and J. F.. Kurose (1986), "Examples of Using the RESearch Queuing Package Modeling Environment (RESQME)." In Proceedings of the 1986 Winter Simulation Conference, IEEE, Piscataway, NJ.

Lane, G. M., J. M. Fegan, P. J. Nolan, and J. Flynn (1989), " A Framework for the Hierarchical Representation of Discrete Event Simulation Models using Macros." In Proceedings of the Third European Simulation Congress, Edinburgh, Scotland, SCS Europe, Ghent, Belgium.

Law, A. M., and M. G. McComas (1990), "Secrets of Successful Simulation Studies," Industrial Engineering, 22, 5.

Lomow, G., and D. Baezner (1990), " A Tutorial Introduction to Object-oriented Simulation and Sim++." In Proceedings of the 1990 Winter Simulation Conference, IEEE, Piscataway, NJ.

O'Loughlin, M., M. Meagher, and R. Harper (1988), "Simulation at Digital Equipment Corporation: The Process Expert as Simulation Expert." In Proceedings of the 1988 Winter Simulation Conference, IEEE, Piscataway, NJ.

Suri, R., and M. Tomsicek (1988), "Rapid Modeling Tools for Manufacturing Simulation and Analysis." In Proceedings of the 1988 Winter Simulation Conference, IEEE, Piscataway, NJ.

Tempelmeier, H. and Th. Endesfelder (1987), "Der SIMAN Modul Prozessor--ein flexibles Softwaretool zur Erzeugung von SIMAN-Simulationsmodellen." Angewandte Informatik, 19,2, pp. 104-110.

Thomasma, T. and O. Ulgen (1988), "Hierarchical, Modular Simulation Modeling in Icon-based Simulation Program Generators for Manufacturing." In Proceedings of the 1988 Winter Simulation Conference, IEEE, Piscataway, NJ.

Thomasma, T., Y. Mao, and O. Ulgen (1990), "Manufacturing Simulation in Smalltalk." In Proceedings of 1990 Western Simulation Multiconference, IEEE, Piscataway, NJ.

Ulgen, O. and T. Thomasma, (1990), "SmartSim: An Object-Oriented Simulation Program Generator for Manufacturing Systems." International Journal of Production Research, 28, 9, 1713-1730.

Zeigler, B. P. (1984), Multifaceted Modeling and Discrete Event Simulation. Academic Press, London.

Zeigler, B. P. (1989), Hierarchical Modular DEVS: Model Knowledge and Endomorphy in Object-Oriented Simulation. Academic Press, Boston.

AUTHOR BIOGRAPHIES

C. DENNIS PEGDEN is the president of Systems Modeling Corporation. Dr. Pegden has taught Industrial Engineering at The Pennsylvania State University and the University of Alabama in Huntsville. During his tenure at the University of Alabama, he studied simulation language design and led in the development of the SLAM simulation language. After joining the faculty of The Pennsylvania State University, he continued his work in simulation language and developed the SIMAN simulation language. Dr. Pegden received his Ph.D. in 1976 in Industrial Engineering from Purdue University where he studied optimization.

DEBORAH A. DAVIS is the vice president of software development with Systems Modeling Corporation (SM). Since joining SM in 1984, she has worked on development of SIMAN, Cinema, and other SM products. Miss Davis received B.S. and M.S. degrees in Industrial Engineering and Operations Research from The Pennsylvania State University. Her current interests include simulation language and user interface design, new applications of simulation technology, and object-oriented methodologies. She served as the Business Chair for the 1991 WSC and will be the General Chair of the 1994 WSC. She is a member of the SCS, IIE, Tau Beta Pi, and Alpha Pi Mu.