Technical Report
CMU/SEI-93-TR-14
ESC-TR-93-192

Carnegie Mellon University
Software Engineering Institute

Structural Modeling: An Application Framework and Development Process for Flight Simulators

Gregory D. Abowd
Len Bass
Larry Howard
Linda Northrop

August 1993

Table of Contents

Technical Report
CMU/SEI-93-TR-14
ESC-TR-93-192
August 1993

Structural Modeling: An Application Framework and Development Process for Flight Simulators

Gregory D. Abowd
Len Bass
Larry Howard
Linda Northrop

Real-Time Simulators Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

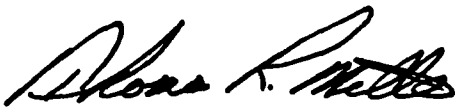
SEI Joint Program Office
ESC/ENS
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER



Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

The Software Engineering Institute is sponsored by the U.S. Department of Defense.
This report was funded by the U.S. Department of Defense.

List of Figures

Structural Modeling: An Application Framework and Development Process for Flight Simulators

Abstract: In this paper, we present the structural modeling approach, an application framework and development process for the construction of flight simulators. Structural modeling was developed to address functional, nonfunctional, and process requirements for flight simulators. It has been successfully used in the development of large scale (one million lines of Ada code) flight simulators for the United States Air Force. A structural model promotes a simple and coherent software architecture with a small number of specialized structural elements obeying a few systemwide coordination strategies. It is this simplicity coherence of the software architecture that enables analysis to demonstrate the quality of the system.

1 Introduction

1.1 Historical Motivations

Structural modeling is an object-based software engineering strategy developed by the collaborative efforts of the United States Air Force, Air Force contractors, and the Software Engineering Institute (SEI) to address the problems associated with the development and evolution of flight simulator software. Flight simulators train pilots for flight missions in a safe, convenient, and cost-efficient way. To effectively provide this training, the software for a flight simulator must work with the hardware to faithfully reproduce the behavior of the aircraft being simulated. Flight simulator software, therefore, must perform in real time, be developed and continually altered to keep pace with the technological advances in the simulated aircraft, and undergo a validation process that certifies acceptability as a pilot training device. In addition to this inherent complexity, the flight simulator software is typically very large in scale, in the range of one million lines of high-level language code.

Work on structural modeling began in 1986 when Air Force engineers recognized that the traditional software architecture for flight simulators was reaching its limits. The scale and complexity of the software had precipitated numerous problems in both the developed product and the development and evolution processes. Modifications to the simulator software were severely lagging behind modifications to the aircraft being simulated, resulting in a software product that did not faithfully simulate the current aircraft. As is often the case in large-scale software development efforts, geographically remote software teams were concurrently developing parts of the flight simulator system. The time required to integrate these parts developed by diverse work teams was growing at alarming rates. The correction of errors in the simulator software was complicated and time-consuming. Modifications to add functionality were also time sinks. Often the cost of modifications to the software exceeded the software development cost.

Moreover, certain flight behaviors were becoming increasingly difficult to simulate because the organization of functionality in the simulator was different from that in the physical aircraft. A good example of this was in the introduction of malfunctions into the simulation. Malfunctions in the physical aircraft, such as the failure of a pump in the hydraulics system, propagate their effects up to higher level systems of the aircraft; so the pump failure might lead to a failure in a hydraulically-controlled actuator for the air brake, which then renders the air brake inoperable. The architecture of the simulation, however, did not explicitly represent the connection between the pump, the hydraulics system, the actuator, and the air brakes. Rather, the simulation would separately model the hydraulics system and the braking system, with no model for the actuator itself. An accurate portrayal of this malfunction would, therefore, require changes to both of these simulation models. Other malfunctions were even more complex, involving more simulation models whose logical connection in the physical aircraft was not represented explicitly in the simulation architecture.

The unwieldy software architecture reduced effective communication of the design in reviews and interactions among work team members. Communication within a design team depends on shared concepts and representations. Ad hoc solutions to simulation problems written in overly flexible general purpose programming languages were rampant in the design of the system and they were mostly concealed from system-level designers. Communication about the system with the domain experts and users was even more difficult. Ultimately, this lack of effective communication decreased the control and visibility of the software to the point where it registered technical risk in development and maintenance.

The recognition of this technical risk was the catalyst that drove the development of the structural modeling method. The broad objective behind structural modeling was to take a problem domain of great complexity and scale and to abstract it to a coarse enough level to make it manageable, modifiable, and able to be communicated to a diverse user and developer community. As a result of the collaboration between the SEI, the Air Force, and Air Force contractors, a culture of structural modeling has evolved. Structural modeling experience has been gained in a number of recent simulator acquisitions, including the B-2 Weapon Systems Trainer, the C-17 Aircrew Training System, and the Special Operations Forces Aircrew Training System. While data specific to those acquisitions are not generally available, there have been some internal reports within the SEI and Air Force that have described portions of the object-based technology underlying the structural modeling approach [Lee 88, USAF 93]. In addition, the Real-Time Simulators Project at the SEI is currently drafting a guidebook describing in great detail structural modeling as it applies to the development of an air vehicle within a flight simulator, specifically addressing the case study of the T39A flight simulator.

The theory and practice underlying structural modeling is now sufficiently mature to warrant its description for an audience outside the flight simulation domain. Though all examples in this paper refer to this domain, the concepts and practices of structural modeling should interest all whose main concern is the engineering of large-scale object-based systems. This paper provides an account of our experience for the scrutiny of the software engineering community

and puts forth a position that a simple and clearly defined software architecture is essential for the development of large systems that satisfy both functional requirements and nonfunctional qualities.

1.2 Overview

In this paper, we distinguish between a *structural model*, an application framework for flight simulators, and the *structural modeling process*, the means by which the application framework is engineered into a complete system. In Section 2, we discuss the requirements relevant for the construction of a flight simulator and highlight those that drive the definition of a structural model and the structural modeling process. In Section 3, we define the structural model for the air vehicle system of a flight simulator in terms of the structural elements, or classes of the model, and the coordination, or structural relationships, that exist between the elements for control flow and data exchange. In Section 4, we outline the activities involved in the structural modeling process. We conclude in Section 5 with a summary of the outcomes of our experience with structural modeling and the open issues that will direct our future research.

2 Requirements

The structural modeling method had to result in

- A software *product* that would satisfy various requirements and embody certain nonfunctional qualities that would considerably diminish the technical risk.
- A *process* that would be visible and understandable.

In addition, since the flight simulator software would be written in Ada, it was important that the structural modeling method yield a design that could be readily implemented in an object-based language such as Ada. We will now discuss the influence of these separate classes of requirements on the development of structural modeling for flight simulators.

2.1 Systemic Requirements

Systemic requirements are those requirements that arise because the simulator is operating on a computer. These involve the management of computer resources used to enable the simulator to respond to actions of the crew in real time. Time must be managed within the flight simulator, and must therefore be coordinated across the various components of the software. Different calculations within the simulation must be performed at different rates in order to achieve realistic performance. Because flight simulators are typically implemented on tightly coupled multiprocessors, part of managing time is coordinating the computations of the different processors.

2.2 Modeling Requirements

Modeling requirements are introduced because the software must execute simulation models to mimic the behavior of the real aircraft. The way the software is modularized, or partitioned, affects the simulation models that are used. For example, if the simulation model for the landing gear involves both the tires and the shock absorbers, the software that implements the model must have knowledge of the characteristics of both tires and shock absorbers. Fidelity, what features and characteristics of the real-world domain are simulated, and accuracy, the match between simulated behavior and real-world observations, are the essential modeling requirements.

2.3 Mission Requirements

Mission requirements are introduced by the set of capabilities that the simulator is intended to support for training. The level of detail of simulation models required for a particular flight simulator depends on the specific training mission. Most mission requirements, such as the need to introduce malfunctions to train the crew for abnormal situations or to provide playback capabilities for review, recur from system to system.

Of the above three classes of requirements, it is the systemic and selected mission requirements that drive the definition of the structural model, because they are common to all simulators. The modeling requirements are volatile, especially across different flight simulators. Since a wide variety of simulation models must be supported, no single one can dictate a structural model. The commonality of systemic requirements and some of the mission requirements, on the other hand, can be factored into the high-level design decisions of the structural model.

2.4 Nonfunctional Qualities

Satisfying the above requirements is a natural objective for our structural model, but it is not a novel objective for a simulator architecture, nor was it the initial motivation for our structural modeling approach. The initial motivation for the development of structural modeling came from system deficiency in nonfunctional qualities. Such qualities are only indirectly visible to the end user but directly alleviate technical risk associated with the development and evolution of a large, complex system such as a software simulator. Whereas the earlier requirements we discussed can be validated by appeal to the user community, these nonfunctional qualities are validated by the development team itself. Safety, reliability, maintainability, modifiability, integrability, extensibility, and usability are nonfunctional qualities that are desirable in flight simulator software.

Maintainability and integrability were considered to be top-priority considerations, for reasons discussed in Section 1. Maintainability, which in a broad sense encompasses modifiability and extensibility, is the ability to make necessary corrections, modifications, and enhancements. Integrability is the ability to integrate easily the output of different work teams. As Section 1 suggests, the evidence we use to support our claims that earlier simulator designs were difficult to maintain and integrate is entirely anecdotal. We do not have access to any empirical evidence that directly ties the cost of changes to the software to the architectural design of these systems. We rely entirely on the consensus of the flight simulator community, and that consensus is fairly clear that by 1986 much improvement was needed in the maintainability and integrability of flight simulator software.

2.5 Process Requirements

A software engineering approach to the development of a large, complex system such as a flight simulator must involve a process that is visible, reviewable, repeatable, and documented. It was essential to establish a process that made the overall structure of the software explicit early, so that developers could work with domain experts and simulation analysts to assess both adherence to functional requirements and the existence of the nonfunctional qualities. It is imperative to be able to abstract away the unnecessary details. Communication feedback loops among work team members and between developers and the whole gamut of users are essential to the alleviation of technical risk and the success of the software project. The structural modeling process had to facilitate this necessary communication.

3 The Air Vehicle Structural Model

3.1 Definition

A structural model is a a reusable collection of classes of differing levels of abstraction providing the basis from which the flight simulator software is derived. This characterization corresponds nicely to the definition of application frameworks discussed in the object-oriented literature [Johnson 91]. For historical reasons, we refer to individual classes in the structural model as *structural elements*.

Figure 3-1 depicts the structure of a generic flight simulator. In this paper, we restrict our attention to the description of the air vehicle.

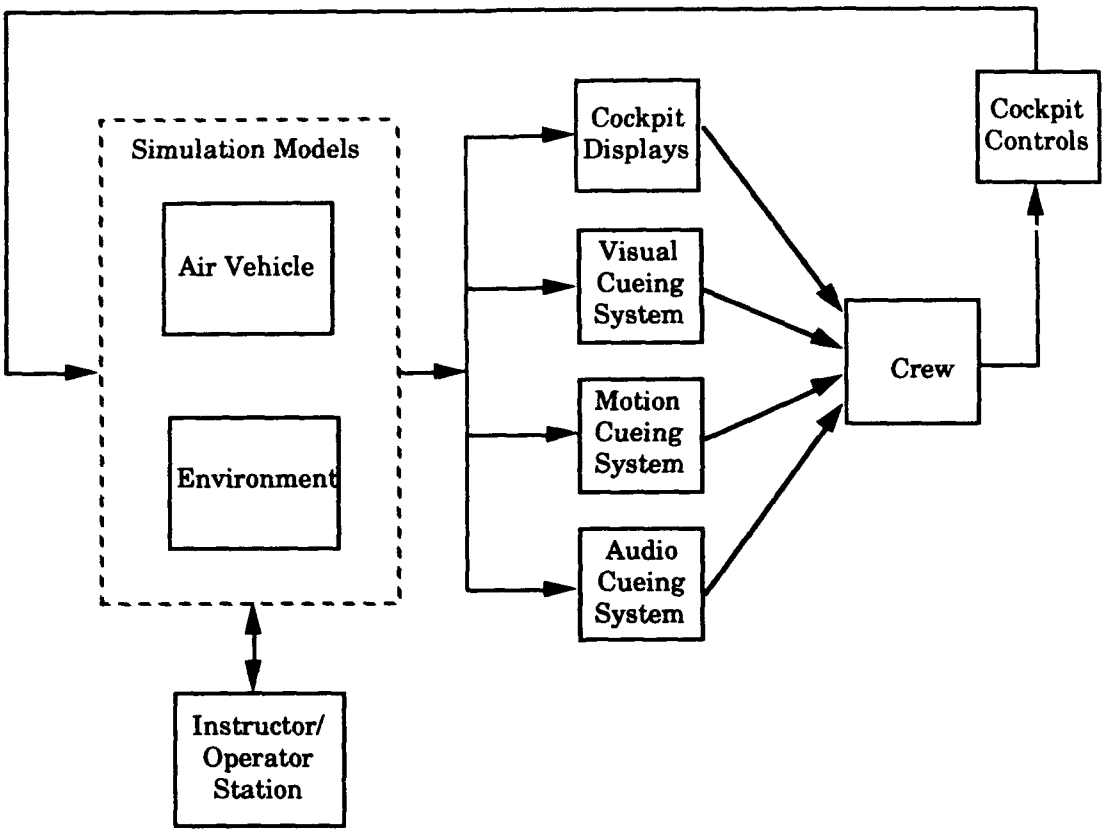


Figure 3-1: A Generic Flight Simulator

Figure 3-2 depicts the air vehicle structural model (AVSM) for the flight simulator, the details of which will be described in this section. In this figure we have chosen to expose the structural elements of the AVSM as named boxes. In addition we show the data and control relationships that exist between the various structural elements.

The simulation within the air vehicle is based mainly on a number of periodic calculations that are repeatedly processed within some simulation time interval, or frame. In an ideal situation, a nonpreemptive scheme is used on a single process to schedule the various calculations within a cyclic executive. There might be situations in which it is not possible to schedule all calculations within the simulation frame, and so the processing load is segmented across multiple frames but still within a single process. To avoid the overhead of segmentation, a nonpreemptive scheduling scheme can be used to distribute calculations across multiple processes. Introducing multiple processes also eases the burden of scheduling nonharmonic calculations. In addition to easing the scheduling burden, multiple processes can be distributed across multiple processors to increase the speed of the overall simulation. The AVSM is the structure for a single process within the air vehicle. Multiple processes will have similar AVSMs within them, the only communication being a synchronization between the cyclic executives and a distributed shared memory, both of which we will describe in more detail later.

The AVSM is divided into an application level and an executive level. The application level contains most of the modeling information needed to simulate a given air vehicle and is constructed so as to most closely mimic the construction of the physical aircraft. The executive level of the AVSM is concerned with

- The execution of the simulation in real time.
- The interface to an instructor/operator who controls the activity of a given training mission.
- The procedures by which data integrity is preserved in a potentially distributed multiprocessing software platform with shared resources.

The executive-level elements focus control within a given process to a single thread.

The key distinction between application- and executive-level structural elements is in their level of abstractness and number of instances. Application-level structural elements are fully abstract; that is, no instance variables are declared and operation definitions are deferred until instantiation. Executive-level structural elements are not as abstract, as most of their behavior can be defined in the class definition, the only difference among instances being contained in data, or instance variables.

Five structural elements of the AVSM are portrayed:

1. Component
2. Subsystem controller
3. Periodic sequencer
4. Event handler
5. Timeline synchronizer

We will discuss the role and some of tne important features of each of these five structural elements, starting at the bottom with *component* and working upwards to the *timeline synchronizer*. Throughout our discussion, we will try to separate the conceptual issues involved in the structural model from implementation issues arising because the simulators are built on shared memory machines using Ada.

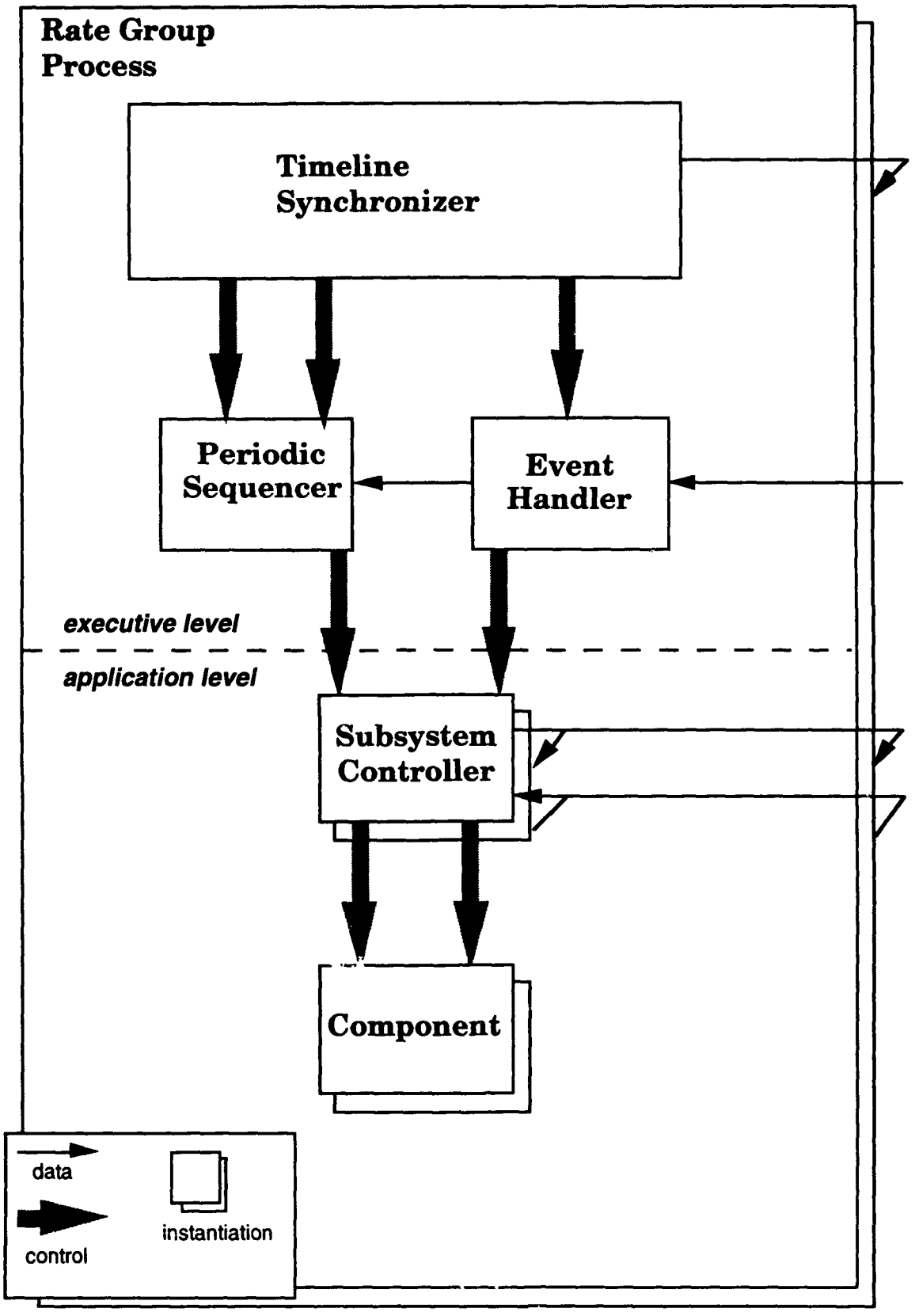


Figure 3-2: The Air Vehicle Structural Model (AVSM)

3.1.1 Components

In general, we find that the physical aircraft is organized into single item *components* with very low-level functionality (such as pumps, valves, regulators, switches, relays, etc.) and aggregated components, or *subsystems*, which serve higher level functions within the aircraft (such as flight control or hydraulics). Components in the AVSM are the structural elements that realize the simulation models of the low-level physical components. Each component encapsulates a set of variables that represents the state of the simulation model. Figure 3-3 presents a more detailed view of the component structural element.

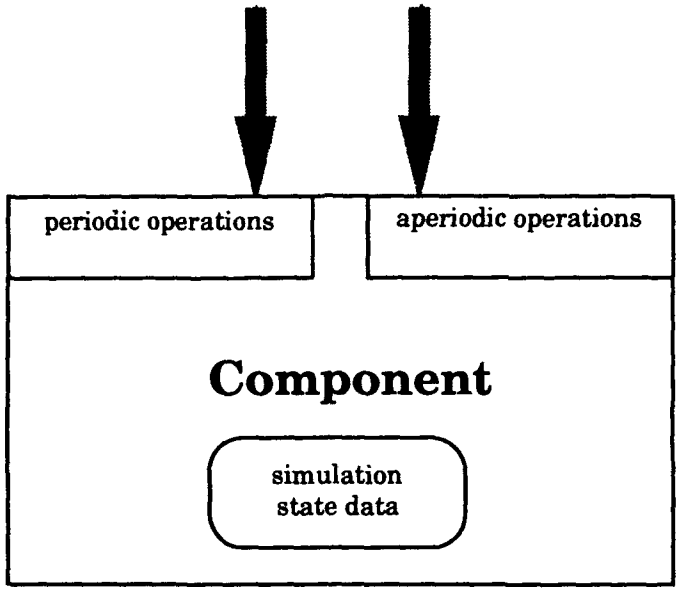


Figure 3-3: The Component Structural Element

There are two types of operations by which the state of a component can be altered: periodic and aperiodic (or event-driven). `Update` is the only periodic operation, and it is used to request that a component should alter its encapsulated state variables to reflect the passage of some simulation time interval, or frame. The inputs that the component requires to perform the update are provided as parameters of the operation request, and the values it produces as outputs are returned as a response to the request. In this way, the component is isolated from all other components in the system. The aperiodic, or event, operations are `initialize`, `set_parameter`, and `process_malfunction`, and these are used sporadically during the course of the simulator's operation based on requests from the instructor/operator. These operations will require only input parameters.

3.1.2 Subsystem Controllers

As described earlier, subsystem controllers are structural elements used to manage a cohesive collection, or ensemble, of components. Subsystem controllers act as an interface between the components they manage and the rest of the system. They also serve as the basic

units used for the allocation of software to computational resources. Subsystem controllers encapsulate a set of variables used to represent the state of the components in the ensemble. This encapsulation hides the existence of specific components from all other components of the simulation. The subsystem controller structural element is depicted in greater detail in Figure 3-4.

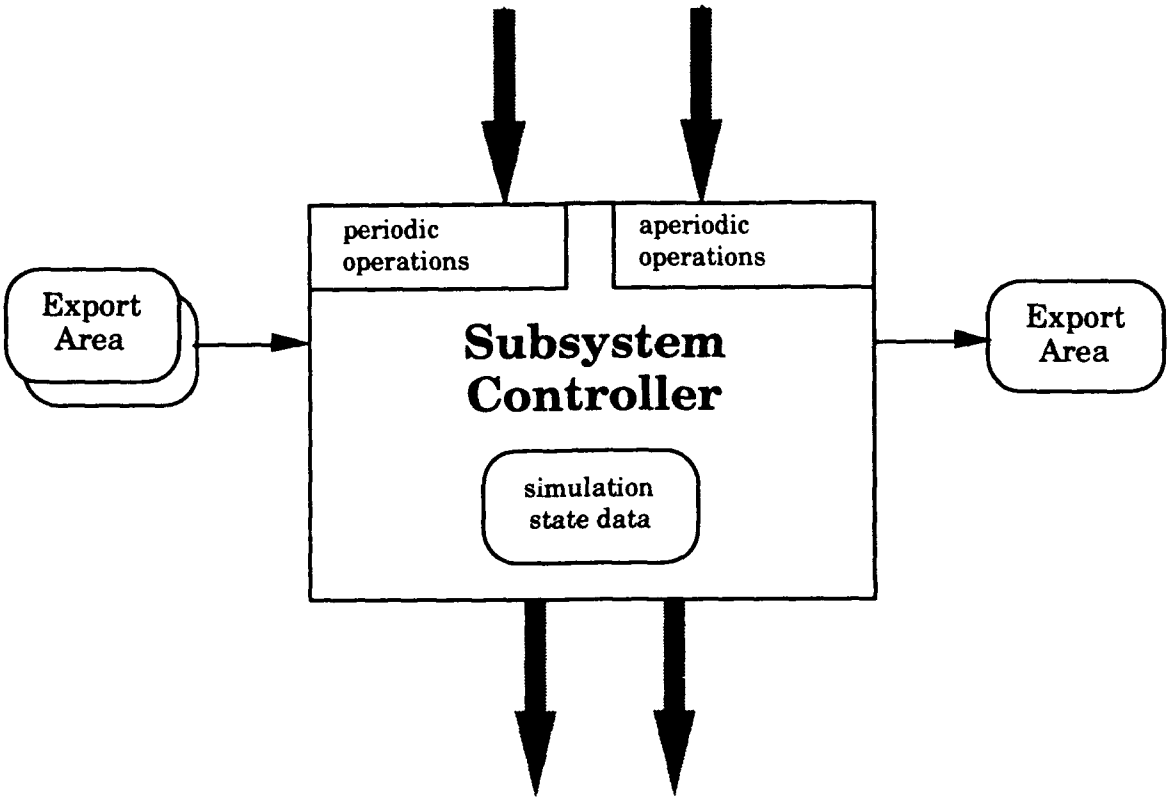


Figure 3-4: The Subsystem Controller

Like components, subsystem controllers have periodic and aperiodic operations. The periodic operations include `update` as well as two additional operations, `import` and `stabilize`. `Update` is used to request that the subsystem controller update the state of its ensemble. The `import` and `stabilize` operations provide ways to deal with data exchange between subsystems to ensure data consistency and stability of the simulation model. The `stabilize` operation results in communication from the subsystem controller to the periodic sequencer, explaining the upward pointing data connection arrow in Figure 3-4. The aperiodic operations of the subsystem controller include those of the components and two additional ones, `hold_parameter` and `configure`, again driven by instructor/operator requests. The effects of the aperiodic operations are typically achieved by invoking a particular component through one of its aperiodic operations. For example, the `process_malfunction` operation invoked

by the operator results in a request to the appropriate component where the malfunction resides. The `hold_parameter` and `configure` operations are used to request particular changes to the variables encapsulated by the subsystem controller.

Unlike components, which receive all information about the rest of the simulation through values passed as parameters to their operations, subsystem controllers can exchange information with other subsystems, both those within the same process and those in other processes, through *export areas*. These export areas have a single owner/writer, but many readers. They can be implemented, for example, by means of a distributed shared memory. The `import` operation signals the subsystem controller to access data from the export areas of other subsystem controllers. The result of the `update` operation is that the data in the subsystem controllers own export area is modified to reflect the current state of the subsystem for others to reference.

3.1.3 Periodic Sequencer

The periodic sequencer is a structural element that manages all periodic processing of the flight simulator structural model for a given process. The periodic sequencer structural element is depicted in greater detail in Figure 3-5.

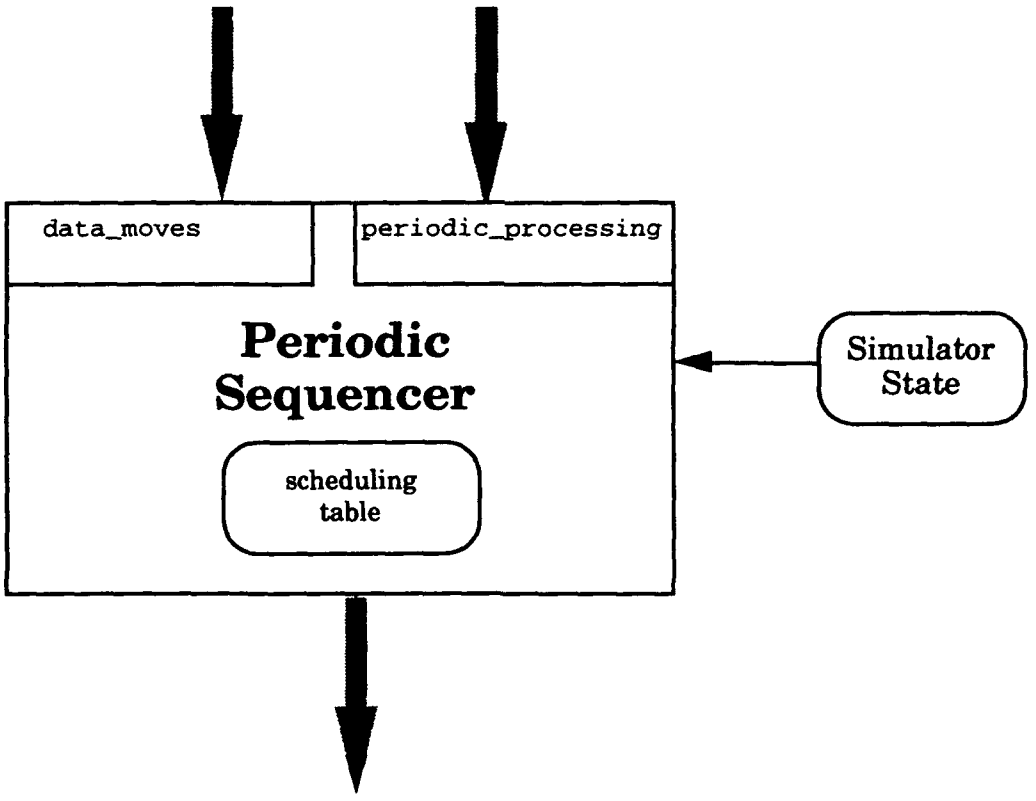


Figure 3-5: The Periodic Sequencer

This structural element is responsible for the nonpreemptive scheduling of all periodic processing of the subsystem controllers upon invocation of its `periodic_processing` operation. Which subsystem controllers to invoke during each scheduling interval (frame), and which of their periodic operations to use, is determined by each subsystem's periodic rate and the simulator's overall operating state. Schedules are precomputed based on worst-case execution behavior and stored in a scheduling table within the periodic sequencer. The periodic sequencer is also responsible for controlling the exchange of data through export areas of its subsystem controllers. This activity is invoked by means of the periodic sequencer's `data_moves` operation. The scheduling table also holds scheduling information for data accesses of the subsystems.

3.1.4 Event Handler

The event handler is at the same level as the periodic sequencer. It is a structural element that manages all aperiodic processing of the flight simulator structural model within a given process. The event handler structural element is depicted in greater detail in Figure 3-6.

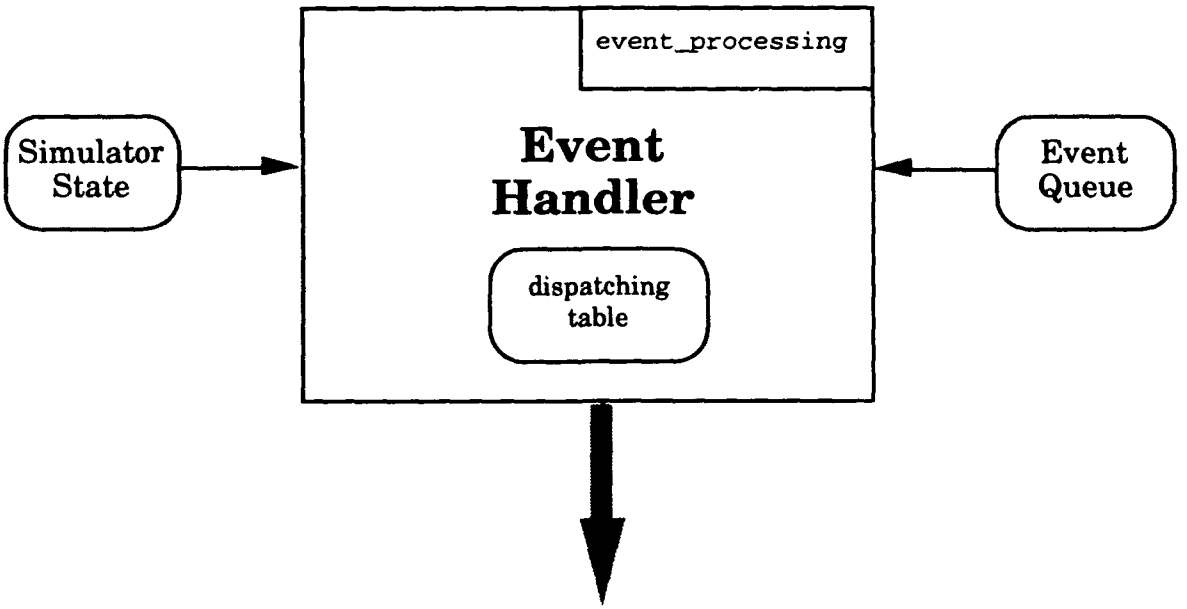


Figure 3-6: The Event Handler

The primary purpose of the event handler is to manage the interface with the instructor/operator station (IOS). Events generated by the instructor/operator indicate whether to begin or end the simulation or to inject a fault or malfunction into one of the subsystems or components. The IOS places events on an event queue, and the event handler takes these events, as a result of invoking its `event_processing` operation, and dispatches control to the

appropriate subsystem controller in response. The event handler uses a dispatch table to organize the invocations of the particular subsystem controllers according to the event being processed. Routing information, which subsystem controller to invoke and by which of its aperiodic operations, is passed as part of the request from the IOS.

The instructor/operator is in charge of setting the overall state of the simulation by announcing certain events, such as one to freeze the simulation. Once the event handler processes such an event, it must communicate to the periodic sequencer the current overall state of the simulation, since that is one of the indices into the periodic scheduling table.

3.1.5 Timeline Synchronizer

At the highest level of the flight simulator structural model is the timeline synchronizer element, which is a cyclic executive. The timeline synchronizer structural element is depicted in greater detail in Figure 3-7.

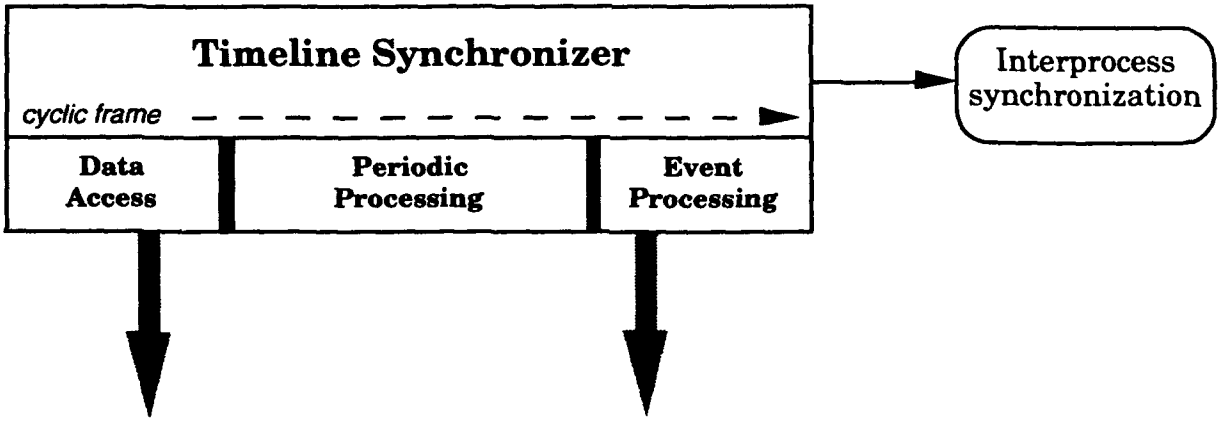


Figure 3-7: The Timeline Synchronizer

Time on each processor is divided into intervals called *cyclic frames*. The timeline synchronizer is responsible for synchronizing the start of frames, and periods within frames, with other processes. It initiates the synchronized exchange of data between subsystem controller export areas (through the `data_moves` operation of the periodic sequencer) and invokes the periodic sequencer and event handler to perform their processing at appropriate times during the frame, as indicated in Figure 3-2. In a multiple process configuration, one process assumes the role of the master timeline synchronizer and runs at the highest frame rate of all processes. This master timeline synchronizer uses interprocess synchronization (implemented, for example, as semaphores) to coordinate the starts of the frames, and periods within those frames, among all of the processors.

3.2 Coordination Model

The fundamental relationships that unite the structural elements of the AVSM constitute the coordination model of the flight simulator structural model [Gelernter 92]. We consider it a fundamental exercise in the description of any structural model to make explicit and separate the coordination mechanisms. A simple and coherent coordination model is an important feature of a software architecture that enhances its analyzability.

One of our design goals was to reduce coupling between structural elements, so that the effects of changes to one structural element are minimized across the system. In the AVSM, this goal has been achieved by the stipulated requirement that control must flow in a strictly downward manner. The AVSM embodies a number of subordinate relationships, that is, control relationships of the form "A calls B." These subordinate relationships are depicted in Figure 3-2 by the thick gray arrows and further emphasized by the relative vertical positioning of the structural elements. Control relationships are sometimes accompanied by data communication, as is the case when an operation of a subordinate element is called with parameters. Communication relationships are identified by the thin black lines in Figure 3-2. The subordinate element provides operations that fully define the control and communication relationships with its superior. Thus, interpreting the control and communication relationships in Figure 3-2 reveals that no data is communicated between the timeline synchronizer and both of its direct subordinates, the periodic sequencer and the event handler. The subsystem controllers directly control all components within their subsystem and data is communicated in both directions given that the `update` operation of the component takes input and output parameters.

In addition to this strictly top-down flow of control and communication, there are some additional mechanisms for communication between structural elements. These mechanisms are supported through data interfaces and are used to indicate coordinate rather than subordinate relationships. Four such relationships were described in the discussion of the individual structural elements:

1. Between instances of subsystem controllers, using export areas.
2. Between the instructor/operator and the event handler, using the event queue.
3. Between the event handler and the periodic sequencer, using a variable to indicate the overall simulator state.
4. Between the master timeline synchronizer and its slaves, using some interprocess synchronization mechanism (such as semaphores).

The organization of the timeline synchronizer is mainly to facilitate these various coordinate data exchanges. It ensures temporally consistent data communication between subsystem controllers by synchronizing accesses to the export areas at times when the simulation models are quiescent. Further, the organization of the timeline can be used to synchronize access to the data interface between the periodic sequencer and the event handler, as well as between the event handler and the instructor/operator station.

It is important that a small number of subordinate and coordinate structural relationships are used to describe the relationships between many instances of structural elements. This leads to an architecture with many instances of structural elements, but with a limited number of relationships between those elements. This architecture is easier to understand than the more traditional architectures used for flight simulators of the past. The behavior of each software instance, and the kinds of relationships the instance has with other instances, can be predicted from its form; that is, from the structural element upon which it is based. Likewise the behavior of the system as a whole, and its relationships with other parts of the simulation (for example, the instructor/operator station) can be predicted from its form; that is, from the structural model as a whole.

4 Analysis of the AVSM

The primary purpose of a structural model is to facilitate the prediction of how well the completed system will satisfy functional, nonfunctional, cost, and schedule requirements. In this section, we will explain how the AVSM satisfies the various requirements and nonfunctional qualities outlined in Section 2.

4.1 Systemic Requirements

The main systemic requirement is to reduce latency within the computer simulation to make the training simulator responsive in real time to the actions of the trainee pilot. The key to satisfying this requirement is to be able guarantee schedulability of the periodic processing within a satisfactory time interval. The bulk of the scheduling task to ensure that timeliness of state changes throughout the system is still left to the simulation designer. However, the AVSM supports this task by centralizing all scheduling information within the periodic sequencer structural element. As we discussed earlier, scheduling within a single process is performed nonpreemptively, in the manner of a cyclic executive. To ensure timeliness, strict adherence to such a scheduling policy would require segmentation of slower rate processing with an associated overhead. Distributing the structural model over several processes allows for preemptive scheduling using a rate monotonic priority assignment [Klein 93].

The separation of periodic processing from event handling at the executive level eliminates the need for subsystem controllers and components to poll their environment for significant aperiodic events. The low incidence of these aperiodic events relative to the periodic processing in the domain makes this separation possible.

4.2 Modeling Requirements

The main modeling requirements involve the fidelity and accuracy of the simulation with respect to the physical aircraft. These requirements are the most volatile from simulator to simulator, so we did not construct the AVSM with the intent to satisfy these requirements. However, the organization of the application level of the AVSM into subsystems and components aids in fulfilling fidelity and accuracy requirements from one simulator to the next. Fidelity requirements dictate the level of granularity for partitioning the simulator into subsystems that correspond to subsystems of the physical aircraft. Accuracy requirements dictate the kinds of simulation models that must be supported within components. Therefore, whereas the AVSM was not constructed to satisfy any particular set of fidelity and accuracy requirements, it provides an appropriate language for expressing them case by case.

4.3 Mission Requirements

Only some of the mission requirements are common across all simulators. One of those requirements is the ability of the instructor/operator to introduce malfunctions to the aircraft during otherwise normal operation. The AVSM strongly supports the simulation of malfunctions that can be entered at arbitrary times by the instructor/operator. On the physical aircraft, the locus of a malfunction is one of its components. Since the AVSM is structured similarly to the physical aircraft, malfunctions in the simulator can also be introduced at the component level. Not only does this provide a more natural mechanism for propagating the effect of a malfunction across many subsystems, it also has resulted in a more faithful simulation of those effects to the pilot trainees.

4.4 Nonfunctional Qualities

4.4.1 Modifiability

Modifications to the simulator arise because of changes to the physical aircraft. The changes to the physical aircraft are generally accomplished through modifications to its physical components. The close correspondence between the high-level designs of the simulator and physical aircraft, therefore, improves the maintainability of the simulator.

The allocation of functionality of the physical aircraft to the AVSM application-level structural elements is analyzed for two characteristics, coverage and modifiability. In our case, coverage implies that all subsystems of the aircraft should be allocated by the partitioning; that is, given some subsystem of the aircraft, its functions should be identifiable in an instance of a subsystem controller. Modifiability is analyzed by examining typical modifications and by determining the data coupling present among the subsystem controllers. For each typical modification a measurement is taken that equals the number of different components that must be altered to achieve the modification. This measurement quantifies the difficulty of making the particular modification. An analysis of the collection of such measurements yields a systemwide estimate for modifiability.

The data coupling among the subsystem controllers is measured using an N-squared chart. This chart has all of the subsystem controllers along both axes with an X in every box in which data flows from one controller to another. The density of this matrix indicates coupling of the entire system. Systems that have high coupling (indicated by a dense N-squared matrix) are difficult to modify. Note that this argument only holds for modifications affecting the interface of structural elements.

4.4.2 Integrability

Because simulators are very large software systems, it is common for them to be developed by large and distributed design teams. In such a situation, integrability of separate design modules is a key to reducing technical risk. The simplicity of the structural model is key to supporting the integrability. The AVSM has very few types of structural elements, and each has a very

fixed interface with the rest of the system and a fairly rigid internal structure. The type of a structural element dictates its internal form and its external appearance. Ultimately, this rigidly defined architecture leads to a mechanical process for deriving the implementation of a simulator from its design. While this implementation process can by no means be completely automated—for instance, scheduling must be done by hand—the structural modeling process includes specification forms and code templates that greatly support principled code generation and integration. Some aspects of the structural modeling process are discussed in Section 5. Greater details on structural modeling are currently being documented by the SEI.

4.4.3 Communicability

Because the structure of the software simulator now closely resembles the physical aircraft, the allocation strategy described above will enhance the communication between designers and domain experts. The partitioning decisions between one component and another have already been made by the engineers who designed the aircraft. There is a wealth of knowledge in the industry concerning the design of the physical aircraft, and the design of the air vehicle should benefit from this accrued design knowledge. Basing the software on these partitions, therefore, simplifies software decomposition.

Within the design team, again the simplicity of the structural model facilitates communication. The strictly enforced simple coordination model eliminates most ambiguous relationships between structural elements, and the type of structural element dictates its form.

5 Structural Modeling and Other Life Cycle Activities

The structural model provides an application framework for the air vehicle portion of a flight simulator. The development process that takes a structural model into a complete system is called *structural modeling*. A more detailed description of structural modeling is currently being written by the SEI. The activities required in structural modeling are

- Structural model design and analysis
- Preliminary specification
- Preliminary review
- Detailed specification
- Detailed review
- Incremental implementation

The main focus of this paper has been on the structural model design and analysis activity. In this section we briefly outline the other life cycle activities.

5.1 Specification and Review

We assume at this point that there has been an analysis of the partitioning of physical aircraft subsystems and components to application-level structural elements satisfying the modeling and mission requirements for the intended simulator. A simulation analyst can provide the details of the models to be used for each instance of the subsystem controller or component. The simulation analyst provides this information in two levels of details, at different times in the process: the preliminary design review (PDR) and the critical design review (CDR). The information provided is captured in a *specification form instance* that becomes part of the structural model for the system. The specification form can be viewed as a structured language that permits a description of the system design. Using this language, the simulation analyst records information about subsystem controllers or components being developed. The set of all specification form instances will therefore contain all of the modeling design elements for the system.

The information in the specification form evolves between the two design review periods. The preliminary form of the subsystem controller specification, for example, will have largely textual descriptions. Little is known about the components at PDR, except for an identification of what components there might be. At CDR, the specification forms will have mathematical or algorithmic descriptions of all subsystems and components. Thus, the preliminary form will discuss at what fidelity the subsystem is to be modeled, and the later form will provide the model itself. Most significantly, the specification forms provide a structured means of communication that directs the dialogue between those specifying the simulation and those implementing it.

5.2 Implementation

To convert the specification into an implementation, there are two activities that must be accomplished:

1. The structural elements must be written as code from which instances can be formed.
2. The specific information from each specification form template must be converted into executable code within the framework of the structural element code.

The class definition in an Ada implementation of a structural model is called a *template*. Templates are the code fragments associated with the structural elements. The information provided by the simulation analyst on the specification form is used to define the functional methods for the behavior and thereby augment and complete the abstract definitions found in the structural elements.

For the purposes of this paper, we present structural modeling as if it were a staged, top-down process. In reality, the development of a structural model is much like any other design activity in that a design represents a current hypothesis, which is tested by considering a collection of representative instances of the hypothesis and continually refining and testing the design. The structural modeling process assumes the existence of a structural model and proceeds by developing a component partitioning and translating this partitioning into an implementation.

6 Conclusions and Future Work

The development of the air vehicle structural model for flight simulators and the use of the structural modeling process has proven successful in reducing the technical risk that motivated the research. Strict adherence to some simple coordinating mechanisms between a small set of structural elements provides us the ability to analyze the software structure and demonstrate that it satisfies various requirements and nonfunctional qualities. Specific benefits realized in flight simulator software development from the structural model described or

- Separating the coordination model from partitioning strategy allows embodiment of the coordination model into structural elements.
- The use of the same coordination model across the total system allows integrability of independently developed software components.
- Systemic and the particular mission requirements that drive the structural model are generic across flight simulators. Considerable design reuse is made feasible. Code reuse is restricted to the level of the component.
- The modeling and remaining mission requirements are more volatile than the other requirements, and it is appropriate that they are no longer the main drivers of the software design.

The current demand for software engineering approaches to object technology is great [Berard 93]. Structural modeling is such an approach. A repeatable, visible process, a chronology or life cycle, and a workable team organization are specified. Structural modeling successfully facilitates the necessary communication among developers, experts, and users. The uniformity of elements allows description of system behavior to be captured in specification forms that are easily understood by simulation analysts. Structural modeling has provided a discipline for consistently evaluating and enforcing design decisions across a large project. Though we have not discussed it in this paper, human and resource usage can be estimated throughout the process through the generation and analysis of prototypical and synthetic instances of elements.

In order to guide flight simulator software designers in the use of structural models and structural modeling, the SEI is preparing a structural modeling guidebook. The guidebook focuses on the most fundamental structural modeling practices that have already been validated. The process definition continues to evolve, however, and certain technical issues are still being addressed. For example, some of the issues related to how components should be partitioned, especially in the threat/environment area, do not yet have formulated answers. The prospect of a future object-oriented Ada and, with it, the capability of inheritance is another issue that might influence some modification in the current structural model. Perhaps the most apparent open issue, however, is to what extent the benefits of structural models and structural modeling are specific to flight simulators and to what extent could they have wide applicability to other problem domains.

We believe that the structural model is a good example of a clearly defined software architecture. As such, it should prove to be a useful model for the study of software architectures in general. For example, Garlan and Shaw, in their seminal introduction to software architectures, discuss the importance of applying different software architectural paradigms to the interpretation of the same software system [Garlan 93]. Flight simulators provide an excellent case study. The AVSM is an example of a software architecture for a flight simulator that reflects the composition of the physical aircraft. Another useful architecture for a flight simulator is one that reflects the control loops inherent between various subsystems and the pilot. In fact, the earliest software architectures for flight simulators were based on these control loops. This architectural paradigm is similar to that of a process control system. It would be an interesting exercise to demonstrate the correspondence between the AVSM and a control loop architecture.

7 Acknowledgments

The authors would like to acknowledge the contributions of the various members of the Real-Time Simulators Project at the SEI, the Air Force, and its contractors toward our current understanding of structural modeling.

References

[Berard 93] Berard, E.V. *Essays on Object-Oriented Software Engineering,* Volume 1. Englewood Cliffs, N.J.: Prentice-Hall, 1993.

[Garlan 93] Garlan, D. & Shaw, M. "An Introduction to Software Architecture." *Advances in Software Engineering and Knowledge Engineering,* Volume 1, World Scientific Publishing Company, 1993.

[Gelernter 92] Gelernter, David & Carriero, Nicholas. "Coordination Languages and Their Significance." *Communications of the ACM 55,* 2 (February 1992): 97-107.

[Howard 93] Howard, Larry & Bass, Len. "Structural Modeling for Flight Simulators," pp. 876-881. *Proceedings of the 1993 Summer Computer Simulation Conference – SCSC '93.* Boston, MA: July 1993.

[Johnson 91] Johnson, R. E. & Russon, V.F. *Reusing Object-Oriented Designs* (Technical Report UIUCDCS-R-91-1696). University of Illinois at Urbana-Champaign, May 1991.

[Klein 93] Klein, M.; Ralya, T.; Pollak, B.; Obenza, R.; & Harbour, M.G. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems.* Boston, MA: Kluwer Academic Publishers, 1993.

[Lee 88] Lee, K.; Rissman, M.; D'Ippolito, R.; Plinta, S.; & Van Scoy, R. *An OOD Paradigm for Flight Simulators,* 2nd Edition. (CMU/SEI-88-TR-30, ADA204849). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1989.

[Meyer 88] Meyer, B. *Object-Oriented Software Construction.* New York, NY: Prentice-Hall, 1988.

[SEI 92] Software Engineering Institute. "Reducing Technical Risk with Structural Modeling," *Bridge* (a quarterly publication of the SEI), December 1993.

[USAF 93] United States Air Force. *An Introduction to Structural Models* (USAF ASC-TR-93-5008). Dayton, OH: Wright-Patterson AFB, 1993.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-93-TR-14	5. MONITORING ORGANIZATION REPORT NUMBER(S) ESC-TR-93-192
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute 6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213	7b. ADDRESS (city, state, and zip code) HQ ESC/ENS 5 Eglin Street Hanscom AFB, MA 01731-2116
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI Joint Program Office 8b. OFFICE SYMBOL (if applicable) ESC/ENS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (city, state, and zip code)) Carnegie Mellon University Pittsburgh PA 15213	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO 63756E; PROJECT NO. N/A; TASK NO N/A; WORK UNIT NO. N/A
11. TITLE (Include Security Classification) Structural Modeling: An Application Framework and Development Process for Flight Simulators	
12. PERSONAL AUTHOR(S) Gregory D. Abowd, Len Bass, Larry Howard, and Linda Northrop	
13a. TYPE OF REPORT Final 13b. TIME COVERED FROM TO	14. DATE OF REPORT (year, month, day) August 1993 15. PAGE COUNT 30 pp.
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)

coordination models
design specification and evaluation
object -oriented application frameworks
partitioning
software quality attributes
structural modeling

19. ABSTRACT (continue on reverse if necessary and identify by block number)

In this paper, we present the structural modeling approach, an application framework and development process for the construction of flight simulators. Structural modeling was developed to address functional, nonfunctional, and process requirements for flight simulators. It has been successfully use in the development of large scale (one million lines of Ada code) flight simulators for the United States Air Force. A structural model promotes a simple and coherent software architecture with a small number of specialized structural elements obeying a few systemwide coordination strategies. It is this simplicity coherence of the software architecture that enables analysis to demonstrate the quality of the system.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT ☐ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution
22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas R. Miller, Lt Col, USAF	22b. TELEPHONE NUMBER (include area code) (412) 268-7631 22c. OFFICE SYMBOL ESC/ENS (SEI)