Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

THE MODELING METHODOLOGY, MODEL SPECIFICATIONS AND DEVELOPMENT OF CASI: CASE/ARCHITECTURE SIMULATION INTEGRATION

Arnold J. Almanzor
Paul R. Work

Computer Sciences Corporation
Integrated Systems Division
304 West Route 38, Post Office Box N
Moorestown, New Jersey 08057, U.S.A.

ABSTRACT

This paper discusses the CASE/Architecture Simulation Integration (CASI, pronounced "CASEY"), methodology for integrating performance engineering architectural modeling information with CASE-based systems requirements analysis and design. The CASI approach is to embed data structures within a CASE tool so that the output of the CASE tool can serve as an input to an architectural modeling tool. Portions of a sample model, developed to show how the CASI data structures are embedded within the CASE based model, are presented. Also discussed are potential future directions where the CASI methodology may evolve.

Keywords: Modeling Methodology, CASE, Architecture Simulation, Performance Engineering

1 INTRODUCTION

The purpose of CASE/Architecture Simulation Integration (CASI) is to develop a common model between system software requirements and architectural performance simulation. Computer-Aided Software Engineering (CASE) tools provide systems designers and engineers with an easier way to model systems. Systems designers model the functionalities of a system and use CASE tools such as Cadre Technologies' teamwork to create data flow diagrams. Systems engineers, on the other hand, model the architecture of a system and use CASE tools such as CACI's NETWORK II.5® to run performance simulations. However, modeling the same system in different CASE tools can lead to various analytical and design problems. The major concern is that there is not a unified concept of the system. According to Hatley and Pirbhai (1987), "the requirements and architecture models complement each other well. Processes in the requirements model can be allocated to slots in the architecture model..." The CASI methodology will serve as an approach to integrate CASE tool technology and help bridge the communication gap of modeling a system between systems designers and engineers.

1.1 CASI Example Model

The following material was developed as an example model of a system so that the architectural information necessary for simulation could be shown. The example model uses the CASE/Architecture Simulation Integration (CASI) methods for embedding architectural performance characteristics, requirements, constraints, capacities, and budgets. This is done so that the performance engineering tasks necessary to ensure a reasonable design are incorporated while the system is being analyzed and synthesized and not put off to some modeling group distinct from the engineering and design team or just put off until "another time" when it's "more convenient". According to Jain (1991), " Performance evaluation is required at every stage in the life cycle of a computer system..." We are extending this concept to state the importance of analyzing the system architecture as soon as possible in the early stages of system design.

1.2 Problem Statement

A hypothetical aerospace company needs an Aircraft Monitoring System (AMS) to be placed in one of their prototype aircraft. The purpose of the Aircraft Monitoring System is to monitor and poll different sensors on the aircraft. These sensors should measure the engine's pressure and temperature, fuel tank level and smoke detection. The system should also include a timer, a clock and a CRT to display the status of each of the sensors. The AMS we used is based on an example that can be found in Cadre Technologies' teamwork/SA

and teamwork/RT User's Guide, Release 4.0, (1990).

2 REQUIREMENTS ANALYSIS

The analysis of this system will be based on Hatley and Pirbhai's structured analysis methodology. The central component of the system shall be called Monitor_ Aircraft. The main function of Monitor_ Aircraft will be to poll the sensors to determine their status. The sensors will be external to Monitor_Aircraft. The sensors will notify Monitor_Aircraft of their status and whether or not a problem has occurred. The system should monitor these sensors about once a second. The external sensors shall be called the Engine_temperature_sensor, Engine_pressure_sensor, Smoke_detector and Fuel_tank. The other components shall be called the Timer, CRT and Clock.

Monitor_Aircraft will request messages from the Engine_pressure_sensor, Engine_temperature_sensor and Fuel_tank to gather information of their status. These sensors, in turn, will then send their data information to Monitor_Aircraft. The Smoke_detector will send a message to Monitor_Aircraft of its status. Since Monitor_Aircraft needs to monitor the sensors about once a second, the Clock will send an interrupt once a second to Monitor_Aircraft. Monitor_Aircraft then requests data from the sensors and sets a Timer to a one-second interval. During this one-second interval, data from the sensors will be accepted for this period. Once Monitor_Aircraft has received its information after polling its sensors then it will display the data on a CRT.

3 CREATING THE AMS teamwork MODEL

Upon completion of the requirements analysis, our next step was to put the requirements in a computer-based model. The CASE tool used was Cadre Technologies' teamwork. The following are portions of the AMS teamwork model. The Context-Diagram illustrates the overview of the AMS (refer to Figure 1).

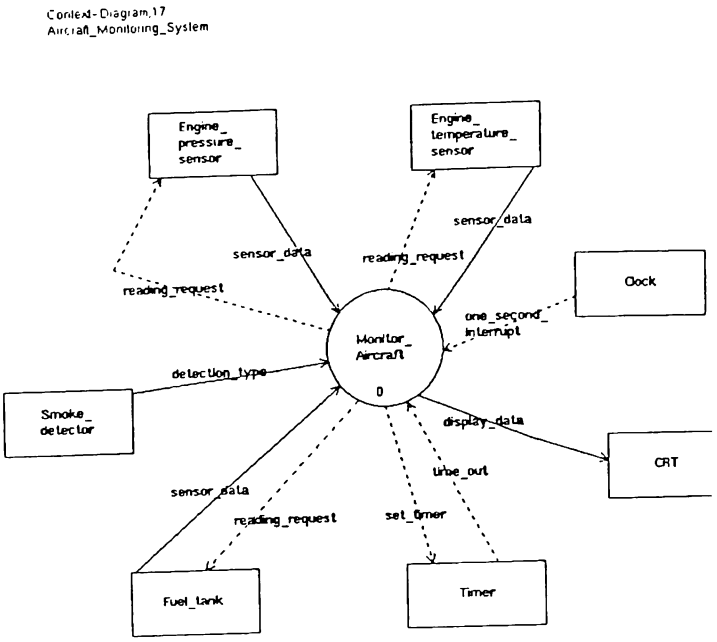

Figure 1: The Context-Diagram of the AMS

Monitor_Aircraft can be further abstracted into two internal subsystems which are Monitor_Sensor and Generate_Alarm. The Monitor_Sensor subsystem can be further decomposed into six processes (refer to Figure 2). The Receive_Sensor_Data process can receive sensor_ data and read the time_of_day from the read-only storage area. The Receive_Sensor_Data process can send out several events. The sensor_data_received event is an input to the Monitor_Sensor State Event Matrix (Monitor_Sensor_SEM). This matrix can also receive a time_out from the Timer_Handler process and an one_second_interrupt from the One_Second_Interrupt Handler. The events that can be output from the Monitor_Sensor_SEM are sensor_data_received, set_ timer, reading_request and record_timeout. The sensor_data_received, fuel_data_received and record_ timeout events are inputs to the Monitor_Sensor Process Activation Table (Monitor_Sensor_PAT). The Receive_ Sensor_Data process can write to a storage area called sensor_reading. The Determine_Fuel_Capacity process can receive from the sensor_reading data store and output an event called fuel_status. The Determine_Range process can read from the sensor_reading and range_ constants data stores and output a sensor_status event. The Record_Timeout process can read the time_of_day and range_constants storage areas and output to the sensor_reading storage area.

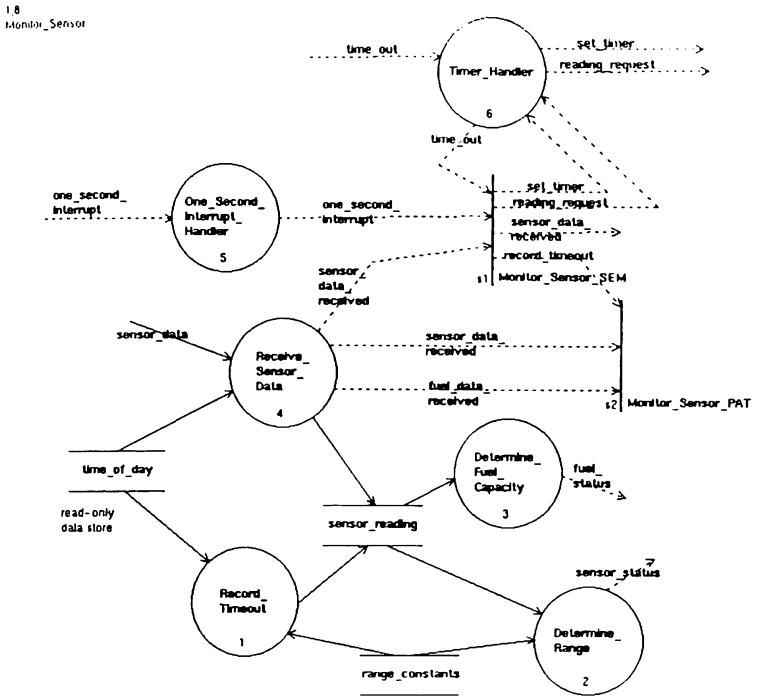

Figure 2: The Monitor_Sensor Subsystem

The Monitor_Sensor_PAT (refer to Figure 3) shows control actions (left of the double vertical broken line) and processes (right of the double vertical broken line). The Monitor_Sensor_PAT status indicates whether or not to enable particular processes. If the control action fuel_data_received is sent from the Receive_Sensor_Data process and it is true then execute the Determine_Fuel_ Capacity process. If the control action sensor_ data_ received is sent from the Receive_Sensor_Data process and it is true then execute the Determine_Range process. If the control action record_timeout is sent from the Monitor_Sensor_SEM and it is true then execute the Record_Timeout process followed by the Determine_ Range process.

fuel_data_received	sensor_data_received	record_timeout	"Record Timeout"	"Determine Range"	"Determine Fuel Capacity"
	"TRUE"			1	
"TRUE"					1
		"TRUE"	1	2	

Figure 3: The Monitor_Sensor Process Activation Table

4 DEFINING THE SYSTEMS ARCHITECTURE FOR THE AMS

Now that the software requirements have been modeled, we considered the architecture of the system which would include the hardware, software and performance simulation characteristics (refer to Figure 4).

The Aircraft Monitoring System will have two main processors called the Monitoring_Subsystem and the Alarm_Display_Subsystem. The eight storage devices will be the Alarm_Display_Crt, Engine_Pressure_ Sensor, Engine_Temperature_Sensor, Smoke_Detector, Fuel_Tank, Memory, Timer and Clock. The two transfer devices will be the Crt_Interface and the Vme_Bus. The two LANs will be the Dedic_Smoke_ Alrm_Bus and the Main_ Avionics_Data_Distribution_ Bus.

The Monitoring_Subsystem will be connected to the Vme_Bus and the Main_Avionics_Data_Distribution_ Bus. The software modules that will reside on this processor are Record_Timeout, Determine_Range, Determine_Fuel_Capacity, Store_Sensor_Data, Process_ Sensor_Data, Read_Sensor_Data, One_Second_ Interrupt_Handler, Setup_Timer and Process_ Timeout.

The Alarm_Display_Subsystem will be connected to the Crt_Interface, Vme_Bus, and Dedic_Smoke_Alrm_Bus. The software modules that will reside on this processor are Display_Fuel_Status, Display_Sensor_Status, Process_Smoke_Detector_Alert and Smoke_Detector_ Interrupt_Handler.

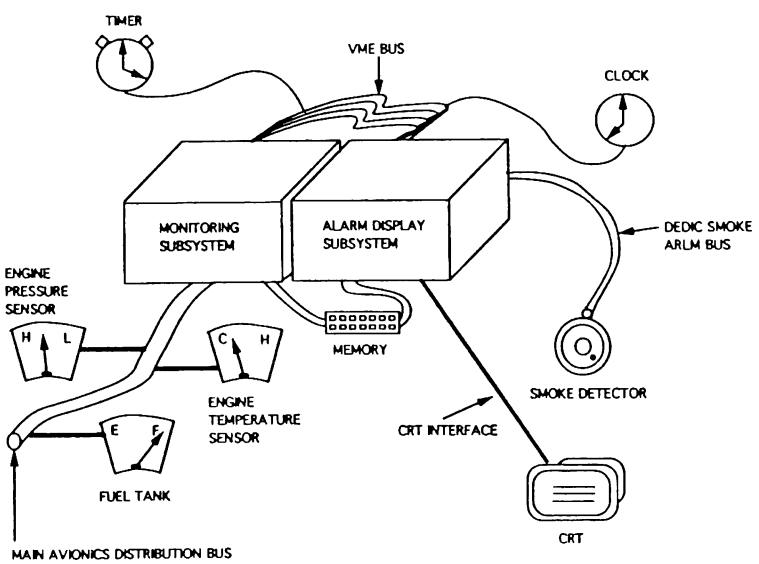

Figure 4: The Systems Architecture of the AMS

5 EMBEDDING CASI STRUCTURES IN CADRE'S teamwork MODEL

The three primary areas in which the user embeds CASI structures in Cadre's team*work* model are in the Notes attached to the Context-Diagram (refer to Figure 5), the P-Spec's Notes attached to the Process Bubbles (refer to Figure 6), and the Data Dictionary Entries (DDEs) (refer to Figure 7). The CASI structure will only be recognized if the location is correct and if the CASI structure is surrounded by the characters ("*~" asterisk-tilde and "~* " tilde-asterisk).

The following figures are examples that show the embedded CASI structures within the team*work* model for the AMS.

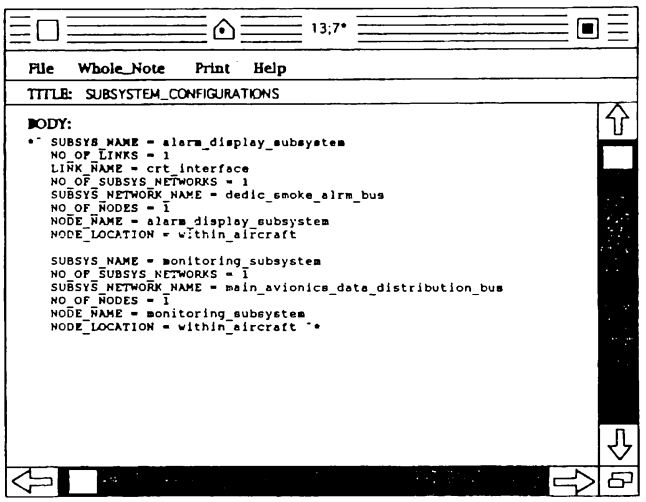

Figure 5: CASI Structures Attached to the Context-Diagram Notes

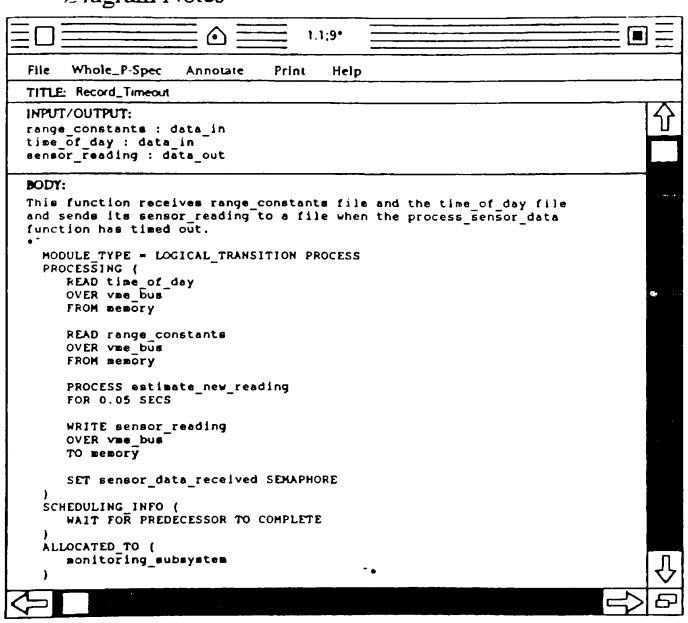

Figure 6: CASI Structures Attached to the P-Spec Notes

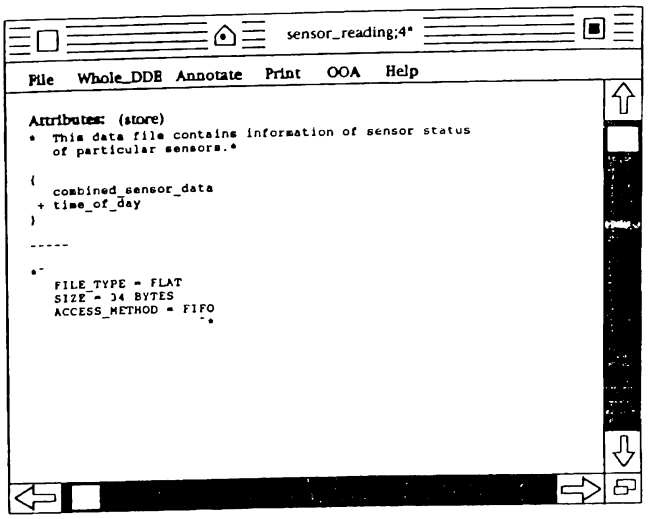

Figure 7: CASI Structures Attached to the Data Dictionary Entries

6 TRANSLATION OF CASI STRUCTURES INTO ARCHITECTURE SIMULATION MODEL

The CASI Implementation Design Approach diagram (refer to Figure 8) illustrates the process of extracting the CASI structures from Cadre's team*work* in which a CASI extraction/translation program translates CASI structures to NETWORK II.5® structures (1992). The CASI extraction/translation program will be generic so that other architectural simulation models can be substituted such as SIMAN and SES/Workbench. In this case, the resultant file will be the NETWORK II.5® .net file. This file will be later imported into NETWORK II.5® so that the performance simulation can be executed and reports generated.

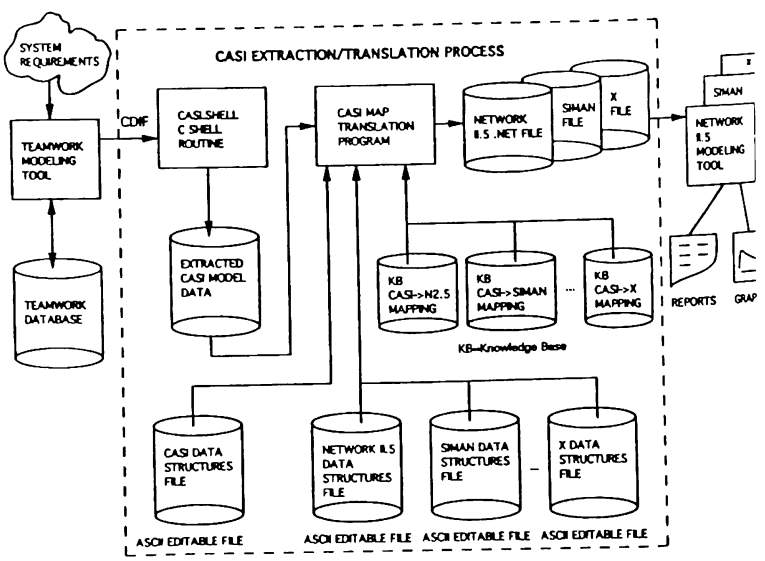

Figure 8: CASI Implementation Design Approach

7 FUTURE DIRECTIONS FOR CASI

There are two major directions that can be taken to evolve the CASI methodology. The first is to address object-oriented CASE-based modeling such as described by Zeigler (1990), and the second is to apply artificial intelligence techniques.

Since the architectural elements and processing specifications within the CASI methodology are to a certain extent already object-oriented, the transition to object-oriented CASE-based modeling should be somewhat straightforward. The processing specifications currently deal with the operations to be performed by a single processing node while the architectural elements are structured on the characteristics of the objects in the system architecture, such as processors, LANs, and files. Therefore, transitioning from the functional analysis methodology of the Cadre's team*work* SA/RT world to an OOA/OOD world should be achievable.

In the area of artificial intelligence, expert system technology could be visited. Here the conventions and rules of thumb used by system architectures for certain parameters of the architectural elements could be supplied by an expert system knowledge-base, e.g., basic assumptions for protocol definitions. In addition, consistency rules could be added to ensure that when certain bounds or that other required elements are also specified for consistency and completeness of the model such as suggested by Fishwick (1992). As another alternative by Aronson (1991) could be to combine both OO and AI techniques to take advantage of the strengths of both.

8 CONCLUSION

It has been shown that the CASI model specifications and development methodology can be applied to a simple real-world problem. The CASI approach has demonstrated that it can embed performance simulation characteristics within a functional CASE based model using tools such as Cadre Technologies' team*work*. According to Connie Smith (1990), "we can expect future CASE tools to incorporate performance prediction and assessment features." In CASI's model specifications and development methodology, we have shown that this can be done now.

REFERENCES

Aronson, Jesse S. 1991. *AES: An Object-Oriented,Knowledge-Based Approach to Simulation.* VA: 375-379.

Fishwick, Paul A. and Bernard P. Zeigler. 1992. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation.*. 2:1:52-81.

Hatley, Derek J. and Imtiaz A. Pirbhai. 1987. *Strategies for Real-Time System Specification.* New York: Dorset House Publishing.

Jain, Raj. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* New York: John Wiley & Sons, Inc.

NETWORK II.5® Data Structures. 1992. California: CACI Products Company.

Smith, Connie U. 1990. *Performance Engineering of Software Systems (The SEI Series in Software Engineering).* New York: Addison-Wesley Publishing Company.

teamwork/SA® teamwork/RT® User's Guide Release 4.0. 1990. Oregon: Cadre Technologies Inc.

Zeigler, Bernard P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems.* San Diego: Academic Press, Inc.

AUTHOR BIOGRAPHIES

Arnold J. Almanzor is a systems analyst and programmer at the Integrated Systems Division of Computer Sciences Corporation, Moorestown, NJ. Mr. Almanzor holds a BS in Information Systems from Drexel University and is currently pursuing a MS in Information Systems at Drexel University. His research interests include computer-based tools for modeling and assessing the performance of system architectures as well as human computer interface design. He is a member of the IEEE Computer Society and ACM.

Paul R. Work is a Deputy Director of Systems Engineering with the Computer Sciences Corporation, Moorestown, NJ. Mr. Work holds a BS in Computer Science/Mathematics from Roger Williams University; and closing in on a MS in Engineering Management from Drexel University; has over 20 years of software and systems engineering experience in the commercial and military sectors; and is responsible for providing performance and systems engineering services. He is a member of the Society for Computer Simulation, IEEE Computer Society, and ASEM.