Prototype and Test Environment for ATCS Data Communications

DOROTHY A. COLBURN

Advanced Train Control Systems (ATCS) are distributed command, control, and communications systems that also provide an infrastructure usable for train control and numerous ancillary applications. A crucial stage in ATCS development is the verification of the logic that allows applications to access this infrastructure. The communications protocols used to connect applications in different physical components are described in ATCS Specification 200, which represents the protocols as finite state machines using CCITT Recommendation Z.101 notation. These state machine representations are converted into a statechart representation in a computer-aided software engineering (CASE) tool. This tool simulates the protocol operations and generates an ADA prototype of the protocols. The communications between applications within a physical ATCS component are not standardized. For prototyping purposes, however, these communications have been modeled as a software bus. The software bus allows applications to communicate with the prototype of the protocol stack, providing an integrated environment in which the interactions of applications, the protocols, and the bus may all be examined and monitored. The prototype code generated by the CASE tool is supplemented by ADA code developed to simulate events that would occur within the interactions of the communications protocol. The interdependency of the software bus, the CASE-generated ADA code, and the programmer-developed ADA code provides an environment for meaningful prototyping and testing of the access logic, as well as the communications protocols. Instances of this prototype may be connected through a network emulator for use in further testing and prototyping the ATCS communications architecture.

Advanced Train Control Systems (ATCS) are a joint U.S.-Canadian endeavor focused on increasing railroad efficiency and service performance. ATCS computers will automatically monitor and control train traffic, allowing railroads to improve service while reducing costs through more precise management of resources (i.e., people, capital equipment, and fuel). During the last decade, the ATCS program has evolved from the determination of requirements through the identification of required components and the writing of performance specifications, to the development of procedures for testing components and complete systems.

The ATCS data communications system structure is based on the International Standards Organization's Open Systems Interconnect (OSI) Reference Model (*1*). This seven-layer protocol model was selected to define the points of interconnection between systems. Figure 1 shows the relationship between the OSI model and the ATCS communications architecture for communications between a dispatch center and a locomotive computer.

ARINC Research Corporation, 2551 Riva Road, Annapolis, Md. 21401.

The ATCS seven-layer architecture can be divided into two subsystems: the lower layer protocols (LLP) and the upper layer protocols (ULP). The LLP provide for the switching and routing of information, media access control, synchronization, framing, error detection and recovery, and the actual movement of the bits across a transmission medium. The ULP provide data formatting and identification, interaction between applications, and end-to-end data integrity and quality of service.

TESTING THE ATCS DATA COMMUNICATIONS SYSTEM

The communications components of ATCS provide functionality within the lower three layers of the data communications system protocol. These are the mobile communications package, base communications package, front-end processor, and cluster controller. All seven layers of the protocol are implemented within the end-user computer system.

To test the functionalities of the lower layer communications components, a communications interoperability tester (CIT) was developed. This test system, written in the ADA programming language, allows for a stimulus-response, script-based–type testing of the four ATCS communications components in a simulation environment.

According to ATCS Specification 210, "A central objective of ATCS is to ensure safe train movements and track occupancies which avoid physical conflict, and other operational hazards of similar magnitude." To ensure that the implemented ATCS will meet this objective, development of a simulation model of an ATCS is imperative. As a subset of simulating an entire ATCS, a working prototype of the communications protocols is being developed.

SELECTION OF A PROTOTYPE MODEL

Some of the advantages of building a prototype are discovering design problems early, spotting system interface problems from the start, and using the prototype to experiment with how proposed enhancements and modifications will affect the system (*2*).

Fundamental to the success of a prototype is the ability to develop a good model of the system. As Hoover and Perry have pointed out, "Models are not true or false, but rather they are useful and appropriate for the analysis at hand" (*3*). The type of model required for the simulation of the ULP must have the following characteristics: descriptive, discrete,

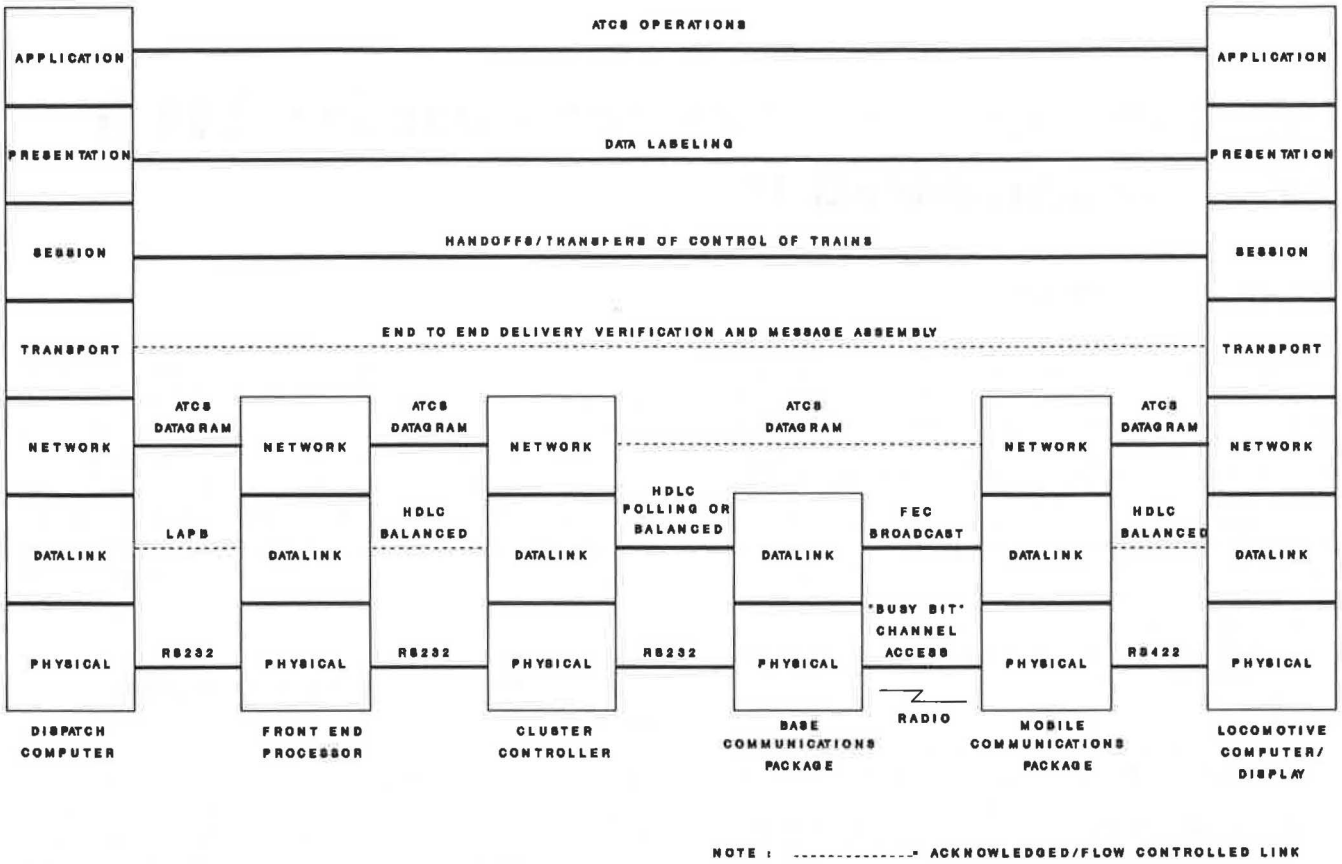

FIGURE 1 ATCS data network by OSI layer.

probabilistic, dynamic, and open-loop. The purpose of this prototype is to describe the system behavior rather than to optimize the system; therefore, a descriptive model of the system is required. Data packets, and messages, arrive at discrete points in time; hence, the system model must be discrete. The arrival times of these packets and messages cannot be predicted, requiring that the model be probabilistic. All communications networks are dynamic, with the number of messages/packets in queue at any given time being variable. Finally, a communications network is an open loop; the system output is not fed back into the network to modify subsequent outputs, but is consumed by the environment with which it interacts.

With the required model classified as a discrete event model, and the objective to simulate the model, the next step is to select a technique for developing and simulating the system model. The major tasks in model construction have been defined as (a) develop a computer program flow chart for the model; (b) select a programming language; (c) provide for generation of random numbers and collection of performance measure values in the program; and (d) write and debug the program code (*3*).

Two approaches for developing model charts of the system were identified: the physical flow and the state transition flow. The physical flow approach identifies and diagrams the physical entities of the system. For the communications network, this would be the receive and transmit queues, processing queues, and any internal queues between layers of the protocol (see Figure 2). The state transition approach uses state transition diagrams (*4*) to represent a system. This is an event-driven approach that uses state variables to describe the system (see Figure 3). Both of these approaches are helpful in understanding and diagramming the intricacies of a system.

The ATCS communications protocols are described in ATCS Specification 200 as finite state machines (FSMs) (see Figure 4) using CCITT Recommendation Z.101 notation (*5*).

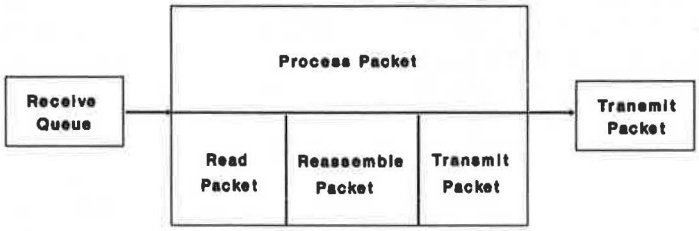

FIGURE 2 Physical flow chart example.

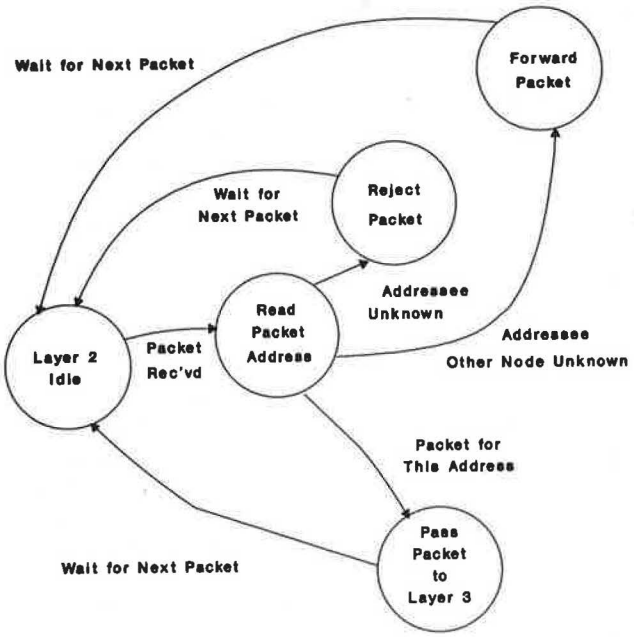

FIGURE 3 State transition diagram example.

Statecharts, extensions to FSMs, have been developed and implemented in STATEMATE, a set of tools intended for the specification, analysis, and design of reactive systems. STATEMATE was selected for the development of the prototype model for the simulation of the ULP.

STATEMATE allows for the development of the model from three separate, but related, points of view: structural, functional, and behavioral (*6*). Figure 5 illustrates how these views are linked together to create a logical model of the system. The structural view of the model is decomposed into modules. These modules may describe the actual hardware components of the system, or they may define the software components of a system: the subroutines, packages, or tasks. This view of the system is modeled using module-charts (see Figure 6).

The conceptual view of the model is decomposed into its functions and controls. The functional view of the system is described using activity charts. The system's functions are described as activities (see Figure 7). An activity typically accepts inputs and produces outputs. Activities that reside at the lowest level of the system may also be described as code in a high-level programming language. Activity charts may also contain data stores and control activities. Data stores portray a temporary or permanent storage area for data. Con-

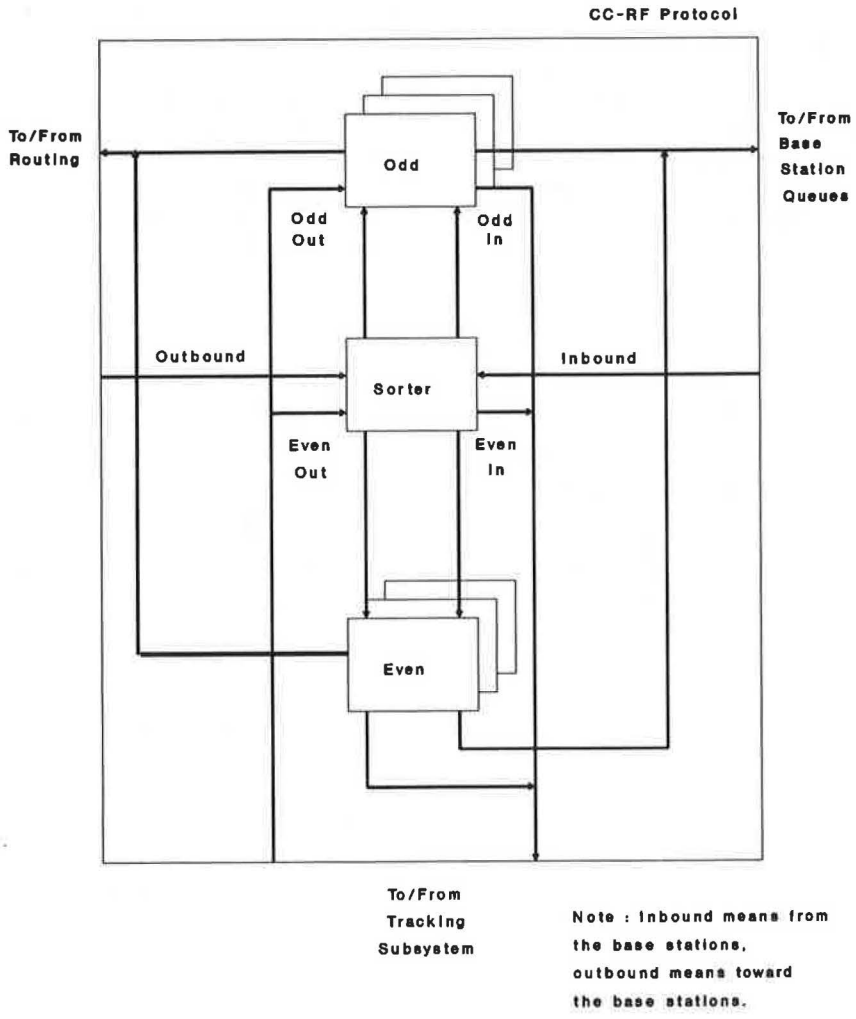

FIGURE 4 Specification 200 finite state machine example.

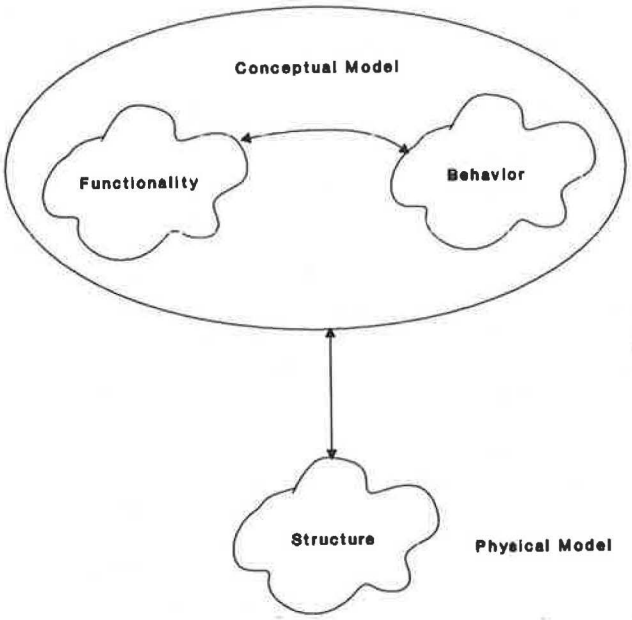

FIGURE 5 Structure of a STATEMATE model.

trol activities, the behavioral view of the system, are described as statecharts (see Figure 8).

The STATEMATE simulation tool can be used to simulate the dynamic behavior of the system at any stage during and after its development. The simulation tool may also be used to debug and verify that the system works as expected. It can also be used to demonstrate the behavior of the system or to run it through various scenarios and observe the system's interactions.

PROGRAMMING THE MODEL

After a model has been developed and analyzed, STATEMATE can then translate the developed system into code in one of two available high-level programming languages, ADA or C. This prototype code represents the executable conceptual model of the system. This code may be compiled and linked as is, or it may be enhanced to call other code developed by programmers.

For implementing the ULP, ADA was selected as the programming language for code development. One of the main reasons for the selection of ADA is that the CIT, which was built to test the lower three layers of the protocol, was developed in ADA. Should the ULP model and any of the CIT software need to interface, the process would be expedited because they were both developed in the same language.

Although there are arguments that the overhead associated with the use of ADA in OSI-style communications systems is significant (*7*), the intent is not to develop an efficient model of the system as much as it is to develop an *accurate* model of the system. Therefore, any overhead introduced by the language would be moot.

There are also arguments that code generated by CASE tools is cumbersome, poorly structured, and inefficient. The STATEMATE developers admit that the code may be inefficient (*7*). However, the advantages of having the CASE tool generate the code outweigh the disadvantages. Having the system generate the code that can then be compiled and linked to form the executable program ensures that the final program accurately and precisely correlates to the diagrammed model. The only potential areas of deviation from the communications specification are (a) incorrectly entered diagrams or (b) incorrect implementation of the manually produced primitive event code.

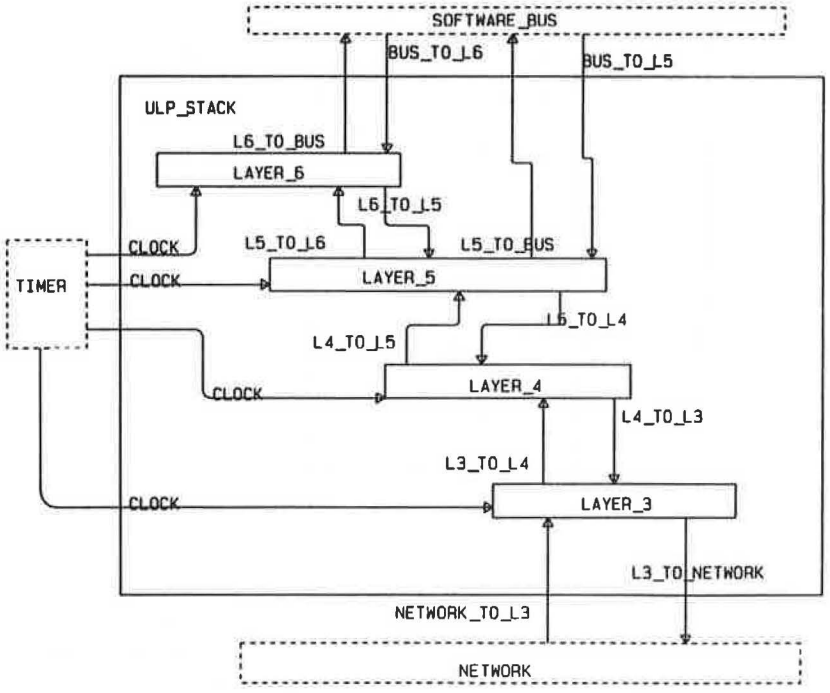

FIGURE 6 ULP module chart example.

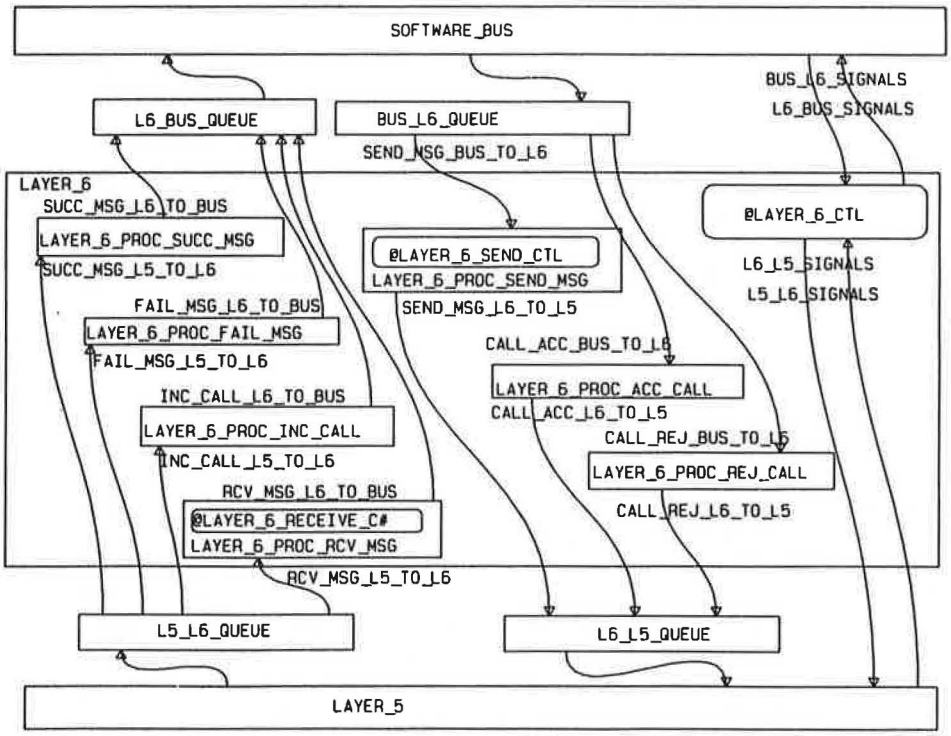

FIGURE 7 ULP activity chart example.

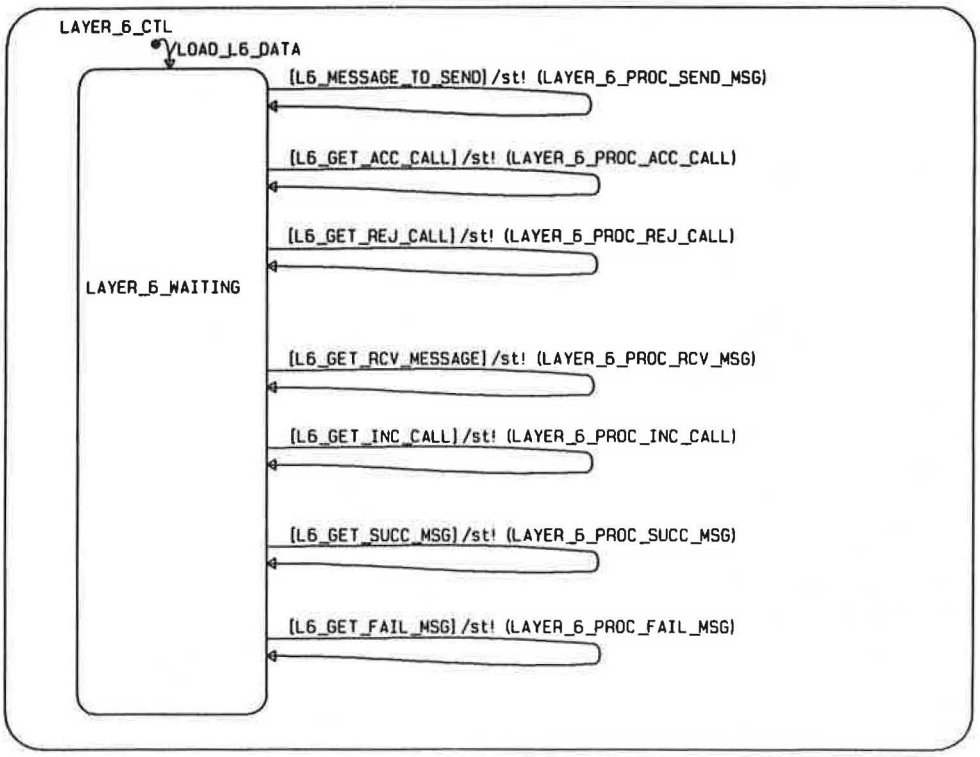

FIGURE 8 ULP statechart example.

Using the code-generation tool also reduces the number of hours required to develop the simulation model. For example, the STATEMATE-generated code handles the ADA tasking development and synchronization, a function that can take a programmer weeks to refine.

Because the purpose of this project is to develop a fully functional model of the system, the prototype code produced by STATEMATE is edited to call modules written by the programmer—referred to as primitive events. These primitive events are called to perform such things as storing a message header in a queue or retrieving a message header from a queue.

MODEL IMPLEMENTATION

At the current writing, Layers 5 and 6 of the ULP have been diagrammed in STATEMATE, verified using the testing tools of STATEMATE, and the ADA code generated for these two layers. The code has been edited to call the programmer-developed primitive events. After being compiled and linked together, the code was debugged by stepping through the logic of the model.

Many uses of the final product of this model have been suggested. One plan calls for further development of a limited version of the entire ATCS. This would require a simulation model that could simulate ATCS behavior in the command and control of trains. To pass messages from the ULP model to the planned models of ATCS components, a "software bus" has been developed. This software will pass simulated ATCS communications between the ULP model and the component models during simulation of the system.

Plans have also been made to use the model to test a ULP stack developed by other organizations. In this manner, the peer-to-peer relationships of the ULP under test could be verified.

MODEL VERIFICATION AND VALIDATION

The ULP model is being developed using modularity and stepwise refinement. The first segment of the model developed was Layer 6 of the protocol. This module was then interfaced with the software bus and a stub of Layer 5, debugged, and tested. Layer 5 was then developed, debugged, and tested in the same manner. Layers 3 and 4 will follow this same development method.

At completion of the entire model, the model will be validated. Verification and validation of a model is at best an inexact science, yet the credibility of the model must be established. Many formalized methods for model validation have been developed, such as the Delphi Method, the Turing Test, and the Structured Walk-Through. Although all of these methods have something to offer, and the exercise of any of them will, no doubt, turn up errors in the model's logic, each has its own drawbacks.

Plans are to test the ULP model by first stepping through it with the debugging tools available in the VAX implementation of ADA and the debugging tool in STATEMATE. This process will validate the layer-to-layer services of the model. The next test will be to perform communications between two executing copies of the model to validate peer-to-peer functionalities of the model. A third validation of the model will be its execution with the software bus at the upper end of the ULP stack and an ATCS network emulator at the lower end of the ULP stack. This process will validate the end-to-end functionality of the protocol model between the user application and the network emulator.

At each of these validation stages, the model should be tested not only to ensure that it functions properly under normal conditions but also to observe its behavior under extremes.

CONCLUSION

Prototyping is based on building a model of the system under development and then simulating the system's behavior using the model. The use of STATEMATE to visually formalize and then generate code to develop the ULP simulation model takes advantage of the benefits of prototyping. The use of ADA as the programming language exploits the tasking capabilities of the language and provides for future interfaces to the CIT. This approach maximizes the capabilities of the available technologies and the computer system. The result should be the development of a model that accurately represents the real system, its functionalities, and its behavior.

ACKNOWLEDGMENT

The author would like to thank Robert Ayers, Jack Bailey, and Denny Lengyel of ARINC Research Corporation, without whom this project never would have begun. Robert Ayers deserves special thanks for his technical support, assurances, and constructive criticism during the development of the model thus far and for his design and development of the software bus. Thanks also to Jennifer Miller for keeping the development computer system and software up and running during this project.

REFERENCES

1. The International Telegraph and Telephone Consultative Committee. *Data Communication Networks Open Systems Interconnection (OSI) System Description Techniques, Recommendations X.200–X.250*, Vol. VIII, 1985.
2. K. E. Lantz. *The Prototyping Methodology*. Prentice Hall, Englewood Cliffs, N.J.
3. S. V. Hoover and R. F. Perry. *Simulation: A Problem-Solving Approach*. Addison-Wesley, New York, 1989.
4. A. M. Davis. A Comparison of Techniques for the Specification of External System Behavior. *Communications of the ACM*, Vol. 31, No. 9, Sept. 1988, pp. 1,098–1,115.
5. The International Telegraph and Telephone Consultative Committee. *Functional Specification and Description Language (SDL), Recommendations Z.100–Z.104*. Vol VI, Fascicle VI.10, 1984.
6. D. Harel, H. Lachover. A. Naamad, A. Pneuli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, Vol. 16, No. 4, April 1990, pp. 403–413.
7. N. R. Howes and A. C. Weaver. Measurements of ADA Overhead in OSI-Style Communications Systems. *IEEE Transactions of Software Engineering*, Vol. 15, No. 12, Dec. 1989, pp. 1,507–1,517.