Requirement Specifications For Real-Time Communication*

Sean David Sandys Alan Shaw

Department of Computer Science & Engineering

University of Washington

Box 352350

Seattle, WA 98195-2350

{sds,shaw}@cs.washington.edu

December 14, 1998

Abstract

Distributed communications are an essential part of many current and proposed real-time systems. However, no existing requirements specification language provides explicit and general support for distributed real-time communications (DRTC), most likely because of its diversity and complexity. Through examples, we first justify the need for handling DRTC at the specification level; an air traffic control application is discussed in some detail. A useful model for DRTC is then presented, emphasizing real-time requirements and differences between DRTC and conventional communications; these differences include time and message constraints, as well as failure and reliability issues. We then describe our approach to specifying the many kinds of send and receive actions, communications channels, and message types within a real-time framework. The specification scheme is currently under development and is an extension of a state machine language that has been used successfully for some large applications.

1 Introduction

When expressing requirements on the behavior of real-time systems, it is not enough to define the behavior of individual components in isolation; it is also important to describe the aggregate behavior of the system. This includes not only the control and process behavior of the system, but also the commu-

*The work was supported in part by the Army Research Office under grant number DAAH04-94-G-0226.

nication and interaction between components. In particular, *Distributed Real-Time Communications* (DRTC) has become an essential element in current and proposed systems.

There has been much work in requirements specification languages for real-time systems. Most of these languages provide abstractions that are primarily process or component centered even if they handle parallelism and offer a basic communication facility. However, not much work has been done in developing general methods for treating DRTC which, because of its diversity and complexity, poses many challenging, interesting, and difficult problems. Our contributions are to present the issues in detail, provide a useful model for DRTC, and define a research framework for DRTC specifications, focusing on state-based methods.

The rest of the paper is organized as follows. Section 2 argues for the importance and variety of distributed real-time communication by examples. Section 3 presents a model for DRTC. Section 4 surveys current specification methods, examining how these methods support DRTC. Some of our work on and ideas for achieving a general specification mechanism for DRTC is discussed in Section 5.

2 Examples of Communications in Real-Time Systems

There exist a wide variety of complex computer controlled and monitored systems. Examples of larger systems are air traffic management (ATM), highway traffic control, military command and control, large medical systems such as radiation therapy equipment, and power generation. Some smaller — although still complex — systems include medical devices like defibrillators, flight management, automobile control, and building monitoring and control.

Although these systems range from large loosely coupled multi-component systems to smaller tightly coupled systems with fewer components, they do have a variety of features in common. They are all distributed and they exhibit a diverse communication style, not only across systems but sometimes even within a single system. These applications all have real-time constraints on their behavior and the constraints also exist on communication. Finally, there is a growing number of these systems, and they are becoming ubiquitous.

2.1 A Comprehensive Example: Air Traffic Management Systems

We present an example of an air traffic management system (ATM) to illustrate the variety of communication styles present in real-time systems. ATM systems were selected as an example for a number of reasons. First, they are an important, widely known, and difficult application [15]. Second, they also contain examples of virtually every known communication styles. Finally, it is an application with which we have some knowledge and experience, based on a recent safety study supported by NASA [10] and several years of using these systems in research and educational exercises.

Overview of ATMs. Although there are many air traffic management systems, the ATMs that are outlined here are those that monitor aircraft flying in United States airspace. Figure 1 shows the basic control points in the U.S. air traffic control environment. U.S. airspace is divided into volumes called *sectors*. If a plane is flying in a controlled airspace, there is a single air traffic controller responsible for that aircraft. As aircraft pass from one sector to another, control of that aircraft passes from one controller to another. The sectors near airports are called *Terminal Radar Approach Control (TRACON)* sectors; here, the air traffic controller directs traffic to the tower for landing and away from the tower to En Route sectors after takeoff.

The main goals of these systems are safety, efficiency, and performance. We want to avoid collisions and other hazards. In order to accomplish this goal, air traffic controllers need to maintain aircraft separation, as well as avoid weather hazards, natural obstacles and restricted airspace. With respect to efficiency and performance, air traffic controllers would like to maximize airspace capacity and airport throughput, and minimize aircraft delays, while minimizing fuel consumption.

Distributed Communications in ATMs. At each individual center, there may be a conventional distributed system with workstations and servers, running a set of different processes that exchange information. There are also many forms of communication between centers. For instance, when an aircraft passes from one sector to another, control of the aircraft is passed from one air traffic controller to another. This hand-off procedure, whether automated or manual, requires one-to-one, synchronous communication with strict time deadlines. Proposed changes in ATM systems include the ability for controllers to send flight plans directly from ground ATM systems to flight management systems on-

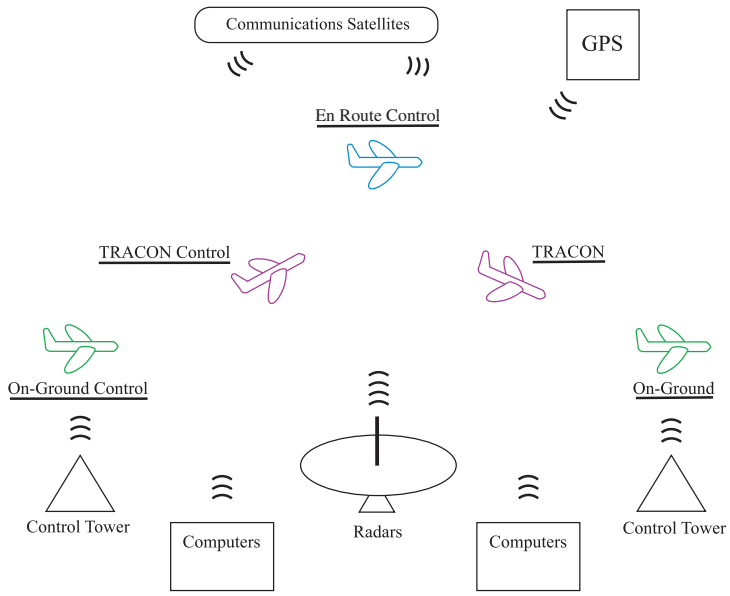

Figure 1: A simple view of the various components and control points in modern air traffic control

board aircraft. Radar will be augmented by aircraft that can report their position — determined through broadcast communication with global positioning systems – directly to ground control. Another interesting communications aspect of the system is that many of the components in the system are mobile.

As illustrated by the ATM example, real-time systems can exhibit a wide variety of communication styles and many of the communication instances have real-time constraints. In our view, a specification language should provide abstractions that support the description of all forms of DRTC.

3 Communications Model

We present a model of real-time communication that is useful for expressing *requirements*. Our concern is not necessarily with particular protocols or individual physical interconnects between components; it is with the requirements on how various components exchange information.

For these purposes, several elements of communicating components are identified (Figure 2). Each

communication involves a sending process P_s, and one or more receiving process(es) P_r, which determine or specify what information is exchanged. Each process also has an interface between the process and the channel, pcI_s and pcI_r, which acts as a layer between the communicant and the transmission medium. Finally, the channels C_s and C_r describe the mediums over which information is passed. C_s and C_r represent logical channels of communication. When implemented, these channels may be realized by the same physical channel, but they still could have different logical properties depending on the requirements imposed by the sender and receiver.

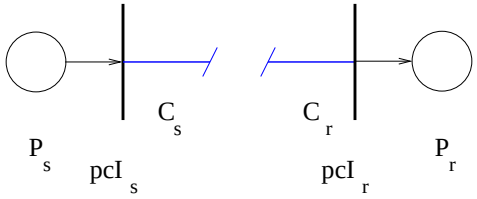

Figure 2: The Participants in inter-component communication

3.1 Characteristics of DRTC

There are a few distinctions between DRTC and conventional distributed communication. It is these distinctions that make the specification of DRTC a particularly difficult problem.

One main characteristic of DRTC is the existence of *timing constraints* and *timing properties*. The most basic constraints are timing restrictions on the end-to-end path:

$$P_s \rightarrow pcI_s \rightarrow C_s \rightarrow C_r \rightarrow pcI_r \rightarrow P_r$$

These end-to-end constraints are most often expressed as worst case deadlines on the time from P_s to P_r. It is also possible to have constraints on the minimum time allowed to transmit information along that path, for example, to prevent the overloading of the receiving component.

The timing properties of channels usually are expressed by bounds on the time for accepting and moving information across the channel, specifically, propagation delays and bandwidth. In the case of

process-channel interfaces and processes, we can look at the overhead that the interface or the process adds to communication. In many systems, we find processes that are mobile or can migrate. Here, the bounds on the channel are not solely a product of the medium but also a function of the location of the processes.

It is also useful sometimes to talk about message priority and quality of service. For example, missing a deadline may not be a failure as long as a certain percentage of our deadlines are met. In this case, a failure is not determined by a simple local decision but is instead based on the history of system behavior.

Failures and reliability issues are also integral to DRTC. From the perspective of a receiver, possible failure modes include:

1. received a message not intended for this process.
2. did not receive a message intended for this process.
3. received a message but data is incorrect/corrupt.
4. received a message but message violates some constraints (real-time message deadlines, wrong message size, ...)

A third distinctive component of DRTC is the nature of the *message contents*. Time-stamping of messages is often needed when writing requirements in order to specify behavior that is based on the age of the data being used as well as to allow system decisions about data validity. Time stamping may occur not only on a message send but also on a message receipt; given a single time stamp on receipt and constraints on maximum transmission time, it is possible to place conservative bounds on the time the message was sent. Placing bounds on message size may be necessary. It may be useful to have typed messages. Messages with no value, sometimes called signals, are also convenient.

It is convenient to distinguish between event and state messages. Event messages have semantics close to those of message passing systems, while state message have semantics similar to those of shared memory system with the exception that writes are atomic and there is a single sender [9].

3.2 A Brief Taxonomy of DRTC

The taxonomy that we use is similar to that given for conventional distributed communication [1]. The classification is divided into three categories: Properties of the sending side, the receiving side, and the communication channel. The properties of the sending and receiving side will include the properties of both their respective components (P_s, P_r) and interfaces (pcI_s, pcI_r).

We distinguish between blocking (synchronous with the corresponding receive) and non-blocking *sends*. A send can be either directed where the destination(s) of the send are known, or undirected where the receiver or destination is not known. Additionally, sends may be parameterized by the number of recipients of the message, i.e. whether the send is a unicast, multicast, or broadcast.

Examples: Synchronous sends include commands for immediate action where the sender needs assurance that the message was received; e.g., activating a brake or closing a value. These examples are also suited as directed sends to a single destination (unicast) . Non-blocking sends include some continuous service providers, e.g. a process sending reactor temperatures or a radar reporting aircraft positions. A process sending aircraft positions to any number of ground stations could use an undirected, broadcast mechanism to send this information.

The taxonomy for *receives* is slightly different to that for sends. They can be either blocking, non-blocking, or synchronous. Blocking implies that the receiver blocks until the receive gets a message. A non-blocking receive is one that returns immediately, either with the message (if it is available) or an indicator that no message was present. A synchronous receive is a receive that is matched with a blocking send.

Receives can also be directed or undirected. An undirected receive implies the destination component does not know from which component it expects a message, directed receive means the sender is known beforehand. Instances of real-time interfaces also include ports and mailboxes.

Examples: The real-time send examples have counterparts in real-time receives. A server may block for a new command or request for service (an undirected and blocking receive), or a process may poll periodically for information from a bank of sensors.

A *channel* can be viewed from both the perspective of the sender or the receiver. There can be unicast, multicast, or broadcast channels, as well as uni-directional or bi-directional. Probably the most interesting parameterizations of channels for DRTC their timing properties. This includes bounds on propagation delay and bandwidth.

3.3 Compatibility Constraints

The elements of the classification described in the previous sections are not completely independent. Although one must combine features to describe a communication system, it is not possible to combine features arbitrarily. For example, a synchronous send needs to be matched with a synchronous receive, and a non-blocking send is compatible only with either a non-blocking or a blocking receive. Another area where compatibility is an issue is channels. A channel's propagation delay needs to be compatible with the message deadlines imposed by the receiver. Some of these compatibility constraints, e.g. matching synchronous sends with synchronous receives), can be enforced, by a specification language, syntactically. Others, such as the channel compatibility requirements mentioned above, would need some analysis to verify.

4 A Short Survey of Current Methods

Our discussion will be restricted to state-machine and programming language-based requirements languages. In particular, we are not covering assertional logic methods, algebraic schemes, or Petri net based notations. First these systems are examined with respect to their general communication facilities; then DRTC support is covered.

4.1 Communications Mechanisms in Current Specification Schemes

Three classes of languages are presented: CSP-based, state machine, and other programming language methods. Communication in CSP is done over typed 1-1 channels; sends and receives are both synchronous. Most examples of CSP communication are used in a programming language framework to the describe the behavior of systems. Examples of languages with a CSP style of communication includes CSP [8], timed CSP [16], CRSMs [18], CSRs [5] and LOTOS [4].

Communication in most state-machine based languages occurs through the use of messages or events. Messages are mainly broadcast and often the communication is implicit. These schemes tend to use the same mechanism for exchanging intra-component information as they use for exchanging inter-component information. As such, it is not clear that communication in these systems can be distributed — the style is more akin to shared memory. Systems that fit in this category include: Statecharts [6, 7], RSML [11], SpecTRM-RL [12], and ROOM [17]. SpecTRM-RL does not use events as the basis for communication — it has state message semantics. However, its limitations are similar to those discussed above for other state-machine based systems.

Among these systems, ROOM is the only one offering more than one explicit communication abstraction. It supports both synchronous (blocking) and asynchronous (non-blocking) communication. In ROOM, communication is based on message passing. Messages can be typed as well as have priorities associated with them.

Our third category of schemes include ESTEREL [2], Spec [3], and PSDL [13]. Communication in ESTEREL is done through signals (messages with no values). All sends are explicit through an emit command. Esterel uses non-blocking, broadcast sends and blocking receives. The Spec and PSDL languages are designed for distributed systems. They support typed message passing with either single-buffered event messages or sampled state messages. Component objects, called operators, are described by state machines and can be defined with many timing and other constraints. Message declarations for both inputs and outputs appear in operator interface declarations, but the possible communication abstractions are unspecified.

4.2 Special DRTC Features

In Section 3.1, we described a set of features particular to DRTC. In this section, we discuss the various levels of support that different systems listed in the last section provide for DRTC.

Timing Facilities. Almost all the systems permit the expression of timing constraints, indirectly through the use of timeouts at either the send or receive end of the communication. CRSMs also have direct support for timing constraints on communication, it is possible to express both lower and upper bounds on state transitions. This makes it possible to specify more detailed bounds on communication

in these systems.

Failure Detection and Handling. If a failure can be expressed as a timeout, it is then possible to define failure mode behavior based on the occurrence of that timeout. However, explicit and more direct support for failure detection and handling does not exist in any system.

Message Properties. With the exception of Spec, none of the systems in this survey provide automatic time stamping of messages. Without timestamps it may be possible for a receiver to determine data age, for example, through the use of timeouts and channel properties; however, these methods are awkward and approximate. Two of these systems, Statecharts and CSR, lack direct mechanisms for passing message data; they communicate using signals with no data.

As noted above, all the schemes except one use at most one communications method. To describe an unsupported type of communication, it is necessary to build other abstractions from scratch or simulate them. Abstractions for DRTC are almost non-existent, and many languages have only implicit communication. In general, distributed real-time communication is not fully supported or developed in any existing scheme.

5 Towards a Specification Framework For DRTC

There is an important distinction to be made between an implementation and a requirements specification. One purpose of a specification is to provide a high-level definition for a class of acceptable system behaviors. This description can then be evaluated using a variety of different criteria. This differs from an implementation which details a particular behavior that must satisfy the specification. One of our goals is to develop abstractions for communication that operate at an appropriate level for specification. We are not interested in providing a mechanism for describing communication protocols (e.g. FDDI) but rather a framework for describing a class of communication behaviors that a set of components need to satisfy for system correctness, regardless of which protocol will eventually be used.

5.1 Language Support for DRTC

The goal is to produce a language that supports the features of DRTC described in Section 3. A general specification language should exhibit a number of properties. It should aid the writing and the understanding of a specification. The language should support DRTC explicitly; it is not enough to just provide the ability to express communication implicitly. Specification documents are not written by computer scientists but by engineers with domain specific expertise. With that in mind, it is important to design specification languages for the applications engineer and requirements writer in mind and to provide abstractions that are useful to them.

Composability is another key feature of a good specification system. The facility to effectively combine and reuse components naturally is important, especially when systems are upgraded and requirements evolve. Additionally, it is essential that a requirements language have a well understood formal basis. This make is possible to simulate a specification as well as allow the development and use of both automated and semi-automated analysis techniques.

5.1.1 Overview of SpecTRM-RL

The basis for our research is RSML and its successor SpecTRM-RL. SpecTRM-RL is a state-based specification language that is currently under development as part of a CAD system for digital automation [12]. We made this choice because of our familiarity with the language and its associated tools.[1] In addition, this family of languages has been used to specify a number of large systems [10, 11, 14].

A SpecTRM-RL description of a system consists of a set of components. It is possible to define two types of components corresponding to a standard model of control process systems shown in Figure 3:

- Control Components: These specify the computer systems that provide monitoring and control.
- Process Components: These describe the process being controlled.

The control and process components represent "templates" that provide a framework for the specification writer. Both of these templates give definitions for Inputs, Outputs, and a State Model. In addition, control components are augmented with definitions for both Operating Modes and

[1] As one of the developers of the SpecTRM toolkit, Sandys built a simulator for SpecTRM-RL.

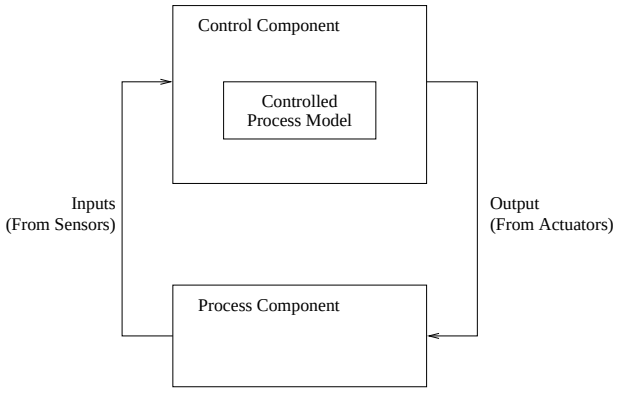

Figure 3: A control-theorist view of control systems

a CONTROLLED PROCESS MODEL. Requirements are then expressed as hierarchical state machines, where the OUTPUTS, STATE MODEL, OPERATING MODES, and CONTROLLED PROCESS MODEL all form parallel state machines.

Communication occurs implicitly using state messages. In general, the state of one component is not directly visible in another component. Communication between two components can currently be understood in two steps:

1. Any data (state information) that a component wants to be visible (sent) to other components needs to be a part of the OUTPUTS state hierarchy.

2. Inputs for a destination component C are just a mapping from a state in the OUTPUTS state hierarchy of the source component, to a local name in C.

In the next section, we illustrate SpecTRM-RL and some ideas on DRTC extensions through an example.

5.1.2 Extensions: Rod Controller Example

The application is a rod controller, the computer control component that commands a rod inside a nuclear reactor to move up – to slow down the reaction, or down – to speed up the reaction, based on the temperature inside the reactor. The communication is illustrated in Figure 4, where the rod

controller is sending commands to the rod in the reactor and is receiving updates from a sensor.

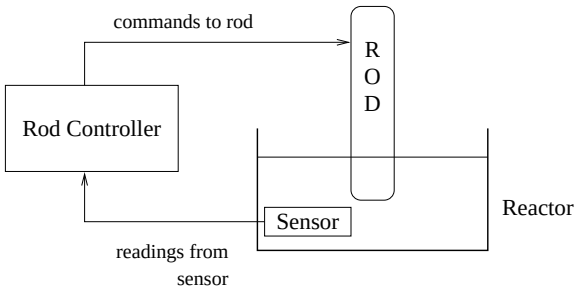

Figure 4: Communication Diagram for the Reactor System

In Figure 5, we present a partial specification in a variant of SpecTRM-RL of a simple version of a rod controller for a reactor system. In this specification, the Operating Modes, State Model, Controlled Process Model, and Outputs are simple hierarchical state machines, and Inputs describes a mapping from a numeric value in the *Sensor* component to a local name in the *RodController* component.

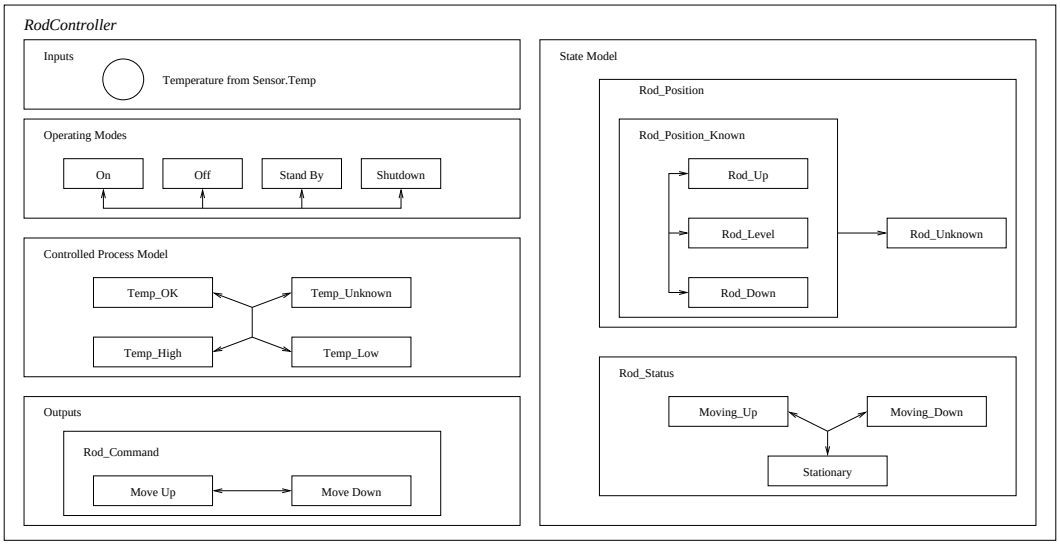

Figure 5: Partial Specification of a Rod Controller

The abstraction that we are going to use to specify communication is an *interface* definition. By carefully defining an interface scheme it should be possible to express a complete range of styles and constraints. Our interface definitions contain the properties of all three elements of our communication model, i.e. the process, the process-channel interface, and the channel. Figure 6 and Figure 7 are examples of input and output interfaces for the rod controller.

There are two parts to the output interface (Figure 6), a condition and a communication definition. In most state-based languages, each transition has some enabling condition that can lead to a state change. A similar mechanism is needed for indicating the potential start or end of a send or receive. These conditions are specified using an and-or table, where if any of the columns evaluate to true, the table evaluates to true. The second part of the interface defines the communication itself. The output interface for the rod controller describes a single output: a send of the message ROD_COMMAND to the rod component. This send is a blocking, directed, unicast send. The minimum response time for this send is 0 seconds; the maximum response time is 10 seconds; i.e. after the enabling condition becomes true, a rendezvous occurs within the bounds $[0, 10]$.

Output Interface Name: ISSUE-COMMAND
Value: ROD_COMMAND
Destination: *Rod* (* directed and unicast *)

Send Type: `Blocking`
Minimum Response Time: 0 seconds
Maximum Response Time: 10 seconds

Condition:

			OR	
AND	In (ROD_STATUS.STATIONARY)	·	T	·
	In (TEMP_OK)	·	F	·
	In (OFF)	·	F	·
	In (SHUTDOWN)	·	·	F
	In (STANDBY)	T	·	·

Figure 6: Output Interface Definitions for the Rod Controller

The input interface (Figure 7) defines how the rod controller receives information from the sensor.

This receive is a non-blocking, directed receive. A condition is used to define how often the receive is polled; in this case the interface is polled every 10 seconds. Here, Poll is a predicate that returns true when the interface is being polled. There are also data constraints; this example has a time constraint on the maximum age of the data read which can be determined by a time stamp. Other possible data constraints are value range and granularity, as well as value type.

Input Interface Name: READ-TEMPERATURE
Value: TEMPERATURE
Source: *Sensor*
Receive Type: `Non-Blocking`

Data Constraints:
Maximum Age: 20 seconds

Condition:

Duration(Not(Poll(READ-TEMPERATURE))) > 10s)	T

Figure 7: Input Interface Definitions for the Rod Controller

Note that the sensor can be a distributed component with decision making. The internal behavior of the sensor would then be described by a component definition. This interface describes how the *Rod-Controller* communicates with the *Sensor* without describing its internal behavior.

5.2 Research Issues

The last section outlined a DRTC addition to a state-based language, using the idea of a component interface containing an enabling condition or trigger and a communication definition for each input and output. We have started developing this idea and various research issues remain.

First, the elements of the interfaces need to be defined more precisely, particularly the options related to timing and data constraints. The interface definitions will lead to additional predicates (e.g. the Poll predicate used in the input interface), and other mechanisms may be necessary to allow the interface and component specifications to interact.

A major problem is to describe the semantics of the DRTC interface definitions in combination with the semantics of SpecTRM-RL, possibly in terms of a conventional state machine or a well understood operational scheme. This is necessary to perform any formal analysis of a specification, either manual or automated, and in order to simulate or execute a specification.

The role of time needs to be handled carefully. One appealing approach is to use the synchrony hypothesis [2] (zero time transitions) for the conventional component state machines, with time increasing only as a result of explicit timing statements or communication.

Much work and testing is necessary to determine the expressiveness, completeness, and convenience of the notation, no doubt resulting in many changes. The goal is to produce a language that meets the conditions of section 5.1.

6 Conclusions

This work has several themes and contributions. We have argued that distributed communications in all its diversity is an essential and complex part of real-time systems. It has been further noted that languages for requirements have emphasized the description of components but have provided little support for distributed real-time communication (DRTC). Our principal contributions are to define a useful model for examining DRTC requirements and to outline a specification approach that handles the full generality of DRTC features within a state-based notation.

Acknowledgements

We would like to thank Jon Damon Reese for his helpful discussions and his for comments on earlier drafts of this paper.

References

[1] Andrews, G. R. *Concurrent Programming - Principles and Practice.* Benjamin/Cummings, Redwood City, CA, 1991.

[2] Berry, G., and Georges, G. The esterel synchronous programming language: design, semantics and implementation. *Science of Computer Programming 19*, 87-152 (1992).

[3] Berzins, V., and Luqi. *Software Engineering with Abstractions.* Addison-Wesley, Reading, Massachucetts, 1991.

[4] Bolgnesi, T., and Brinksma, E. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN systems 14* (1987), 25–59.

[5] Gerber, R., and Lee, I. A layered approach to automating the verification of real-time systems. *IEEE Transactions on Software Engineering 19*, 9 (September 1992), 768–784.

[6] Harel, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8 (1987), 231–274.

[7] Harel, D., and Naamad, A. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology 5*, 4 (October 1996), 293–333.

[8] Hoare, C. A. R. Communication sequential processes. *Communication of the ACM 21*, 8 (August 1978), 666–677.

[9] Kopetz, H. *Real-Time Systems, Design Principles for Distributed Embedded Applications.* Kluwer Academic Publisher, Boston, 1997.

[10] Leveson, N., Alfaro, L., Alvarado, C., Brown, M., Hunt, E. B., Jaffe, M., Joslyn, S., Pinnel, D., Reese, J., Samarziya, J., Sandys, S., Shaw, A., and Zabinsky, Z. Demonstration of a safety analysis on a complex system. In *Software Engineering Laboratory Workshop* (December 1997), NASA Goddard.

[11] Leveson, N. G., Heimdahl, M. P. E., Hildreth, H., and Reese, J. D. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering* (September 1994), 684–707.

[12] Leveson, N. G., Reese, J. D., and Heimdahl, M. SpecTRM: A cad system for digital automation. In *Digital Avionics System Conference* (November 1998).

[13] Luqi. Computer-aided prototyping for a command-and-control system using caps. *IEEE Software* (January 1992), 56–67.

[14] Modugno, F., Leveson, N. G., Reese, J. D., Partridge, K., and Sandys, S. Integrated safety analysis of requirements specifications. In *IEEE International Symposium on Requirements Engineering* (1997), pp. 148–159.

[15] Perry, T. In search of the future of air traffic control. *IEEE Spectrum* (August 1997), 18–35.

[16] Reed, G., and Roscoe, A. A timed model for communicating sequential processes. In *International Colloquium on Automata, Languages and Programming* (1986), Springer-Verlag, pp. 314–323.

[17] Selic, B., Gullekson, G., and Ward, P. T. *Real-Time Object-Oriented Modeling.* John Wiley & Sons, Inc., New York, 1994.

[18] Shaw, A. C. Communicating real-time state machines. *IEEE Transactions on Software Engineering 18*, 9 (September 1992), 805–816.