SAE The Engineering Society For Advancing Mobility Land Sea Air and Space®
INTERNATIONAL
400 Commonwealth Drive, Warrendale, PA 15096-0001

SURFACE VEHICLE INFORMATION REPORT

SAE J2356

ISSUED SEP1997

Issued 1997-09

Submitted for recognition as an American National Standard

A GRAPHICAL MODEL FOR INTERACTIVE DISTRIBUTED CONTROL

Foreword—This document defines an architectural model of a Distributed Embedded System and a method of graphically representing the events, activities, and transactions that may be used for supporting interactive Distributed Control.

TABLE OF CONTENTS

QUESTIONS REGARDING THIS DOCUMENT: (724) 772-8512 FAX: (724) 776-0243
TO PLACE A DOCUMENT ORDER; (724) 776-4970 FAX: (724) 776-0790
SAE WEB ADDRESS http://www.sae.org

1. Scope—The demonstrated architectural model and associated graphical techniques defined herein were developed to provide a simple method of visualizing the general functional operation or behavior of a Distributed Embedded System with a strong emphasis on representing system time characteristics.

1.1 Background

1.1.1 WHAT IS INTERACTIVE DISTRIBUTED CONTROL?—Interactive Distributed Control, also identified as IDC, is an information and control transfer strategy that allows a network of microcomputer-based modules to support a collection of high-speed distributed functions.

1.1.2 WHY WAS THE SAE IDC COMMITTEE FORMED?—Three key motivations led to the formation of the Interactive Distributed Control Committee.

First—The timing analysis of whether high-speed distributed systems have the capacity to adequately handle all the required functions is critical.
Second—The design of an interactive distributed control system, the development of hardware and software, and the selection of a suitable network protocol can benefit from such analysis prior to system realization.
Third—The interest to establish a benchmark to determine the effectiveness of an IDC system.

As a result, the IDC Committee's main focus is to study the automotive application of high-speed multiplexing (small area networking) to handle distributed motion control.

1.1.3 WHY ARE HIGH-SPEED DATA TRANSFERS NEEDED?—It is quite possible that vehicle motion control processing may become more distributed. The timing of these motion control algorithms, like engine control, braking, traction control, speed control, and transmission control, is considered critical to proper operation.

1.1.4 WHAT PROBLEMS ARE ENCOUNTERED WHEN USING MULTIPLEX FOR IDC?—Two key IDC problems or questions need to be addressed:

a. Is the network speed and bandwidth capable of supporting all of the intended high-speed distributed functions?
b. What microcomputer processing power is required to support the module-related high-speed distributed functions?

Analyzing these problems is the primary focus of the IDC Committee and the basis for this document that examines the modeling and timing characteristics of a generalized Interactive Distributed Control system.

2. References

2.1 Applicable Publications—The following publications form a part of the specification to the extent specified herein. Unless otherwise indicated the latest revision of SAE publications shall apply.

2.1.1 SAE PUBLICATIONS—Available from SAE, 400 Commonwealth Drive, Warrendale, PA 15096-0001.

SAE J1213-1—Glossary—Vehicle Networks for Multiplexing and Data Communications
SAE J1850—Class B Data Communications Network
SAE J1930—Electrical/Electronic Systems Diagnostic Terms, Definitions, Abbreviations, and Acronyms
SAE J1939—Heavy Truck High-Speed CAN
SAE J2056—Class C Multiplexing
SAE J2284—Automotive High-Speed CAN

2.2 Other Publications

Mullender, Sape, Distributed Systems - Second Edition. Addison-Wesley, 1993.

3. *Definitions*

3.1 Distributed Embedded System—A general form of a distributed system characterized by being confined to a small geographic area, interconnected using small area networking, and implemented using microcomputer-based modules, smart sensors, and smart actuators. Alternate term—Small Area Distributed System.

3.2 Distributed Function—A general function which is physically distributed across two or more system entities.

3.3 Interactive Distributed Control—An information and control transfer strategy that uses interaction between a network of microcomputer-based modules to support a collection of high-speed distributed functions.

4. *Abbreviations/Acronyms*

DES—Distributed Embedded System
IDC—Interactive Distributed Control

5. *General—*The intent is to develop a simple and generalized architectural model that includes the essential elements required to support a collection of distributed functions typically found within the boundaries of a Distributed Embedded System. Using these elements or building blocks, the model is further expanded such that the timing aspects of the distributed processes can be both defined and graphically depicted. Three time diagramming formats are presented.

5.1 Essential Building Blocks—A general distributed function, process, or system can be conceptually constructed using five architectural elements (Figure 1). Three elements common to any general system are the input, control, and output building blocks. The other two, transmit and receive, are unique only to distributed systems which have been physically partitioned into two or more entities (usually called nodes of a network).

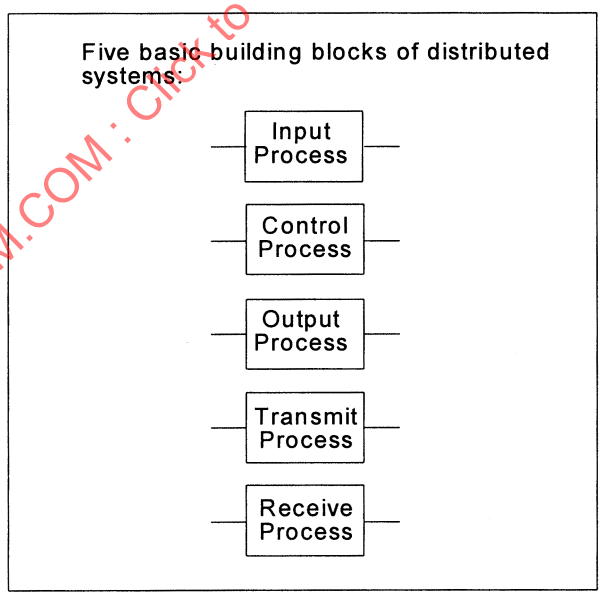

FIGURE 1—ELEMENTS OF THE DISTRIBUTED FUNCTION

a. The **Input** element interconnects an input device or signal to a control element by providing a data representation of the input. In some cases, an input may be connected to more than one control.
b. The **Control** element contains the algorithm for translating input mechanism abstractions into output mechanism abstractions and implements the desired customer and system behaviors for a given function or feature.
c. The **Output** element accepts information in the form of a data structure from a control element and converts the information into system output responses which may be signals or actuated devices.
d. The **Transmit** element accepts information in the form of a data structure from either an input or control element and converts the information into a data transfer activity on a network.
e. The **Receive** element accepts a data transfer from a network and interconnects the received data structure to the correct control or output element.

5.2 Distributing a Function—Once the decision to distribute a system has been made, the selection of partitioning boundaries for each function within the system must be made. This development step will establish the module to which each architectural element belongs. The partitioning choices usually made by the Distributed Embedded System designer and several factors may influence the physical boundary selection including:

a. The type of implemented control architecture, especially whether system-level control is administered centrally or distributed.
b. The type of implemented software architecture, especially whether a standardized software building block approach is used.
c. The expected data rate at a potential boundary; usually partitioning at boundaries that produce lower data transfer rates is preferable to a selection at a high data range.
d. The available microcomputer resources within a related electronic module and the location of the input or output.

Once the boundary is established, the two network connected process blocks, transmit and receive, are appropriately inserted into the overall process as shown in Figure 2. In general, the distributed function contains one of each of the five DES architectural elements.

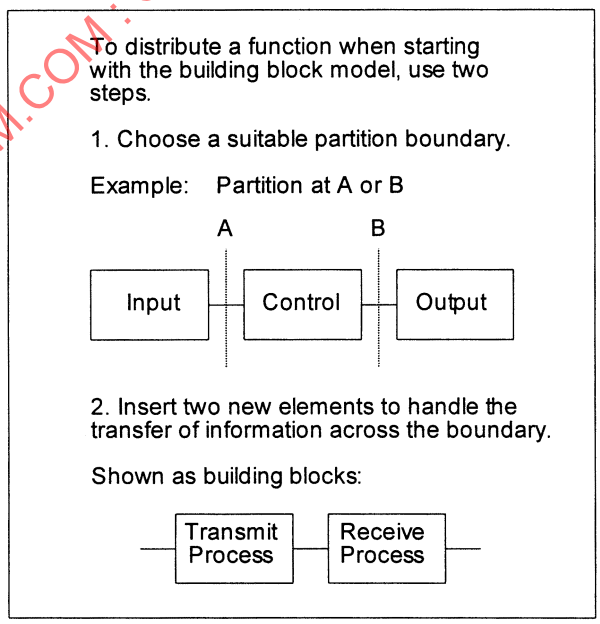

FIGURE 2—DISTRIBUTING A FUNCTION

Based on this architectural model, the partitioning selection appears to be quite simple, especially if each elemental block has already been prepackaged. However, when one moves to implement these concepts, the meaning and importance of partitioning becomes clear. The major realization is the fact that a partitioning boundary is always located within the software.

In the software world, these boundary locations are called interfaces. Each software interface will require a structural design to handle both process invocation and data transfer. Good software design principles encourage the creation of standard or logical interfaces at these partitioning locations. This disciplined approach provides a consistent, easier to manage, and potentially reusable software architecture.

The two examples, Figure 3 and Figure 4, show the common partitioning locations. While partitioning within an input, control, or output block is also achievable, this document does not consider this topic.

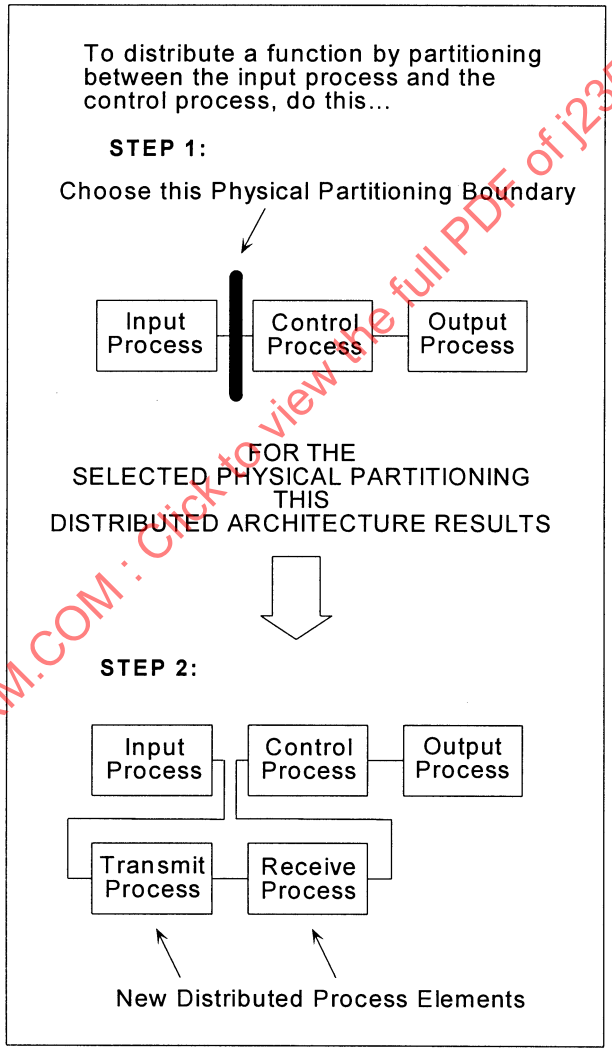

FIGURE 3—PARTITIONING EXAMPLE

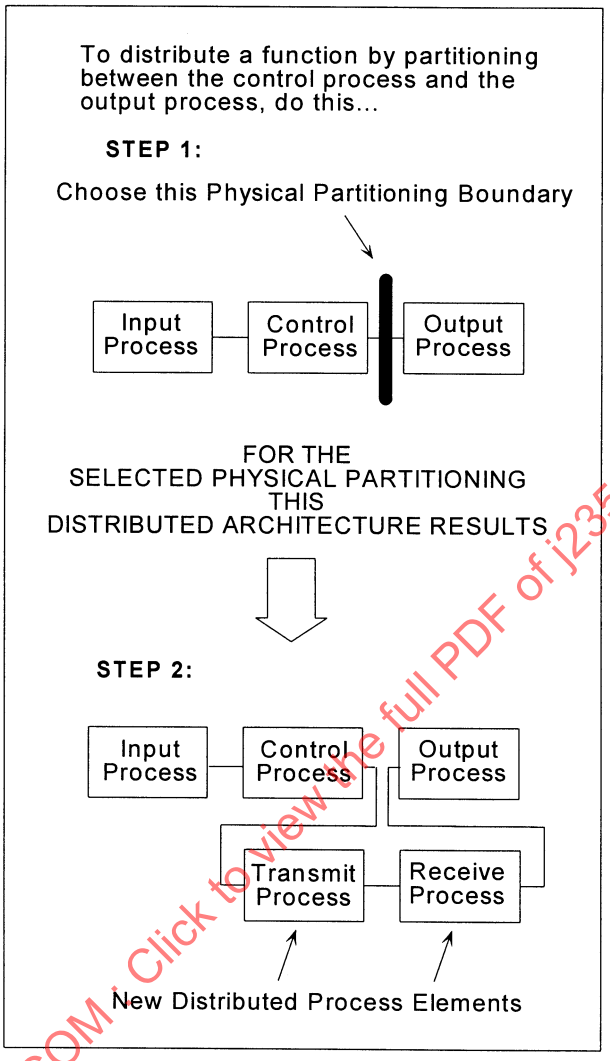

FIGURE 4—ANOTHER PARTITIONING EXAMPLE

Some distributed embedded system designers and experienced software developers recommend:

a. Matching the overall system software architecture to the distributed function architecture.
b. Partitioning using standardized software building blocks to reduce complexity.

5.3 The Event, Activity, and Transaction Diagrams—To help visualize the processes of a distributed embedded system, representations of the system's events, activities, and transactions may be used. Each representation models and shows the timing characteristic of a functional component of the system.

An **event** is an occurrence that is recognizable at a specific point in time, for example, a power-up event or a reset event. The event diagram is shown in Figure 5. An **activity** is a process, a collection of related operations all assigned to a designated group. For example, the initialization of a network is an activity. The activity diagram is shown in Figure 6. A transaction is a representation of an activity used to show the movement from one activity or event to the next. A series of transactions could be used to model a complete distributed function showing each activity across the entire system. The transaction diagram is shown in Figure 7.

Each of these diagram methods is related, as shown in Figure 8, and may be used to show specific timing.

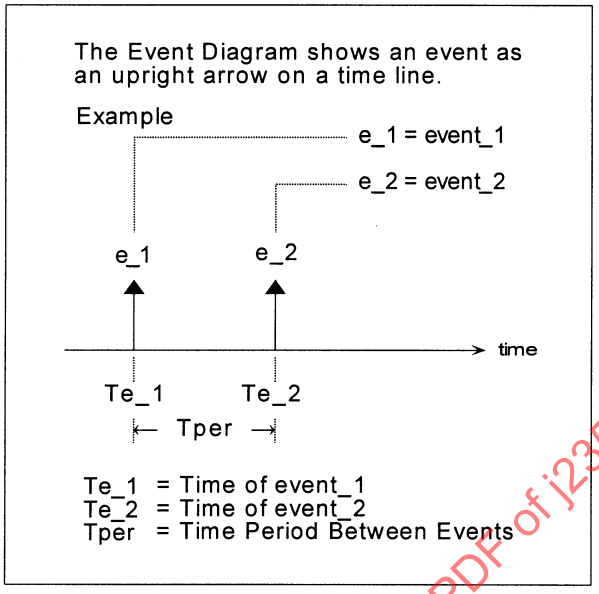

FIGURE 5—EVENT DIAGRAM

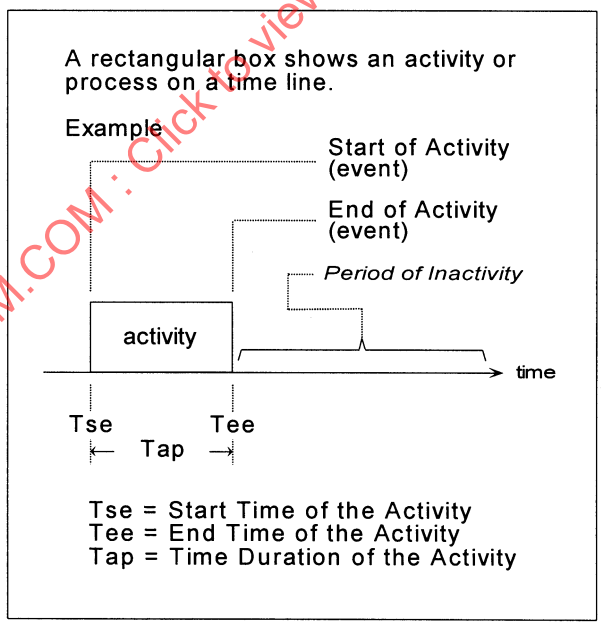

FIGURE 6—ACTIVITY DIAGRAM

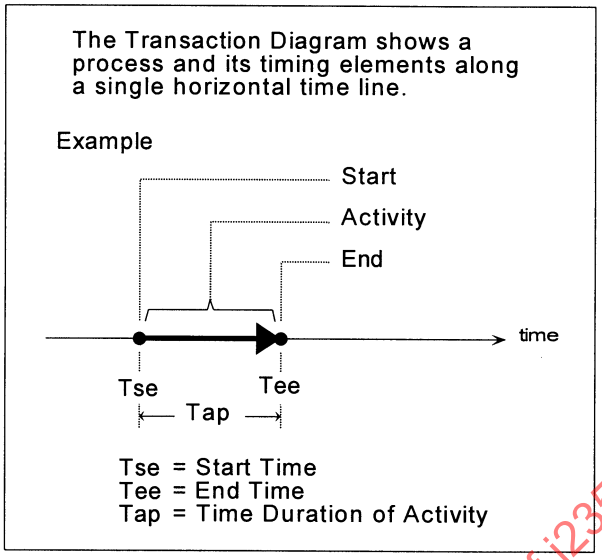

FIGURE 7—TRANSACTION DIAGRAM

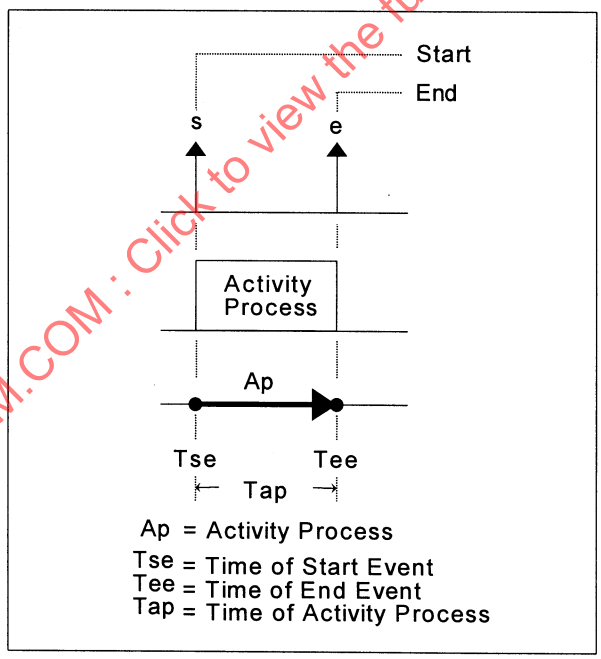

FIGURE 8—ALL DIAGRAM REPRESENTATIONS

5.4 Model of a General Function—The graphical depiction of a general function, a function which may or may not be distributed, can always be represented by the three common building blocks - input, control, and output. Although such a graphical representation is useful, the building block diagram is a physical representation of a function and lacks the ability to show the element of time.

The comparable activity diagram is a functional representation of the same familiar building block components. However, this simple transition, shown in Figure 9 using the same functional elements, now includes the added capacity to indicate time.

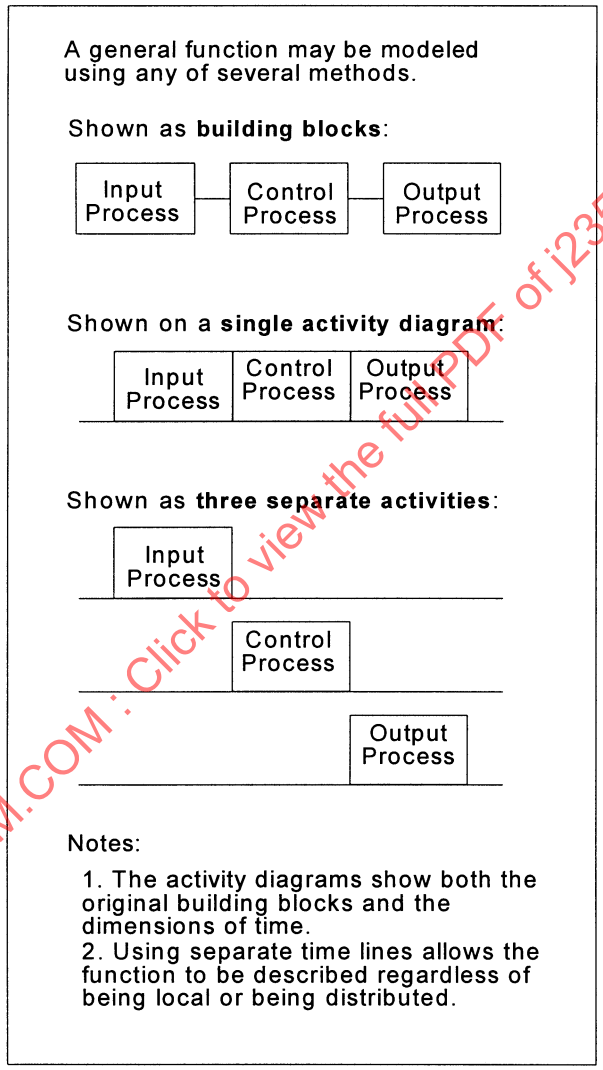

FIGURE 9—MODEL OF A GENERAL FUNCTION

The system designer may choose to show activities on the same time line or indicate each process with its own distinctive time line. Whether the general function is located within a single module or distributed across a group of modules, the choice of multiple separate activity diagrams allows functions to be characterized regardless of the selected physical partitioning. This technique of separating the system activities into individual constituents, which is effectively decomposing the system by function rather than by module, is called functional partitioning.

One may also augment or substitute the activity diagram with a transaction diagram, shown in Figure 10, which can help to show the flow of the function's process components. While the value of the transaction diagram at present seems comparable to that of the activity diagram, the transaction diagram has advantages that become more obvious as the general function becomes implemented as a distributed function. As will be shown later, the transaction allows the ability to interconnect activities of the distributed embedded system.

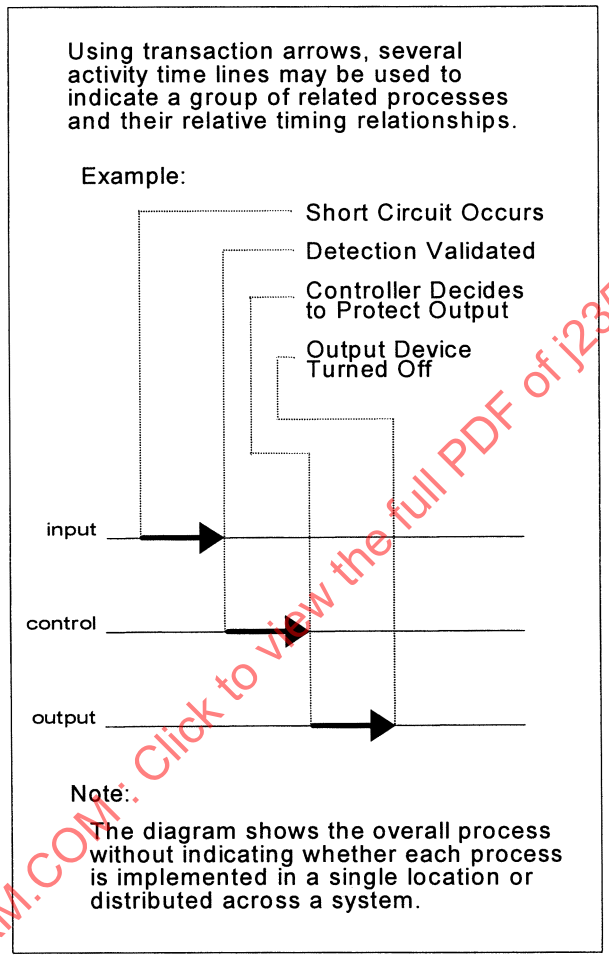

FIGURE 10—MODELING USING TRANSACTIONS

5.5 Process Models—Each of the five architectural elements – input, control, output, transmit, and receive – is a simple process model used to characterize specific types of distributed embedded system behavior. A process model may be represented as a building block, as a collection of events, as an activity, or as a transaction.

5.5.1 INPUT PROCESS—From the distributed embedded system point of view, the **Input Process**, shown in Figure 11, handles the movement of information into the system. The activity includes the detection of input devices and signals, the filtering of such information to distinguish between noise and valid input events, and the packaging of such input information into data structures designed for transfer to the next sequential activity.

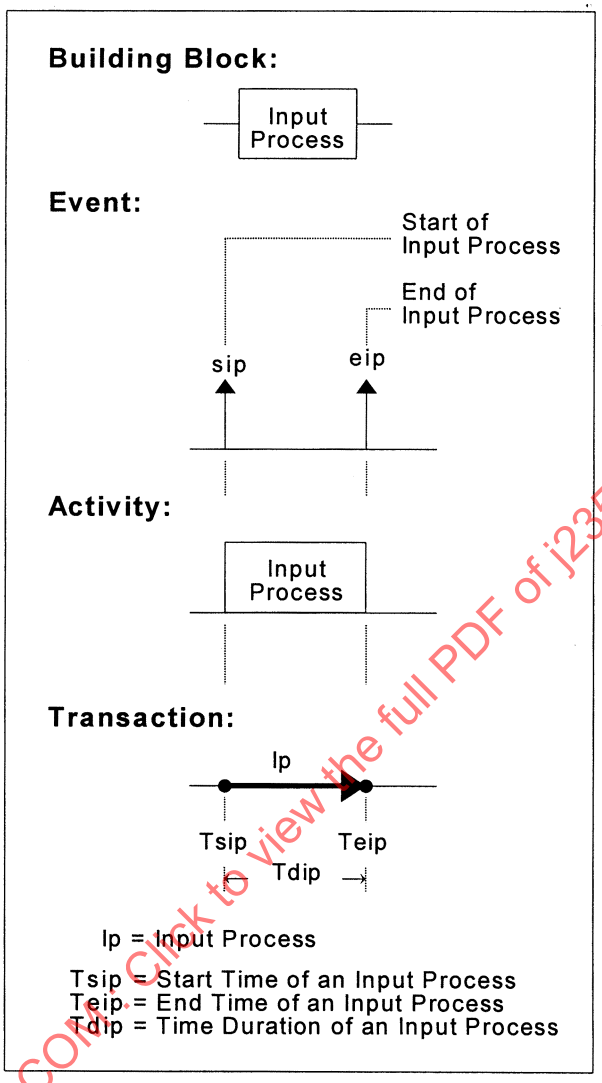

FIGURE 11—INPUT PROCESS

When implemented in software, the input process is a scheduled activity handled as a part of an overall sequence. As a result, a time difference between an actual system input event and the detection of this event will include a certain amount of asynchronous time delay. This and other time parameters are of key interest to the study of high-speed Interactive Distributed Control.

An example of an input process might be the handling of an input switch will require waiting until the software task scheduler begins the input process, reads the switch via an input port, debounces the switch to qualify the state, and updates the related data structure for the next functional activity.

5.5.2 OUTPUT PROCESS—The **Output Process**, shown in Figure 12, establishes the results of the system, essentially moving the information to the end of the overall process. The activity includes accepting a transfer of output-related information from a previous activity, any required pre-processing output operations, perhaps a small amount of output-control processing, and the specific handling of actuating output devices or signals.

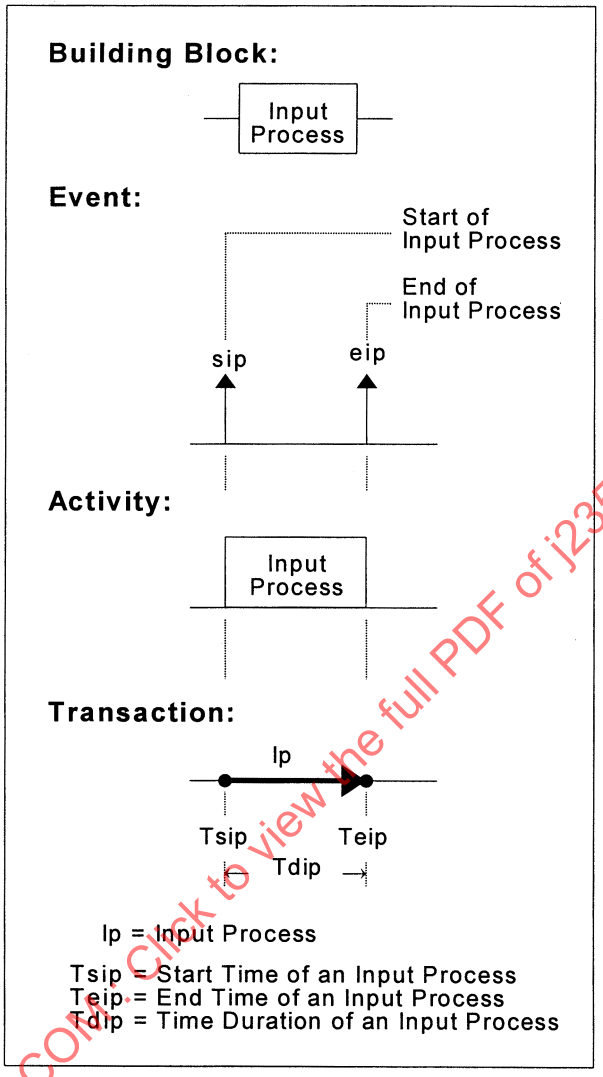

FIGURE 12—OUTPUT PROCESS

When implemented in software the output process, like the input process, is a scheduled activity handled as a part of an overall sequence. As a result, there is an asynchronous time delay between the expected system output event and the actual output of this event. This also is a parameter of key interest to the study of high-speed Interactive Distributed Control.

An example of an output process could be the processing of a solenoid-based mechanism. Operations might include a waiting time until the software task scheduler invokes the output processing routine, an examination of the associated output process data structure, a check of the current solenoid position, an activation of the solenoid actuator circuit, and perhaps an input closely coupled to an output like a short-circuit detection circuit.

5.5.3 CONTROL PROCESS—The **Control Process**, shown in Figure 13, collects the required input information and provides the expected system behavior by establishing both its own operational states and the appropriate output information for the system. The activity includes the examination of input data structures, the execution of algorithms that meet the feature or function requirements, and the packaging of output information into data structures designed for transfer over the network or locally to the next sequential output process activity.

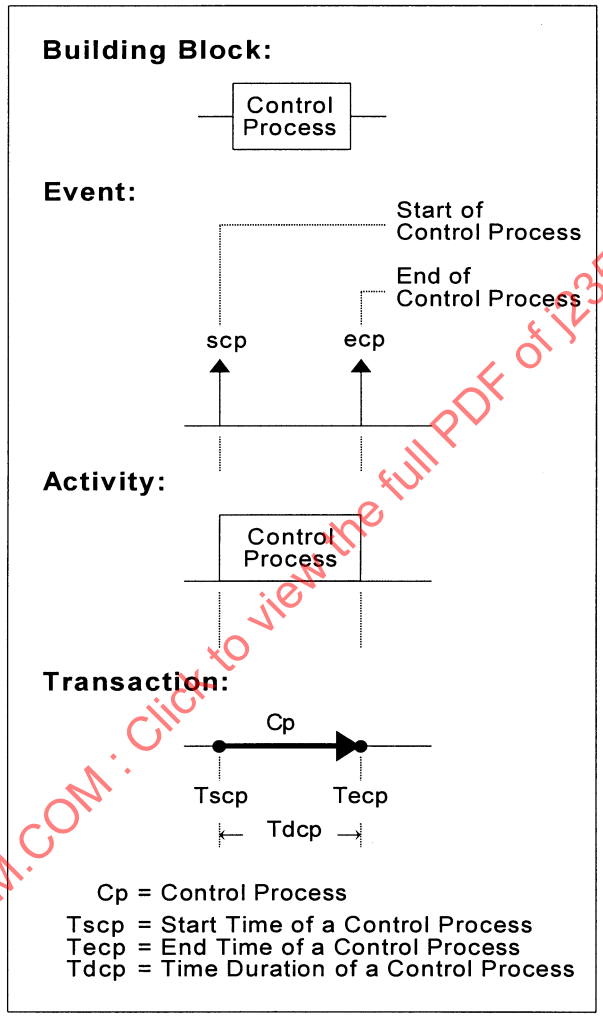

FIGURE 13—CONTROL PROCESS

When implemented in software the control process, like the input and output processes, is a scheduled activity handled as a part of an overall system sequence. Another task scheduler waiting time until the invocation of the control-processing routine is usually a part of the implementation. As a result, the control process also includes a timing element of asynchronicity which is also important to an understanding of high-speed IDC.

5.5.4 TRANSMIT PROCESS—The **Transmit Process**, shown in Figure 14, moves data from a particular input or control process onto the selected network as serial data transfers. It is specific to a distributed embedded system and is used only when a general function is partitioned and becomes a distributed function.

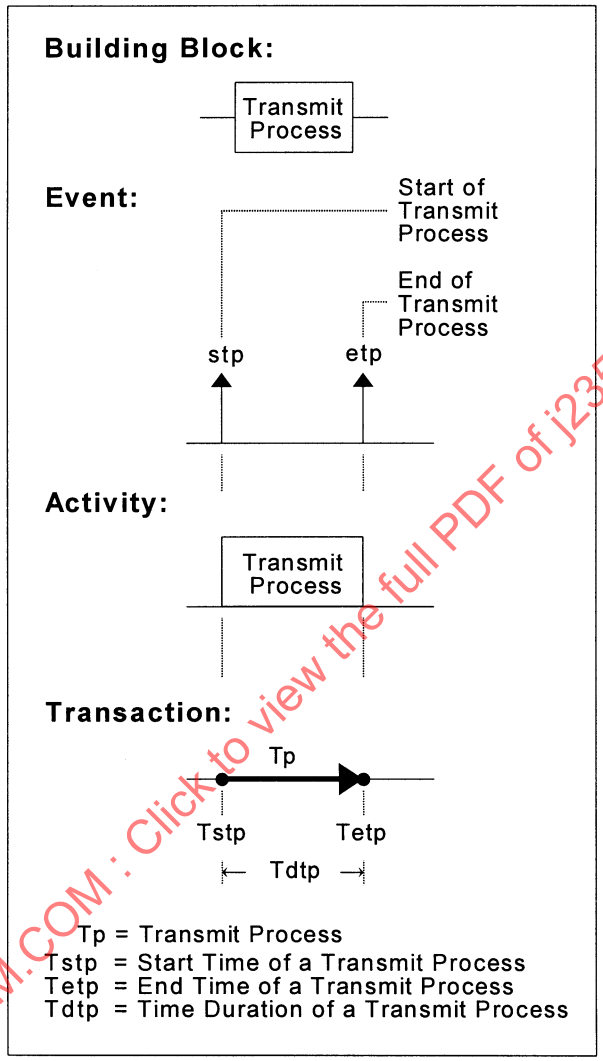

FIGURE 14—TRANSMIT PROCESS

Transmit process activities may include accepting a transmit operation from the previous activity, the forming of network messages, the handling of network transmit behavior, the movement of message information into a possible protocol IC, the actual placement of network messages onto the media, a possible transmission retry strategy, and typically a signal back to the initiating activity indicating a successful message transmission.

When implemented as a combination of software and hardware, the timing of the transmit process, like the other processes, is primarily dependent on the software scheduling, the network transfer hardware, and the current network traffic level. These time dependencies and other aspects of the transmission process produce time delay between the initiation of a transmit process event and the subsequent initiation of the corresponding network message. The time characteristics of the transmit process including the network speed are important parameters of the IDC effort.

5.5.5 RECEIVE PROCESS—The **Receive Process**, shown in Figure 15, extracts serial-data transfers from the embedded network and moves the received data information to a designated control or output process. Like the transmit process, the receive process is specific to a distributed embedded system and is used only when a general function is partitioned and becomes a distributed function. Additionally, the receive process and the transmit process always occur as a pair.

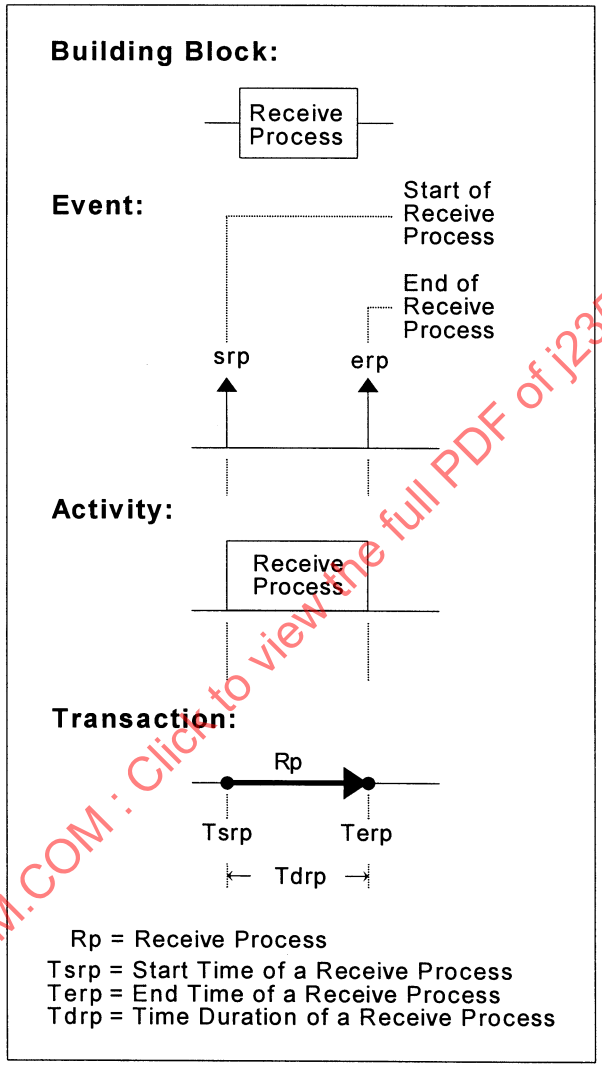

FIGURE 15—RECEIVE PROCESS

Receive process activities may include the accepting and acknowledging a message reception from the network, the parsing (or lookup) of a received message, the handling of network receive behavior, the movement of message information out of a possible protocol IC, the actual placement of the network message information into a suitable data structure for the next expected DES process.

When implemented as a combination of software and hardware, the timing of the receive process, similar to all of the functional processes, is primarily dependent on the software scheduling and on the network reception hardware. These reception time dependencies introduce a time delay between the reception of a network message event and the subsequent initiating event of the next expected control or output process.

Aside from the message transfer time, the time characteristics of the receive process are independent and separate from the transmit process and are important IDC parameters.

5.6 Interconnections—The **Interconnection** is an activity which is used to enjoin two processes which are distinctively separate from each other. One example of an interconnection is a message transferred over a small area network used to continue the sequential operation of a distributed function.

Another common form of interconnection is specific to the software processing activities of a typical microcomputer used to implement the functionality of a Distributed Embedded System. Implementing distributed functions in software constrains the execution of the required functions to be sequential. As a result, software processing is handled as a time ordered collection of sequential activities and the order of these activities is determined by a software task scheduling algorithm. In the distributed environment, software processes require interconnections and it is the chosen form of task scheduling which establishes the resulting characteristics of the interconnections.

Interconnections may be graphically shown and timing characteristics may also be included.

The interconnection diagram, shown in Figure 16, is an extension of the transaction diagram using individual time lines with transactions to indicate the related processes which require enjoining. The interconnection is shown by adding a diagonal transaction arrow from the end of a process to the beginning of the next sequential process.

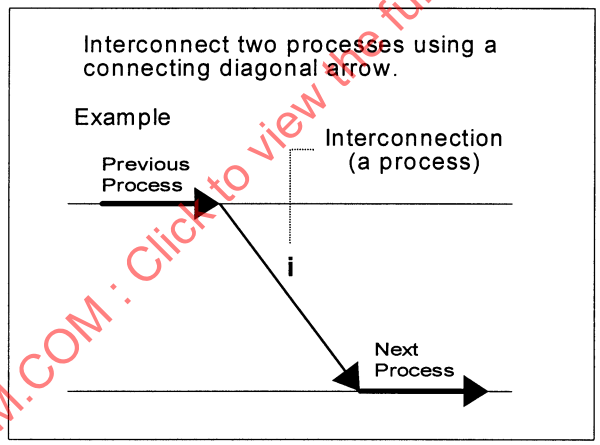

FIGURE 16—INTERCONNECTION

Interconnections consume a portion of a distributed function implementation and as a result, become an important IDC parameter. The timing characteristics of interconnections, shown in Figure 17, are of major interest to the Distributed Embedded System designer.

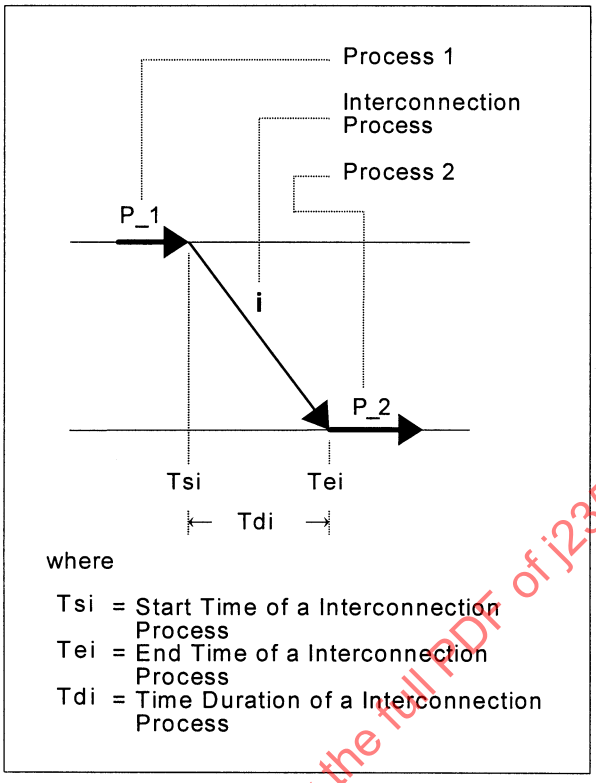

FIGURE 17—INTERCONNECTION TIMING

When the interconnection is established by using a network or an electrical signal (i.e., a PWM signal used for dimming), timing analysis of the hardware implementation must be considered. When the interconnection is accomplished using software, the timing characteristics of the software task scheduling methods must be considered. Several different methods of performing task scheduling are available and each has a unique timing characteristic. Some of the more prevalent techniques include:

a. **Round Robin**—The entire group of tasks is ordered and essentially executed one after another.
b. **Event Driven**—Only the task for which an event, as occurred, is executed.
c. **Time Sliced**—Tasks are divided into groups and are assigned a specific time period to be executed.

Since the implementation of a Distributed Embedded System will generally use multiple microcomputer-based modules, it can be expected that the selected method of task scheduling will be different across the system and, at present, not consistent to any standard.

5.7 Diagramming the Distributed Function—Describing distributed functions is difficult especially when beginning with behavioral details. Starting with generalizations to define functional behavior rather than specific cases may offer an advantage in dealing with the complexity of distributed architectures.

Characterizing the distributed function as a collection of closely related functional elements (input, control, output, transmit, and receive) produces a simple model. One obtains a specific model by adding specific names to the generalized elements. The general concept is reusable.

Using the basis of the transaction diagram, the general distributed function may be graphically shown in Figure 18 using the five architectural elements of the Distributed Embedded System. This tool provides a visual representation of the operational behavior of a distributed function, adds the ability to indicate system timing characteristics, and provides the opportunity to show the process interconnections (see Figure 19).

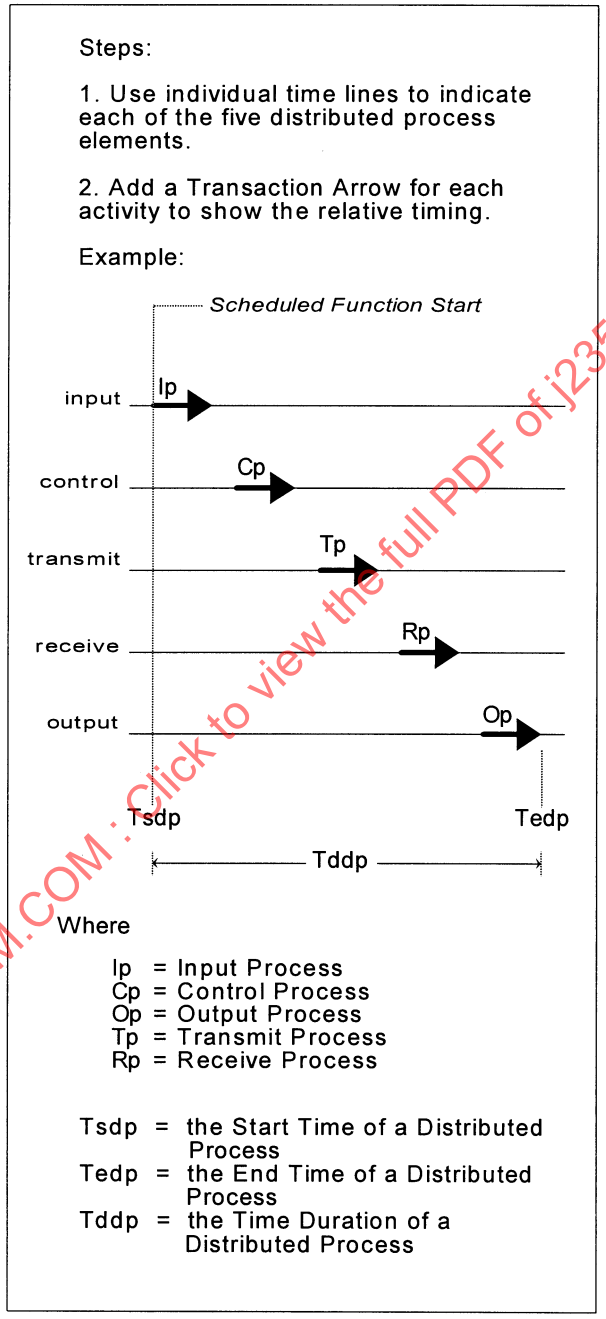

FIGURE 18—DIAGRAM USING TRANSACTIONS

If a Distributed System is defined using the following five architectural elements:

Ip	Input Process
Cp	Control Process
Op	Output Process
Tp	Transmit Process
Rp	Receive Process

then, the following **INTERCONNECTIONS** are required in order to enjoin the complete distributed function:

ICi	Input-to-Control Interconnection
COi	Control-to-Output Interconnection
ITi	Input-to-Transmit Interconnection
CTi	Control-to-Transmit Interconnection
TRi	Transmit-to-Receive Interconnection
ROi	Receive-to-Output Interconnection
RCi	Receive-to-Control Interconnection

The transaction diagram may now also be used to indicate the interconnection timing associated with a given distributed function.

Example:

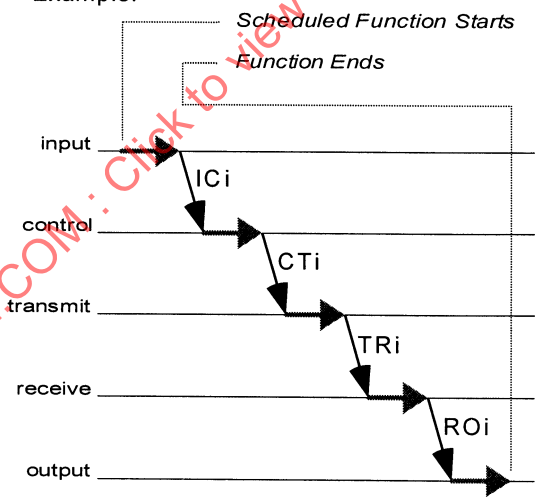

FIGURE 19—COMPLETE DISTRIBUTED FUNCTION

The timing of a general distributed function is composed of nine components; five functional process times and four interconnection times. These nine timing parameters, as ordered by the previous distributed function diagram examples, are

Tdip=Input Process Time
Tdici=Input-to-Control Interconnection Time
Tdcp=Control Process Time
Tdcti=Control-to-Transmit Interconnection Time
Tdtp=Transmit Process Time
Tdtri=Transmit-to-Receive Interconnection Time
Tdrp=Receive Process Time
Tdroi=Receive-to-Output Interconnection Time
Tdop=Output Process Time

Of these nine distributed process times, only one, the Transmit-to-Receive Interconnection Time, shown in Figure 20, is directly related to the data-transfer speed of the selected small area network. Of key importance is the observation that if the network transfer time is small compared to the remainder of the composite process, then the addition of a faster network solution will have virtually no effect on the overall timing of the distributed function. Three of the four interconnection times, as shown in Figure 21, are software related and a direct function of the selected task scheduling algorithms in each of the participating modules. Additionally, it is again important to point out that the initiation of the input process is also a scheduled activity and a higher level transaction diagram showing the beginning of a single distributed function would be necessary when analyzing the timing of multiple distributed functions.

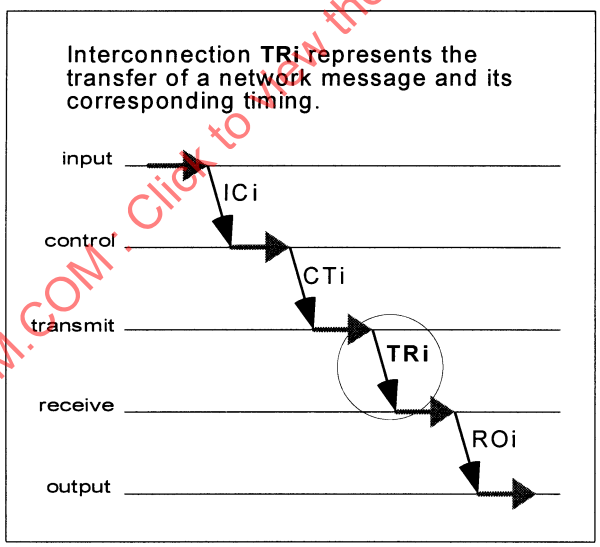

FIGURE 20—THE NETWORK INTERCONNECTION

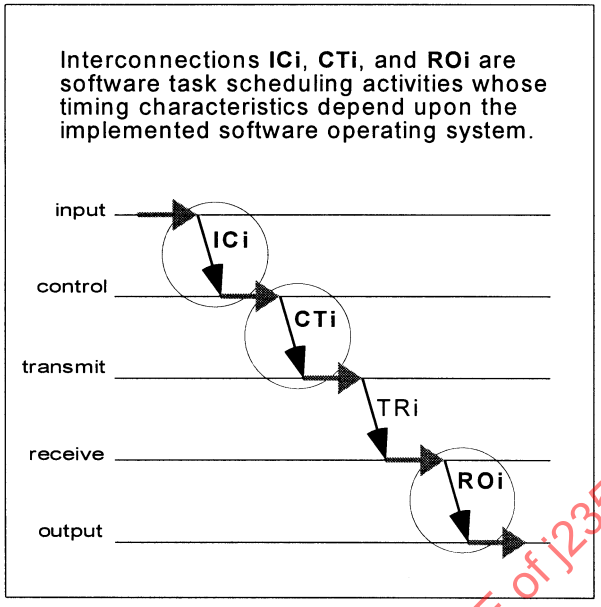

FIGURE 21—SOFTWARE TASK SCHEDULING ACTIVITIES

Typically, software task scheduling is asynchronous with respect to the timing of a distributed function. This means interconnection time is a variable. As a result, the complete processing time of any given distributed function is not constant. Expected response times will statistically vary and depend to a large extent upon the task scheduling algorithm.

Currently, the selection of a task scheduling method is determined by the module software designer to meet the local requirements of module functionality and is usually created to be unique for each application. This means that a certain amount of variability between modules can be expected.

In addition to the timing of task scheduling, the timing characteristics of the individual software processes that handle each of the five functional elements is also of importance. Implementation from one software designer to the next will produce further variability in the timing of high-speed distributed processing.

If characterizing high-speed motion control function timing is a firm requirement, distributed embedded system designers will need to understand the methods, the implementation details, and the timing characteristics of software processing and of task scheduling. Some level of software standardization may be required.

6. ***Summary and Conclusions***—This document has focused on a method of modeling events and timing parameters associated with a future, high-speed application area called Interactive Distributed Control. An architectural model has been presented along with several graphical techniques that may be useful to describe the events, activities, and transactions of distributed functions. The focus has been to provide a diagrammatic method of representing the functional operation and the timing characteristics of the fundamental components of a general Distributed Embedded System.

As this document has demonstrated, the analysis of a generalized distributed function shows insight into the challenges related to the design of high-speed IDC systems.

Three conclusions are recognized by the IDC committee:

a. The choice of the networking solution for an IDC system has a relatively small impact on whether or not the distributed application's timing requirements are met. As the model demonstrates, software design has by far the greatest impact on the end-to-end delay experienced by the distributed process. This clearly indicates that significant study must still be performed in the area of Distributed Embedded System software processing and that some level of standardization in this area may be necessary.
b. In order to properly analyze a system with many high-speed distributed functions operating simultaneously, computer-aided design tools will be needed to support the Distributed Embedded System modeling and to characterize timing behavior.
c. A different microcomputer architecture may be necessary in the future, optimized for distributed applications rather than centralized control.

PREPARED BY THE SAE INACTIVE DISTRIBUTED CONTROL TASK FORCE SUBCOMMITTEE
OF THE SAE VEHICLE NETWORK FOR MULTIPLEX AND DATA
COMMUNICATIONS STANDARDS COMMITTEE

APPENDIX A

TIMING SYMBOLS AND DEFINITIONS

Tap — Time of Activity process

A general term defining the time duration from the beginning of an activity (start event) to the end of the activity (end event).

Tap = Tee – Tse

Tdcoi — Control-to-Output Interconnection Time

The time duration from the beginning of a Control-to-Output Interconnection activity (start event) to the end of the Control-to-Output Interconnection activity (end event).

Tdcoi = Tecoi – Tscoi

Tdcp — Control Process Time

The time duration from the beginning of a Control Process (start event) to the end of the Control Process (end event).

Tdcp = Tecp – Tscp

Tdcti — Control-to-Transmit Interconnection Time

The time duration from the beginning of a Control-to-Transmit Interconnection activity (start event) to the end of the Control-to-Transmit Interconnection activity (end event).

Tdcti = Tecti – Tscti

Tddp — Distributed Process Time

The time duration from the beginning of a Distributed Process (start event) to the end of the Distributed Process (end event).

Tddp = Tedp – Tsdp

Tdgp — General Process Time

The time duration from the beginning of a General Process (start event) to the end of the General Process (end event).

Tdg = Teg – Tsg

Tdi — Interconnection Time

The time duration from the beginning of an Interconnection activity (start event) to the end of the Interconnection activity (end event).

Tdi = Tei – Tsi