AD-A275 336

②

DTIC
S ELECTE
FEB 07 1994
A

USING RDD-100 WITH CoRE

SPC-93099-CMC

VERSION 01.00.03

JANUARY 1994

94 2 04 122

Best
Available
Copy

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 19 January 1994	3. REPORT TYPE AND DATES COVERED Technical Report - Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
Using RDD-100 with CoRE	
6. AUTHOR(S) R. Kirk, H. Osborne Produced by Software Productivity Consortium under contract to Virginia Center of Excellence	G MDA972-92-J-1018

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Virginia Center of Excellence SPC Building 2214 Rock Hill Road Herndon, VA 22070	8. PERFORMING ORGANIZATION REPORT NUMBER SPC-93099-CMC, Version 01.00.01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) ARPA/SISTO Suite 400 801 N. Randolph Street Arlington, VA 22203	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

N/A

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
This document has been approved for public release and sale; its distribution is unlimited. No Restrictions	1

13. ABSTRACT (Maximum 200 words)

   This technical report describes how to develop a specification of software requirements using the Software Productivity Consortium's CoRE software requirements method from a system requirements and design specification developed using Ascent Logic Corporation's RDD-100. Specifically, this report describes the transition from system design to software requirements analysis when: a) RDD-100 has been used to create system requirements and design specifications, or: b) the software requirements analysis activity is to be performed according to the CoRE method.

   RDD-100 is the implementation of the Requirements Driven Design (RDD) system design and engineering method, wherein the system design is driven by the customer's expectations of system behavior. CoRE is the Software Productivity Consortium's software requirements engineering method, created to support the development of precise, testable specifications that are demonstrably complete and consistent.

   An RDD-100 system model contains all of the information needed to begin building a CoRE software requirements specification. This paper describes a process and guidelines for building a CoRE software requirements specification from an RDD-100 system model. The process consists of the following high-level activities: a) identify sources of the necessary CoRE information in the RDD-100 model; b) map that information to an initial CoRE specification; c) complete the CoRE specification.

   The Consortium's experience in the use of CoRE with RDD-100 is based on the development of a CoRE specification of software requirements for the HAS Buoy problem from an RDD-100 model of system requirements and design. The guidance provided in this report documents our experiences and lessons learned from this case study.

14. SUBJECT TERMS Requirements, CASE tool, specification, CoRE, RDD-100, system design	15. NUMBER OF PAGES 70
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

USING RDD-100 WITH CoRE

SPC-93099-CMC

VERSION 01.00.03

JANUARY 1994

Richard A. Kirk
Haywood S. Osborne

Produced by the
SOFTWARE PRODUCTIVITY CONSORTIUM SERVICES CORPORATION
under contract to the
VIRGINIA CENTER OF EXCELLENCE
FOR SOFTWARE REUSE AND TECHNOLOGY TRANSFER

SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

DTIC QUALITY INSPECTED 8

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

This page intentionally left blank.

EXECUTIVE SUMMARY

This technical report describes how to develop a specification of software requirements using the Software Productivity Consortium Requirements Engineering (CoRE) software requirements method from a system requirements and design specification developed using Ascent Logic Corporation's RDD-100. Specifically, this report describes the transition from system design to software requirements analysis when:

- RDD-100 has been used to create system requirements and design specifications.

- The software requirements analysis activity is to be performed according to the CoRE method.

RDD-100 is the implementation of the Requirements Driven Design (RDD) system design and engineering method (Ascent Logic Corporation 1992a), wherein the system design is driven by the customer's expectations of system behavior. CoRE is the Software Productivity Consortium's software requirements engineering method (Software Productivity Consortium 1993) created to support the development of precise, testable specifications that are demonstrably complete and consistent.

An RDD-100 system model contains all of the information needed to begin building a CoRE software requirements specification. This report describes a process and guidelines for building a CoRE software requirements specification from an RDD-100 system model. The process consists of the following high-level activities:

- Identifying sources of the necessary CoRE information in the RDD-100 model

- Mapping that information to an initial CoRE specification

- Completing the CoRE specification

The Consortium's experience in the use of CoRE with RDD-100 is based on the development of a CoRE specification of software requirements for the Host-at-Sea (HAS) Buoy problem from an RDD-100 model of system requirements and design. The guidance provided in this report documents the Consortium's experiences and lessons learned from this case study.

This page intentionally left blank.

1. INTRODUCTION

The RDD-100 tool was developed to support a systems engineering process wherein the system design is driven by the customer's expectations of system behavior. The Consortium Requirements Engineering (CoRE) method is a software requirements engineering method supporting the development of precise, testable specifications that are demonstrably complete and consistent. The Requirements Driven Design (RDD) systems engineering methodology supported by RDD-100 is generally unrelated to the CoRE method, meaning that any direct support from RDD-100 for CoRE is coincidental. Therefore, the best approach to using RDD-100 and CoRE together is to determine how to transition from system specification using RDD-100 to software requirements analysis using CoRE.

1.1 PURPOSE OF THIS REPORT

The purpose of this report is to describe how best to develop a CoRE software requirements specification from an RDD system design model developed using RDD-100. That is, given an RDD-100 model of system requirements and design, this report describes how one can most effectively develop a specification of software requirements using the CoRE method. Cadre's *teamwork* can be used to build a CoRE software requirements specification and is assumed to be the tool of choice for the CoRE practitioner.

An RDD-100 system model contains all of the information needed to begin building a CoRE software requirements specification. This report describes a process and guidelines for building a CoRE software requirements specification from an RDD-100 system model. The process consists of the following activities:

- Use Ascent Logic's Extender to extend the RDD-100 database schema so that it recognizes CoRE elements (optional).

- Given an RDD-100 model of system requirements and design from systems engineering, identify those parts of the model that systems engineers need for CoRE (making use of the extended RDD-100 database schema if possible).

- Generate a report from the RDD-100 model to facilitate mapping to CoRE elements (optional).

- Filter the RDD-100 model into *teamwork* using Ascent Logic's *teamwork* bridge (optional).

- Create an initial CoRE specification from the RDD-100 model (using the generated *teamwork* model and/or CoRE report to facilitate if possible).

- Complete the CoRE specification and begin software design.

Both the CoRE method and RDD-100 tool are continually evolving to include new features and capabilities. This report is based on the *Consortium Requirements Engineering Guidebook*, version 01.00.09 (Software Productivity Consortium 1993) and RDD-100, release 3.0.2 (Ascent Logic Corporation 1992a). Subsequent versions of this report will encompass additional features of enhanced versions of CoRE and RDD-100. Release 4.0 of RDD-100 was delivered during the production of this report.

1.2 BACKGROUND

Figure 1 presents a simplistic, high-level view of the activities performed in developing a system. First, the system requirements are analyzed, and then a system design is created by allocating requirements to hardware and software components. Then, hardware and software development occurs in parallel. This report is only concerned with software development. Software development consists of three activities: requirements analysis, design, and implementation. This report describes the transition from system design to software requirements analysis when both of the following are true:

- RDD-100 has been used to create system requirements and design specifications.

- The software requirements analysis activity is to be performed according to the CoRE method.
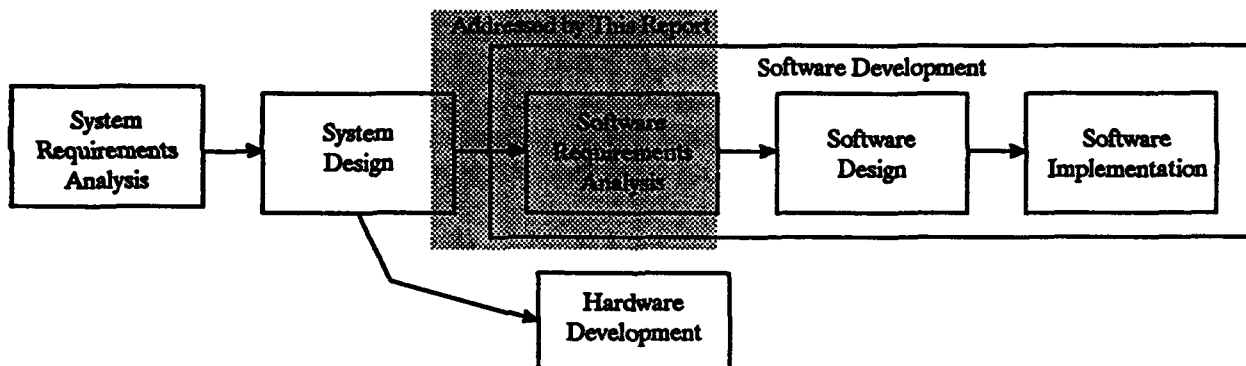
Figure 1. System Development Activities

The shaded region in Figure 1 identifies those activities of system development that are addressed by this report. A small portion of the system design box is shaded because, near the end of system design, some information not traditionally provided by an RDD-100 specification can be provided to ease the transition to software requirements analysis using CoRE. The entire software requirements analysis box is shaded because a significant portion of the CoRE method is affected by the use of an RDD-100 specification.

1.3 QUESTIONS TO BE ANSWERED

This report is intended to provide detailed answers to the following questions:

- Given an RDD-100 specification of system requirements and design, how can an engineer most effectively develop a specification of software requirements using the CoRE method? (An engineer can use the approach shown in Figure 2 and described throughout the remaining sections of the report.)

- Does the RDD-100 system specification contain all the information an engineer needs to build a CoRE specification? (Yes, see Section 3.2.)

- How does an engineer map system requirements expressed in RDD-100 to CoRE? (See Section 3.3.)

- Can the RDD-100 database schema be extended to include the CoRE schema? (Yes, however, Ascent Logic's Abstract Object Editor provides better support for the CoRE schema, as described in Section 4.)

- Can a CoRE specific report be generated using RDD-100's Report Writer that will facilitate the development of a CoRE specification from an RDD-100 model? (Yes, as described in Section 5.)

- Can the RDD-100 bridge to teamwork support an automated transition from an RDD-100 system design to a CoRE software requirements activity using teamwork? (Yes, as described in Section 6.)

This report answers these questions and includes a supporting example.

1.4 INTENDED AUDIENCE

This report is intended to guide software requirements engineers in developing a CoRE specification from an RDD-100 system specification. The systems engineer using RDD-100 to develop a system design can use this report to help identify the kinds of information that a software requirements engineer using CoRE would expect to find in the RDD-100 model, thus allowing smoother transition from system design to software requirements and design. This report can be used by process and method developers to help describe an engineering process, including both the RDD-100 tool and the CoRE method.

It is assumed that readers of this report have a fundamental understanding of RDD-100 and CoRE. Although this report contains brief overviews of RDD-100 and CoRE, readers should be familiar with *RDD-100 User's Guide* (Ascent Logic Corporation 1992a) and *Consortium Requirements Engineering Guidebook* (Software Productivity Consortium 1993) before reading this report.

1.5 ORGANIZATION OF THIS REPORT

The remainder of the report is organized as follows:

- Section 2 provides overviews of RDD-100 and CoRE and compares and contrasts them.

- Section 3 describes an approach for deriving a CoRE software requirements specification from an RDD-100 system design specification. It describes where to look in the RDD-100 model for information needed by CoRE and describes how to build the CoRE model from the RDD-100 specification.

- Section 4 describes how systems engineers might extend the standard, underlying database schema of RDD-100 so that it recognizes CoRE constructs and, therefore, facilitates mapping from an RDD-100 model to a CoRE model.

- Section 5 describes a CoRE specific report that can be generated using RDD-100's Report Writer to facilitate the development of a CoRE specification from an RDD-100 model.

- Section 6 describes how to use the RDD-100 bridge to Cadre's **team***work*/RT tool for those using **team***work*/RT to build a CoRE specification.

- The Appendix contains selected examples from the Host-at-Sea (HAS) Buoy system case study that illustrate the approach described in this report.

1.6 TYPOGRAPHIC CONVENTIONS

This report uses the following typographic conventions:

Serif font . . . . General presentation of information.

Italicized serif font . . . . Publication titles and, in the Appendix, element relationships and attributes.

Boldfaced serif font . . . . Section headings and emphasis.

Boldfaced italicized serif font . . . . Run-in headings in bulleted lists and, in the Appendix, minor subsections.

Italicized sans serif font . . . . RDD-100 element names.

`Typewriter` font . . . . Syntax of code or software responses.

2. OVERVIEW

This section provides an overview of the process and framework upon which the remainder of the report is based. The proposed process for developing a CoRE specification from an RDD-100 specification is composed of the following activities:

- Use Ascent Logic's Extender to extend the RDD-100 database schema so that it recognizes CoRE elements (optional).

- Given an RDD-100 model of system requirements and design from systems engineering, identify those parts of the model that systems engineers need for CoRE (making use of the extended RDD-100 database schema if possible).

- Generate a report from the RDD-100 model to facilitate mapping to CoRE elements (optional).

- Filter the RDD-100 model into teamwork using Ascent Logic's teamwork bridge (optional).

- Create an initial CoRE specification from the RDD-100 model (using the generated teamwork model and/or CoRE report to facilitate if possible).

- Complete the CoRE specification and begin software design.

Figure 2 illustrates the proposed process and shows how it fits into the system development process illustrated in Figure 1. Boxes represent activities, and arrows indicate sequencing; iteration is implicit and is not shown. External activities are those performed as usual, regardless of the use of the proposed process (although some assumptions must be made about the RDD-100 system design model, as described in Section 3.1). Required and optional activities are those activities performed specifically when building a CoRE specification from an RDD-100 system design model, as described in this report.

Although Figure 2 does not illustrate it, the process is iterative, meaning that the sequence of activities may be repetitive (i.e., all arrows in Figure 2 are really bidirectional). For example, after building an initial CoRE specification, systems engineers might decide to revisit the Identify CoRE Inputs activity because of the acquisition of new or clarifying information.

The remainder of this section provides context for the remainder of this report. Sections 2.1 and 2.2 describe the relevant features of RDD-100 and CoRE, respectively. Section 2.3 compares and contrasts RDD-100 and CoRE.

2.1 RDD-100

RDD-100 is the implementation of the RDD system design and engineering method (Ascent Logic Corporation 1992a). RDD encompasses both an empirically derived method for designing systems
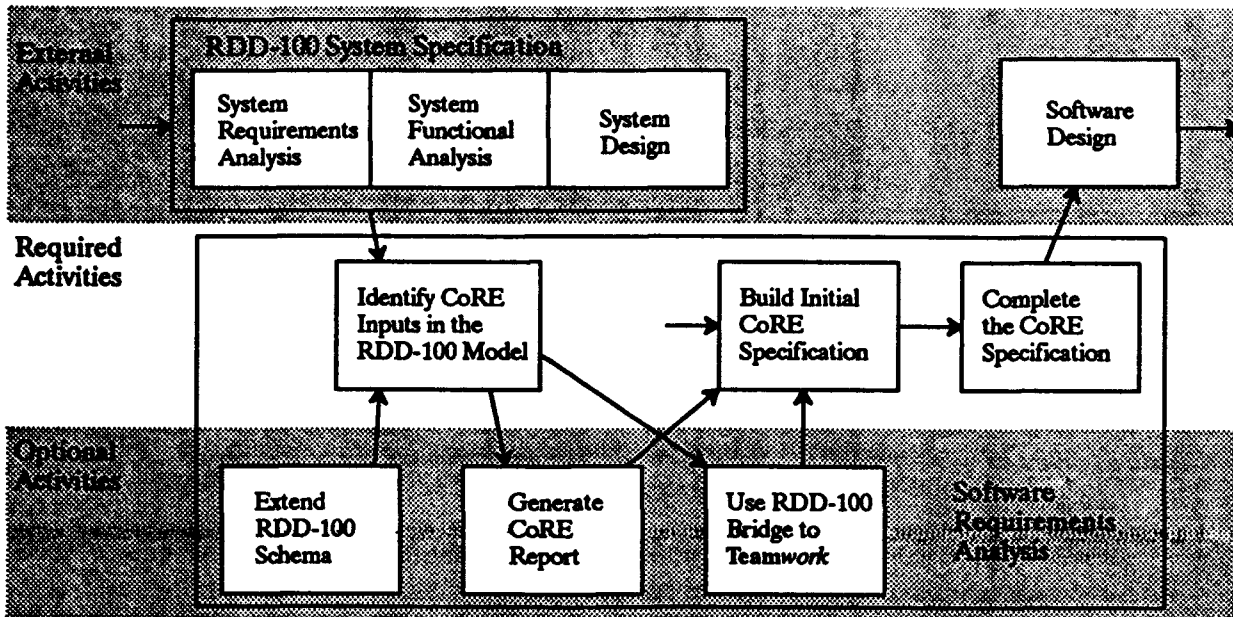
Figure 2. The Proposed Process

and the rigorous use of a structured, executable language that implements an entity-relationship-attribute (ERA) data model for the systems engineering database. Because of the generality allowed by the ERA data model, RDD-100 is equally at home whether used to model an enterprise, a manufacturing process, a large communications system, a subsystem, or relatively low-level behaviors, such as communications protocols. RDD-100 has implemented the ERA model using an object-oriented database, which contains only one instance of the system's descriptive data and provides multiple views of the data for the design and analysis activities.

Automated components of RDD-100 that are of particular interest include (O'Rourke 1993):

- *Element Editor.* Provides an interactive mechanism for specifying and traversing RDD-100 database elements, their attributes, and relationships between them.

- *Requirements Extractor.* Facilitates the specification of a system requirement hierarchy by extracting portions of requirements documents.

- *Graphics Editor.* Permits engineers to describe the dynamic properties of systems using a graphics language describing behavior in terms of the time sequences of inputs, functions, and outputs.

- *Dynamic Verification Facility.* Provides dynamic execution of the system specification as a discrete event simulation from within the systems engineering database.

- *Extender.* Allows the user to extend the underlying ERA database schema of RDD-100 to accommodate more specialized needs (e.g., the engineering change proposal process or the proposal development process).

- *Report Writer.* Allows the generation of predefined or tailored reports from the RDD-100 database.

- *Teamwork Bridge.* Automatically generates a **teamwork** model from an RDD-100 database.

This section provides a high-level description of RDD-100's systems engineering method (see Section 2.1.1) and data model (see Section 2.1.2) to establish a framework for subsequent sections.

Release 4.0 of RDD-100 provides additional tools for manipulating the contents of the database, including a multielement editor and abstract object modeling. Section 2.1.3 provides a brief overview of how those capabilities may be useful for CoRE specifications. However, the details of Release 4.0 capabilities are beyond the scope of this paper.

2.1.1 RDD-100 METHOD

RDD, the method underlying the RDD-100 tool, is aimed at finding a way to develop high-performance, high-reliability systems composed of hardware, software, and people. O'Rourke (1993) describes the RDD-100 method as a process wherein the system design is driven by the customer's expectations of a system behavior, which is traced directly to system requirements and their original source. The system design is expressed as an allocation of system behaviors (functions) onto the various components that will compose the implemented system.

O'Rourke (1993) describes the RDD-100 method as the following set of activities:

- Define the engineering problem and system boundaries.

- Define a candidate component architecture.

- Extract and index requirements; describe desired behavior.

- Decompose and allocate behavior to components (i.e., hardware, software, people, etc.).

- Identify and specify interfaces.

- Perform feasibility and tradeoff analysis.

- Describe failure mode behavior.

- Plan system integration and test.

- Optimize design.

- Generate specifications and documentation.

- Perform verification and validation against expected behavior.

In the context of this report, the first three activities in the above list are considered system requirements activities. The remaining activities are part of the system design process.

2.1.2 RDD-100 MODEL

The RDD-100 model includes two kinds of elements: those that are part of the system requirements model and those that are part of the system design model. RDD-100 system requirements elements of interest for this report include the following (Ascent Logic Corporation 1992a, 1993a):

- *Source* records the paper input to the system design process.

- *SystemRequirement* is a detailed functional, performance, and interface requirement derived over the life of the system specification process.

- *CriticalIssue* is a problem, issue, or limitation.

- *Decision* is a choice that has been made to establish requirements based on *SystemRequirements*.

- *PerformanceIndex* is a quantifiable performance limitation or objective.

- *Constraint* is a required quality or resource limitation on other elements.

- *SystemParameter* is an operating parameter that may affect cost and/or performance during development and/or operation.

Figure 3, derived from Ascent Logic Corporation (1993a, 1.5–1.13), illustrates the fundamental elements and relationships underlying the RDD-100 model of system requirements. All of the relationships in Figure 3 are bidirectional: arrowheads are used to identify the directions implied by the relationship names.
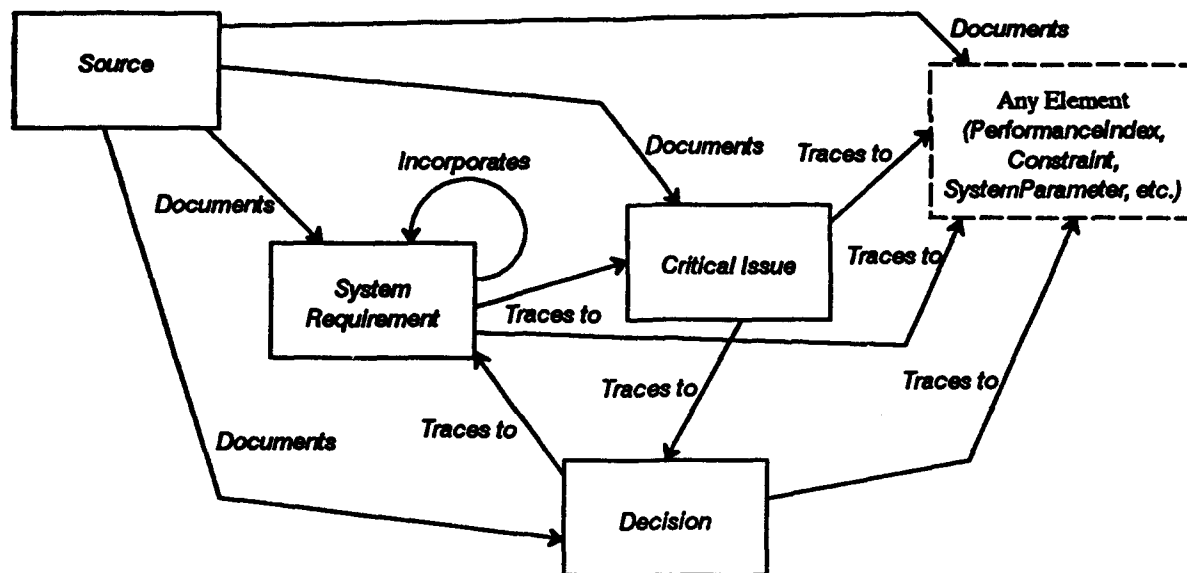
Figure 3. The RDD-100 System Requirements Model

RDD-100 system design elements of interest for this report include the following:

- Items are model-observable inputs and outputs that arrive at or depart from the system.

  - *DiscreteItem* arrives as a unit, represents the limit of observability.

  - *TimeItem* is a class of legal sequences of *DiscreteItems*.

- Functions transform arriving items into departing items.

  - *DiscreteFunction* transforms one *DiscreteItem* into an unordered collection of *DiscreteItems.*

  - *TimeFunction* is an aggregation of *DiscreteFunctions* or *TimeFunctions* that transforms an ordered collection of *DiscreteItems* or *TimeItems* into an ordered collection of *DiscreteItems.*

- *Component* is one of the parts (hardware, software, or human) in a system, for example, a subsystem.

- Graphic constructs represent concurrency, iteration, loops, conditions, selection, and replication.

  - *INet* represents sequences of inputs or outputs.

  - *FNet* represents sequences of behavior (functions).

- *ItemLink* is a logical pathway that carries a message item from one *RDDProcess* to another.

- *ExternalSystem* is a separate system outside the required system's boundary.

- *Interface* is a mechanism for items to flow across the system boundary or from one component to another.

Figure 4 illustrates the fundamental elements and relationships underlying the RDD-100 model of system design. The legend identifies the names of the relationships that identify how the source elements on the left side of Figure 4 relate to the target elements on the right side of Figure 4.

2.1.3 RDD-100 RELEASE 4

The following new products from Release 4 of RDD-100 may provide improved support for CoRE:

- *Multi-Element View.* Provides the ability to view and edit multiple RDD-100 database elements and relationships in a single window. Among other advantages, this tool should facilitate decomposition of *Source* documents into hierarchies of *SystemRequirements.*

- *Abstract Object Editor (and Real World Object Editor).* Provides the ability to overlay a database schema on top of the existing one. This tool allows the user to tailor an RDD-100 model for methods such as CoRE and obviates the need to modify the RDD-100 database schema, as described in Section 4.

A future release of this report will discuss Release 4.0 of RDD-100 in detail.

2.2 CoRE

CoRE is a method for analyzing, capturing, and specifying software requirements (Software Productivity Consortium 1993). The Consortium has worked with industrial developers of real-time and embedded systems to provide a method that addresses their needs. CoRE supports the
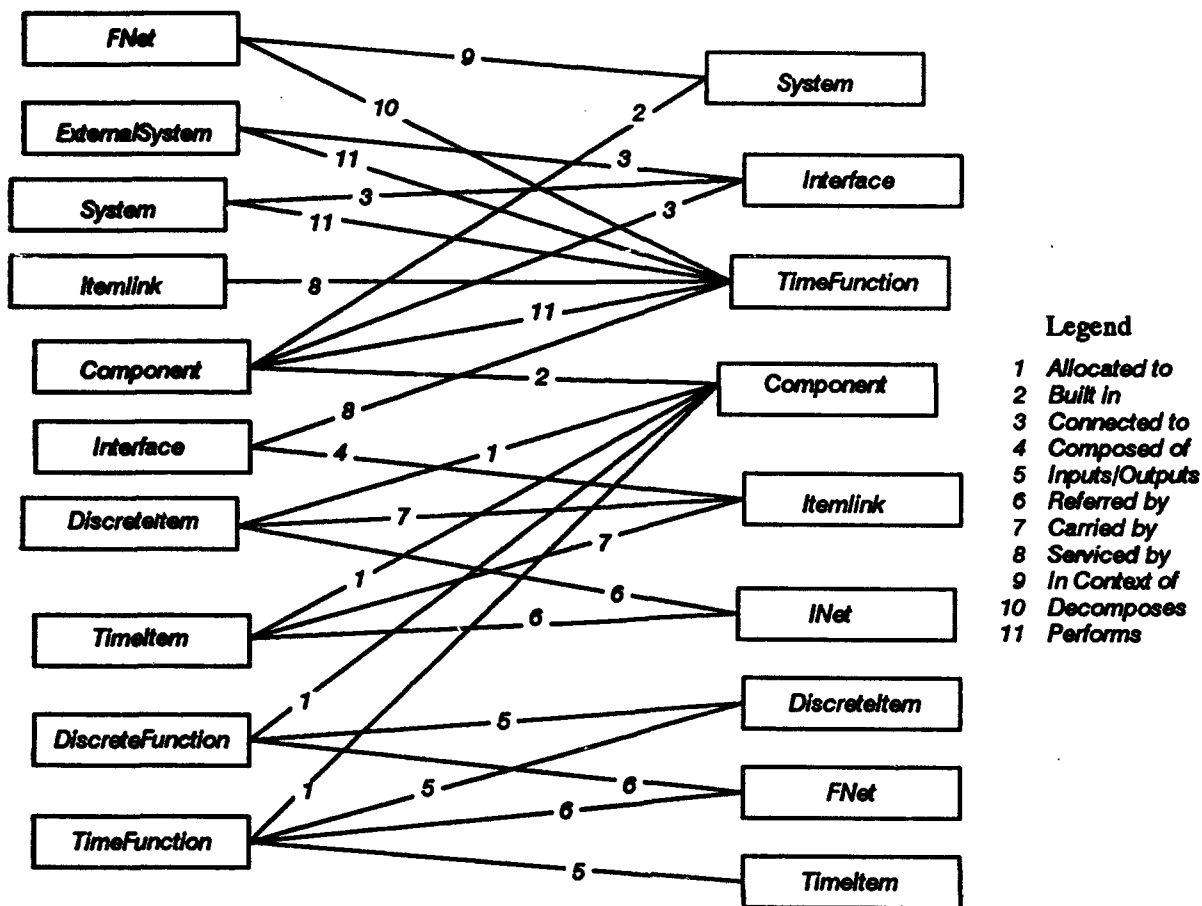
Figure 4. The RDD-100 System Design Model

development of precise testable specifications that are demonstrably complete and consistent. CoRE also supports key process issues, such as managing changing requirements and reuse. CoRE is a single, coherent requirements method that:

- *Integrates Object-Oriented and Formal Models.* A CoRE specification organizes the details of the behavioral model into classes of objects, which provides a mechanism for abstraction, separation of concerns, and information hiding. Systems engineers use the CoRE class structure to address objectives like change management or reuse.

- *Integrates Graphical and Rigorous Specifications.* Graphic representation helps all parties (e.g., customers, engineers, designers, and programmers) to grasp essential relationships among system components. CoRE provides a consistent, rigorous interpretation of both graphical and mathematical notations. This allows the graphical specifications to combine smoothly with the detailed specifications that are best given in mathematical and textual notation.

- *Uses Existing Skills and Notations.* The language used to specify requirements in CoRE is based on familiar concepts and existing notations.

- *Permits Nonalgorithmic Specification.* CoRE is nonalgorithmic in the sense that systems engineers can specify the required behavior of a system without having to provide an algorithm

or detailed design; i.e., systems engineers can always specify the behavior in terms of what the system must do rather than how it does it.

- *Provides Guidance.* The CoRE process model provides practical guidance in developing both the object structure and the behavioral requirements. The behavioral model provides a standardized structure that helps the developer determine the class structure. The behavioral model also forms the basis of a systematic process for developing a complete requirements specification.

Requirements in CoRE are written in terms of two underlying models: the behavioral model and the class model. The CoRE process describes the order in which the specification is composed, the behavioral model captures what the software must do, and the class model organizes that information. Section 2.2.1 provides an overview of the CoRE process, Section 2.2.2 describes the behavioral model, and Section 2.2.3 describes the class model.

2.2.1 CoRE Process Overview

The CoRE process is a sequence of activities that systems engineers follow to develop a CoRE requirements specification. The CoRE process is driven by two concerns. The first concern is the step-by-step construction of a required behavior specification in terms of the CoRE behavioral model for a particular system. The goal is to develop a complete and consistent description of the required behavior. The second concern is the step-by-step packaging of specification pieces in elements of the class structure. This aspect of the method satisfies packaging goals, such as change management and reuse. Because packaging and specification activities overlap in time, the threads of these activities are intertwined in the CoRE method.

The input to the CoRE process is some form of system requirements specification (i.e., the first activity, identifying system constraints, assumes that a system specification is available). The output of the CoRE process is a complete specification of the software requirements (i.e., suitable for a software design process).

The CoRE process is a description of an "ideal" process. The process is idealized rather than "real" in that it does not account for errors, requirements changes, unknown requirements, or other factors requiring additional iteration, experimentation, or backtracking. An ideal process is useful because it provides an external standard to guide development and it serves as a yardstick for measuring progress. Thus, the ideal CoRE process is divided into a sequence of five activities:

- *Identify Environmental Variables.* Systems engineers identify candidate environmental variables and the relations among them. The overall goal is to identify environmental quantities that denote the monitored and controlled variables, relationships that will become parts of the required (REQ) and natural (NAT) relations, and relationships that will become part of the generalization/specialization structure. Identify likely changes and their impacts on these environmental variables.

- *Preliminary Behavior Specification.* Systems engineers identify and specify the monitored and controlled variables. They identify undesired events to which the system must respond and define monitored variables to denote them. They identify the domain and scheduling type for each controlled variable and identify modes.

- **Class Structuring.** Systems engineers create a class structure to address their packaging goals. They decide how the parts of the behavioral model will be allocated among CoRE classes. They create boundary, mode, and term classes based on their packaging goals. They define the class interfaces and identify class dependencies.

- **Detailed Behavior Specification.** Systems engineers complete the class definitions by completing the specification of the controlled variable functions and timing constraints for each controlled variable. They refine the class structure to be consistent with the behavioral model needs.

- **Define Hardware Interface.** Systems engineers define the system inputs and outputs and define the input (IN) and output (OUT) relations.

Figure 5 (from Figure 6-1 of Software Productivity Consortium 1993) illustrates the five activities of the CoRE process along with their corresponding work products. Those work products and activities whose development is affected by the use of an RDD-100 system design model are shaded.
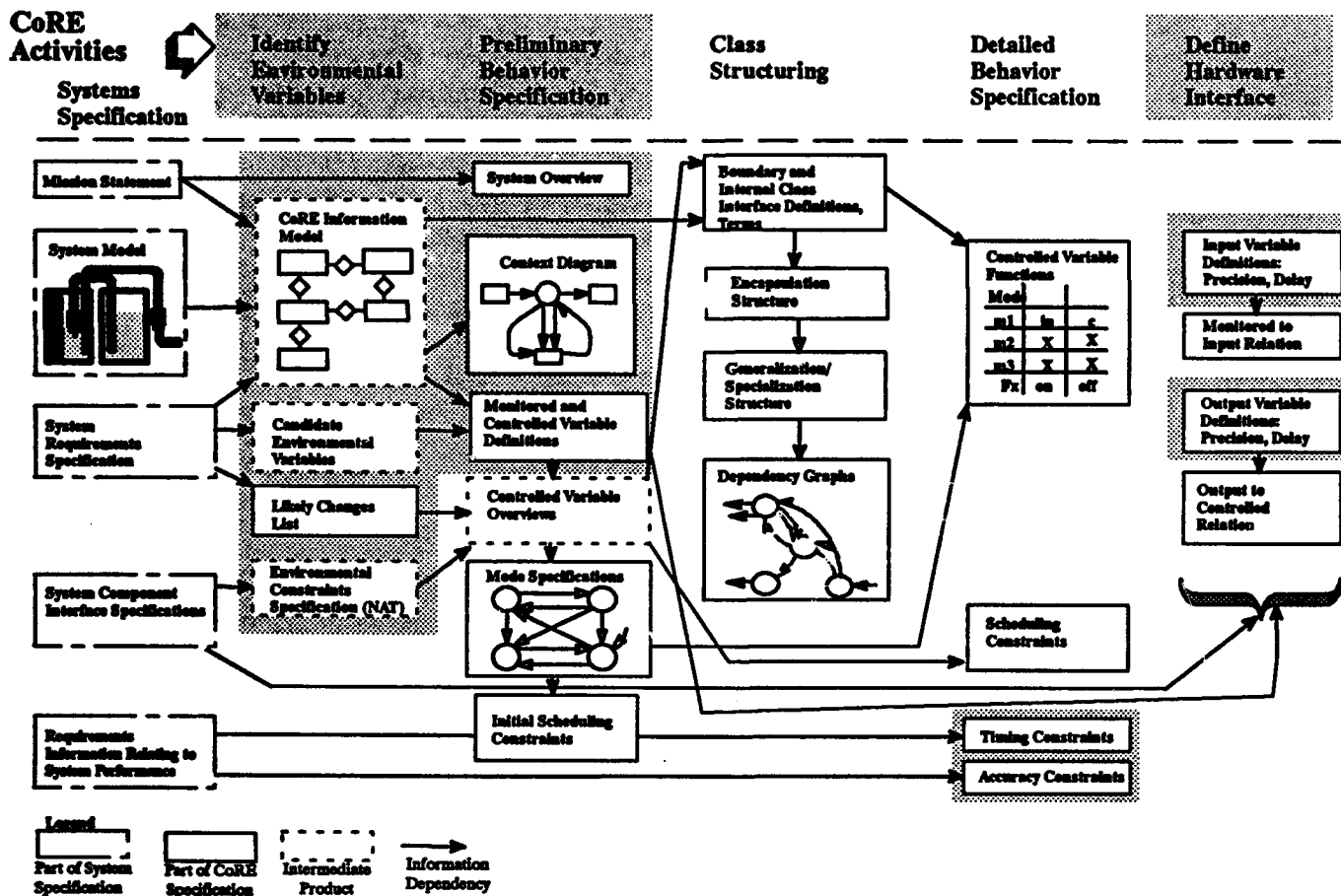
Figure 5. The CoRE Process

2.2.2 CoRE Behavioral Model

The CoRE behavioral model provides a standard formal model for specifying the required behavior of an embedded system. The behavioral model represents the semantics of the requirements specification. Figure 6 illustrates the underlying behavioral model of CoRE.

This section is divided into three subsections, each of which describes a part of the behavioral model that is of particular interest for this report: environmental variables (Section 2.2.2.1), input and output variables (Section 2.2.2.2), and four-variable relations (Section 2.2.2.3).
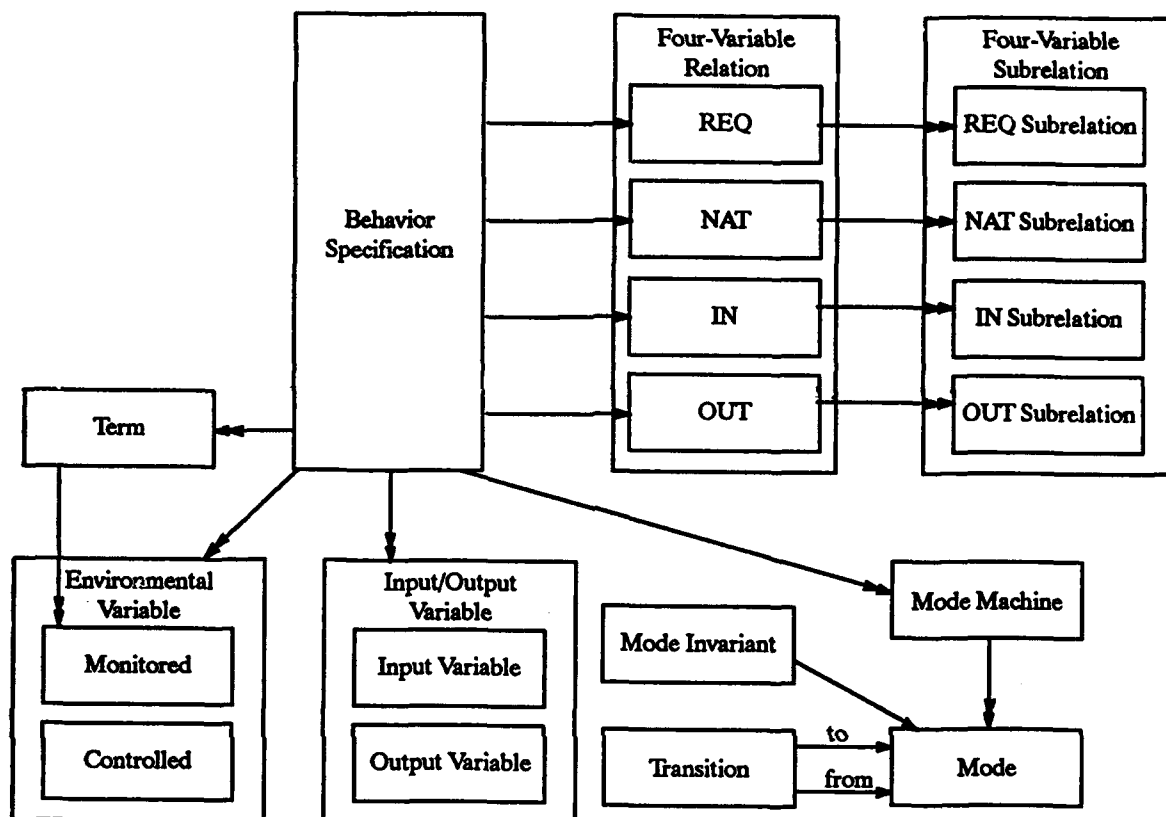
Figure 6. The CoRE Software Requirements Behavioral Model

2.2.2.1 Environmental Variables

Environmental variables are physical quantities of interest in the environment of a system. For example, air pressure is of interest to an automotive engine-control system. There are two kinds of environmental variables:

- *Monitored Variables.* Environmental quantities the system must track (e.g., the ambient air pressure).

- *Controlled Variables.* Environmental quantities the system sets (e.g., the fuel flow to the cylinders).

2.2.2.2 Input and Output Variables

Input and output variables are variables representing discrete inputs or outputs of the software. The complete definition of an input (output) variable describes precisely how the software reads from (writes to) a device, including the protocol for reading from (writing to) a device and a mapping between abstract values and the bit patterns read from (written to) the device.

2.2.2.3 Four-Variable Relations

Four-variable relations contain ordered pairs of environmental variables and input and output variables. There are four kinds of relations in the CoRE behavioral model:

- *NAT.* NAT specifies the external constraints on the values that the environmental variables can assume. These constraints are properties of the environment that affect the software but exist independently of the software.

- *REQ.* REQ specifies properties that the system is required to maintain between monitored and controlled variables. The REQ relation is the fundamental means of specifying behavioral requirements with CoRE.

- *IN.* IN expresses values taken on by the monitored variables as a function of bit settings or other low-level hardware settings (input variables).

- *OUT.* OUT specifies values of controlled variables as a function of the values of output variables.

Figure 7 illustrates how these variables are related by the four kinds of relations.
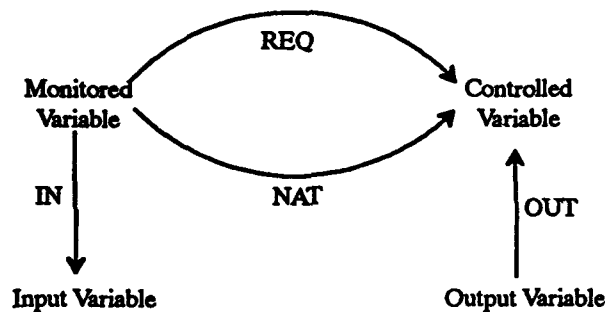
Figure 7. The CoRE Four-Variable Model

2.2.3 CoRE CLASS MODEL

The CoRE class model provides a set of facilities for packaging the information in a CoRE specification. The class model allows you to divide the specification into relatively independent parts and to control the relationships between parts. The class model is not intended to imply any requirements about the behavior of the software.

The information in the four-variable model is partitioned among a set of CoRE classes. A CoRE class is a template for defining a subclass or object (conversely, an object is always an instance of a class). Systems engineers determine such characteristics as the number of classes, what information is hidden by each class, and which parts of the model are allocated to the same class based on their overall goals for the requirements structure. There are three kinds of CoRE classes:

- *Boundary Classes.* Contain the definitions of the system's monitored and controlled variables.

- *Mode Classes.* Encapsulate mode machine definitions and provide mode information.

- *Term Classes.* Provide any terms (named expressions of monitored variables) not provided by the boundary and mode classes.

CoRE's class model exists with quite different motivations than those of design and implementation methods. CoRE deals with objects and classes as buckets for containing interesting parts for the sake of designing the specification rather than the system. Designing a specification with classes differs from designing a system for object-oriented implementation in the following ways:

- The CoRE class structure is part of the requirements specification, intended to facilitate packaging; it is not part of the system itself. It may be that a different class structure will be applied to the system during implementation.

- Although CoRE recognizes an inheritance from superclasses, the idea of class structure for encapsulating (hiding) information and the depends-on relationship between specification objects are much more important.

The inheritance relationship is a dominant feature of the structure of programs to be built with object-oriented implementation languages primarily because of the overwhelming importance of reusing code from superclasses. Code reuse and, consequently, the inheritance relationship are not typically critical concerns during requirements specification.

The classes in a CoRE specification are best viewed as "cell walls" created to contain:

- Encapsulated information (secrets)

- Parts of the system that are likely to change, as opposed to those prone to remain stable

2.3 COMPARING RDD-100 AND CoRE

RDD-100 supports system requirements analysis and design, where a system is made up of hardware, software, and people. CoRE supports software requirements analysis. For CoRE to be applied to a software system that is part of a larger system specified using RDD-100, the RDD-100 model must (and does) contain all of the information needed to build a CoRE specification. Of course, the CoRE requirements writer must know which parts of the RDD-100 model are useful for CoRE and which are not.

The earliest activity in the RDD-100 process consists of decomposing system requirements documents into hierarchies of individual system requirements. The subset of these system requirements related to the software subsystem being specified using CoRE along with any related RDD-100 database elements derived from them provides the basis for a CoRE specification.

RDD-100 encourages the systems engineer to define the environment in which the proposed system will operate. The environment includes those external systems and environmental entities that have an effect upon the system and those variables that are transferred between the system and the environment or external system. This information is essential to a CoRE specification, which specifies the required relationships between variables that are monitored by the software system and those that are controlled. RDD-100 captures this information using behavior diagrams and provides automatic translation to the context diagram format recognized by CoRE.

CoRE provides powerful techniques for nonalgorithmic expressions. CoRE's focus is on tabular mathematical descriptions of actions as a function of events, conditions, and modes. RDD-100 does not currently support this notation; it allows specification of behavior using flowlike descriptions for

timing and sequencing (behavior diagrams). RDD-100 was not intended to and does not provide the tools necessary to build a CoRE software requirements specification.

RDD-100 offers robust modeling features for analyzing a system design. The system design is specified using a hierarchy of behavior diagrams. Making a design analyzable requires the system designer to include decisions related to logic, sequencing, or concurrency in the behavior diagrams. From the CoRE perspective, these kinds of decisions may be considered design decisions that should be avoided during software requirements analysis. As a result, a completed RDD-100 specification, composed via the RDD methodology, may have more information than is necessary to build a CoRE specification. More precisely, it is likely that only the highest level behavior diagrams in the RDD-100 model's hierarchy will be needed for CoRE.

In summary, an RDD-100 system model does not contain all of the information that a CoRE software requirements specification contains. There is an inherent difference in the level and amount of detail between system and software specifications. The software specification contains requirements that are derived from the system specification. A complete CoRE specification cannot be automatically generated from an RDD-100 specification. However, RDD-100 is a suitable starting point for CoRE because it contains all of the information needed to begin CoRE. The CoRE requirements writer must know where in an RDD-100 model to find relevant pieces of information and how to build a CoRE specification from them. The best use of RDD-100 with CoRE is to build the CoRE specification with a more appropriate tool, such as teamwork, and to make use of the facilities RDD-100 provides for transitioning from system design using RDD-100 to software requirements using teamwork (and CoRE).

3. AN APPROACH FOR DERIVING A CoRE MODEL FROM AN RDD-100 MODEL

This section describes an approach, including the process, guidelines, and examples, for deriving a CoRE software requirements model from an RDD-100 system requirements and design model. The activities involved in this approach include:

- Building the RDD-100 model (Section 3.1)

- Identifying CoRE inputs in the RDD-100 model (Section 3.2)

- Mapping the information contained in an RDD-100 model to an initial CoRE specification (Section 3.3)

- Completing the CoRE specification (Section 3.4)

This set of activities is not intended to be strictly sequential; iteration is expected.

3.1 BUILDING THE RDD-100 MODEL

To describe an approach for deriving a CoRE model from an RDD-100 model, it is necessary to make some assumptions about what is contained in the RDD-100 model. This section describes the assumed process for developing a system requirements model using RDD-100. The intention is to provide the least number of constraints possible so that the approach is applicable to the widest possible range of established RDD-100 users.

The assumed process includes a subset of those activities described in Alford (n.d.). The RDD-100 model should not be built any differently than usual, but for the purposes of CoRE, it is assumed that a minimal set of activities has been performed. The assumed process for developing a system requirements model using RDD-100 includes:

- System requirements analysis (Section 3.1.1)

- System functional analysis (Section 3.1.2)

- System design (Section 3.1.3)

Section 3.1.4 offers some suggested tactics for building the RDD-100 model for those systems engineers who know beforehand that CoRE will be used to specify software requirements.

3.1.1 RDD-100 SYSTEM REQUIREMENTS ANALYSIS

The RDD-100 system requirements analysis activity begins by identifying individual system requirements statements from requirements documents (e.g., a mission statement). Each such

requirements document is identified in the RDD-100 database by creating an associated element of type *Source*. The RDD-100 requirements extractor can be used to parse a textual document and to create a hierarchy of individual system requirements in the database, known as *SystemRequirement* elements. The resulting relationships between elements are:

- *Documents* relationships are created between *Source* elements and the *SystemRequirement* elements they contain.

- *Incorporates* relationships are created between *SystemRequirement* elements and their child *SystemRequirement* elements in the hierarchy.

CriticalIssue elements are created for recording technical issues critical to the successful development of the system. *Documents* relationships relate *Source* elements to *CriticalIssue* elements, and *TracesTo* relationships record their traceability from *SystemRequirement* elements. When critical issues are resolved, their resolutions are recorded in *Decision* elements, and *TracesTo* relationships are used to relate these element pairs. *Decision* elements may result in the creation of additional *SystemRequirement* elements, and *TracesTo* relationships are also used to relate these element pairs. *Documents* relationships may also relate *Source* elements to *Decision* elements. Figure 3 illustrates the elements and relationships supporting the RDD-100 system requirements analysis activity.

For the HAS Buoy case study described the Appendix, there were two *Source* elements: the HAS Buoy problem statement (see Section App.1) and a list of requirements stabilities and variabilities (see Section App.1.6). Section App.2.2 documents the hierarchy of *SystemRequirement* elements. *CriticalIssue* and *Decision* elements are described in Section App.2.4.

3.1.2 RDD-100 SYSTEM FUNCTIONAL ANALYSIS

In the RDD-100 system functional analysis activity, systems engineers develop a functional model that reflects the functional requirements of the system (Alford n.d.). The goal is to develop a functional model that represents the desired behavior of the system. During this activity, systems engineers will define how the system will logically operate and provide a basis of how the allocated design must behave. This activity consists of identifying system subsets called *Components*, whose behaviors are specified using behavior diagrams.

Begin the RDD-100 system functional analysis activity by decomposing the system into *Components* as follows:

- Create a *Component* element of type *System* representing the entire specification, with the name of your system (*Components* of a particular type are created by setting the *Component Type* attribute of the *Component* accordingly).

- Create *Component* elements of type *ExternalSystem* representing systems external to your own with which your system must interact.

- Create *Component* elements of type *Environment* representing entities in the environment that have an effect on your system.

- Create *Component* elements of other types (e.g., *Subsystem*, *CSCI*, etc.) representing internal parts of the system such that all *SystemRequirement* elements have been mapped to a

Component element (either directly by the *TracesTo* relationship or indirectly through the functional model).

Specify the behaviors of these *Components* as follows:

- Specify the behavior of the *System Component* element by creating a behavior diagram (*FNet*) that contains the functions performed by all related *ExternalSystem*, *Environment*, and other *Components*.

- Create a *HasContext* relationship between the *System Component* element and the *FNet* element.

- Specify interactions between *Components* using elements of type *TimeItem*, *DiscreteItem*, or *Interface*.

- Describe the precise behavior of an individual *Component* element by decomposing and allocating functionality to *TimeFunction* and *DiscreteFunction* elements.

- Specify timing requirements in the "duration" attributes of *Functions* for subsequent modeling. Create *PerformanceIndex* elements as necessary to capture timing requirements.

- Create additional *CriticalIssue* and *Decision* elements during this activity as you realize their need.

The remaining, lower level details of the behavior diagrams are completed such that sufficient detail is provided to allow RDD-100 to execute the behavior diagram using the Dynamic Verification Facility. These lower level details are considered design by CoRE: although some may imply constraints upon the software requirements, others may not be used during the application of CoRE.

Section App.2.7 of the HAS Buoy case study describes the *Component* elements that were identified and shows the hierarchy graphically.

3.1.3 RDD-100 System Design

System design is the final activity of the RDD-100 process. The RDD-100 system design activity consists of allocating the system behavior (functionality) to the system architecture (Alford n.d.). The engineer should consider several allocation strategies, which can be evaluated using the Dynamic Verification Facility.

System design using RDD-100 is performed by creating *AllocatedTo* relationships between *DiscreteFunction* elements and *Component* elements. *DiscreteFunction* elements are aggregated to reflect the behavior of the *Component* (i.e., show inputs, outputs, and the logic the *Component* is to perform).

Section App.2.7 identifies the *AllocatedTo* relationships that were identified for the HAS Buoy case study.

3.1.4 TACTICS THAT SUPPORT CoRE

This section offers some suggested tactics for building the RDD-100 model for those systems engineers who know beforehand that CoRE will be used to specify software requirements:

- Use techniques that allow you to isolate those parts of the RDD-100 model that are likely to map to CoRE elements (i.e., those RDD-100 elements identified in Sections 3.1.1 through 3.1.3). For example, use naming conventions for identifying those RDD-100 elements that are of particular interest to CoRE (e.g., *Components*, *DiscreteItems*, etc.). In the HAS Buoy example, the names of RDD-100 elements of interest to CoRE were capitalized, while those that were not of interest were stated in lower case.

- When using RDD-100 and creating elements that will later become part of a CoRE specification (e.g., monitored variables, terms, etc.), make sure that these elements are not used in ways contrary to the use of CoRE. For example, perform modeled manipulations on *DiscreteItems* that you expect to become monitored variables via a series of intermediate steps so that you can represent these manipulations as CoRE terms, and store them in relevant problem classes.

- Make the behavior diagram corresponding to the CoRE context diagram executable, and avoid any more detail than is necessary to do so. RDD-100's DVF is a useful tool for evaluating a system design. However, DVF leads you to specify algorithmic *FNets*, which are likely to provide more detail than is necessary for CoRE.

- When using the element editor, specify as much CoRE-relevant information as possible when recording the textual templates associated with elements (e.g., always specify the *Description* attribute of elements). Also, make good use of the consistency checking, particularly the "fundamental" and "system engineering" levels of checking.

An RDD-100 model is generally useful for CoRE, and the remainder of this report is based on the assumption that the systems engineer using RDD-100 was not aware of the intent to subsequently create a CoRE specification. However, if the systems engineer using RDD-100 is aware of a subsequent CoRE specification, the tactics described in this section should facilitate transition from RDD to CoRE.

3.2 IDENTIFYING CoRE INPUTS IN THE RDD-100 MODEL

The RDD-100 system model should contain all of the information needed to begin building a CoRE software specification. In fact, the RDD-100 model is likely to contain more information than is needed for CoRE. In any case, the systems engineer should verify that the RDD-100 model contains the necessary inputs to CoRE.

As Figure 5 shows, the necessary inputs to CoRE are: mission statement, system model, system requirements specification, system component interface specifications, and requirements information relating to system performance. This section is divided into five subsections describing how each of these inputs might be recorded in an RDD-100 model. Table 1 summarizes the mapping from RDD-100 schema elements to CoRE inputs and identifies where to locate candidate RDD-100 schema elements in the RDD-100 System Engineering Notebook (SEN).

Table 1. CoRE Inputs in the RDD-100 Model

CoRE Inputs	Candidate RDD-100 Schema Element(s)	Where Found (SEN Chapter)
Mission statement	Source	External to RDD-100—input documents
System model	FNet (system context diagram) Component	FNet: context diagram is Figure 1-1 Components: Chapter 1, "System Top-Level Description"
System requirements specification	SystemRequirement	Chapter 2, "System-Level Operating Requirements"
System component interface specifications	Interface ItemLink DiscreteItem TimeItem	Chapter 10, "Interfaces Between Components"
Requirements information relating to system performance	PerformanceIndex	Chapter 7, "Performance Indices"

3.2.1 MISSION STATEMENT

The mission statement is a high-level description of system requirements. The CoRE requirements writer should look at RDD-100 elements of type Source to find the mission statement.

For the HAS Buoy case study, the equivalent of the mission statement was the HAS Ada-based Design Approach for Real-Time Systems (ADARTS®) Problem Statement in Section App.1, which was identified in the RDD-100 database by a Source element.

3.2.2 SYSTEM MODEL

The system model is a functional description of the behavior of the proposed system. The CoRE requirements writer should look at the RDD-100 behavior diagram (FNet) that models the behavior of the entire system to find the system model. In the RDD-100 model, a Component element of type System, named appropriately, should be related to an FNet by a HasContext relationship. This FNet models the behavior of the entire system.

Figure 8 shows the system model for the HAS Buoy case study. It is the behavioral model representing the System Component element.

3.2.3 SYSTEM REQUIREMENTS SPECIFICATION

The system requirements specification is a detailed description of system requirements. The CoRE requirements writer should look at the RDD-100 hierarchy of SystemRequirement elements to find the requirements that make up the system requirements specification. RDD-100's Report Writer provides the capability to automatically generate reports, such as the system requirements specification, using a variety of templates, including MIL-STD-490A, DOD-STD-2167A, or user-defined specifications.

The hierarchy of SystemRequirement elements (see Section App.2.2) or the entire RDD-100-generated SEN (see Section App.2) could have served as the system requirements specification for the HAS Buoy case study.

3.2.4 SYSTEM COMPONENT INTERFACE SPECIFICATIONS

System component interface specifications describe the interfaces between the subsystems in a system and between the system and its environment. The CoRE requirements writer should look at RDD-100 *TimeItem*, *DiscreteItem*, *ItemLink*, and *Interface* elements to find system component interface specifications. Of particular interest are those elements that are shared by the *System Component* element (see Section 3.2.2) and other *Component* elements.

Sections App.2.6 and App.2.8 of the HAS Buoy case study identify *TimeItem, DiscreteItem, ItemLink,* and *Interface* elements that may be included in the system component interface specification.

3.2.5 REQUIREMENTS INFORMATION RELATING TO SYSTEM PERFORMANCE

Requirements information relating to system performance typically specify end-to-end system timing requirements. The CoRE requirements writer should look at RDD-100 *PerformanceIndex* elements to find requirements information related to system performance. However, during subsequent CoRE activities, the requirements likely to identify additional timing and accuracy requirements (e.g., for REQ relations) are not and should not be contained in the RDD-100 model.

Section App.2.5 identifies the *PerformanceIndex* elements for the HAS Buoy case study.

3.3 MAPPING THE RDD-100 MODEL TO AN INITIAL CoRE SPECIFICATION

This section describes how elements of an RDD-100 system design model map to a CoRE software requirements specification. Based on the RDD-100 approach described in Section 3.1, it describes what elements of the CoRE model are most likely to be found in the RDD-100 model.

As Figure 5 shows, the first CoRE activity, Identify Environmental Variables, includes development of the following: CoRE information model, candidate environmental variables, likely changes list, and environmental constraints specification (NAT). This section is divided into four subsections, each of which describes what parts of an RDD-100 model contain the information needed to build one of those products. Table 2 summarizes the mapping from RDD-100 schema elements to CoRE products.

Table 2. CoRE Products in the RDD-100 Model

CoRE Products	Candidate RDD-100 Schema Element(s)	Where Found (SEN Chapter)
CoRE information model	*Interface* *Component* *ExternalSystem* *DiscreteItem* *TimeItem*	Chapter 10, "Interfaces Between Components"
Candidate environmental variables	*Interface* *Component* *ExternalSystem* *DiscreteItem* *TimeItem* *CriticalIssue* *Decision* *SystemParameter*	Chapter 1, "System Top-Level Description" Chapter 8, "Item Dictionary"
Likely changes list	*CriticalIssue* *Decision* *SystemParameter*	Chapter 4, "Issues & Decisions"
Environmental constraints specification (NAT)	*Constraint*	Chapter 3, "Design Constraints"

3.3.1 CoRE INFORMATION MODEL

The CoRE information model captures physical entities and the associations between them that may be relevant to the software. An RDD-100 *System Component* element maps to the system entity in the CoRE information model. Other kinds of *Component* elements, especially *ExternalSystem* and *Environment Component* elements, are candidates for additional entities in the CoRE information model. *Interface* elements that represent connections between those *Components* map to relationships between the corresponding entities in the CoRE information model. *DiscreteItem* and *TimeItem* elements that are communicated by those *Components* map to attributes of entities.

Section App.3.3.1 contains the CoRE information model for the HAS Buoy case study.

3.3.2 CANDIDATE ENVIRONMENTAL VARIABLES

Candidate environmental variables (i.e., monitored and controlled variables) can be derived from:

- The likely changes list (see Section 3.3.3)

- Devices, environmental entities, or external hardware or software that has an effect on your system (see Section 3.3.1)

From the RDD-100 perspective, candidate environmental variables can be derived from any of the following RDD-100 elements: *Interface, Component, ExternalSystem, DiscreteItem, TimeItem, CriticalIssue, Decision,* or *SystemParameter*.

Section App.3.1 identifies candidate environmental variables for the HAS Buoy case study.

3.3.3 LIKELY CHANGES LIST

The likely changes list identifies likely changes in system requirements. Likely changes should be indicated by the existence of *CriticalIssue, Decision,* or *SystemParameter* elements in the RDD-100 model. If any of these elements exist in the RDD-100 model, apply the CoRE criteria to determine whether the element indicates the need for an addition to the likely changes list.

Section App.1.6 provides a list of likely changes for the HAS Buoy case study. Section App.2.4 identifies related RDD-100 *CriticalIssue* and *Decision* elements (there were no *SystemParameter* elements).

3.3.4 ENVIRONMENTAL CONSTRAINTS SPECIFICATION (NAT)

Environmental constraints are information about environmental variables related to possible values and interpretation of these values, e.g., the type of the quantity, possible range of values, and maximum rate of change. Environmental constraints may be indicated by the existence of *Constraint* elements in the RDD-100 model. Environmental constraints on the set of possible values that an environmental variable can take on are recorded by the NAT relation.

The HAS Buoy case study does not provide any examples of NAT relations derived from *Constraint* elements.

3.4 COMPLETING THE CoRE SPECIFICATION

This section describes how the remaining parts of a CoRE specification are affected by the use of RDD-100 after mapping the RDD-100 model to an initial CoRE specification as described in Section 3.3. In particular, it describes, from the CoRE perspective, where to find the necessary information to complete the CoRE specification when the RDD-100 is used as the front end to CoRE. Those parts of a CoRE specification that are unaffected by the use of RDD-100 are not described here.

As Figure 5 shows, the following remaining CoRE products are affected by the use of RDD-100 for systems engineering: context diagram (including monitored and controlled variables), input and output variable definitions, and timing and accuracy constraints. This section is divided into subsections that describe what parts of an RDD-100 model contain the information needed to build one of those products.

3.4.1 CONTEXT DIAGRAM

The CoRE context diagram captures the interaction of a software system within its environment. The CoRE context diagram represents a subset of the environment that might be represented in a context diagram for an entire system, such as for the system model described in Section 3.2.2. In some cases, however, the two context diagrams may be equivalent in scope, such as in the HAS Buoy case study, as shown in Section App.3.2.1.

The CoRE context diagram includes a system transformation (Section 3.4.1.1), terminators (Section 3.4.1.2), and monitored and controlled variables (Section 3.4.1.3).

3.4.1.1 System Transformation

The system transformation on the context diagram is derived from the system entity in the CoRE information model, which was derived from an RDD-100 *System Component*, named appropriately (see Section 3.3.1). The name of the system transformation may be inherited from the corresponding RDD-100 element.

3.4.1.2 Terminators

Terminators on a CoRE context diagram represent boundary classes. Boundary classes:

- Represent information about the environment that is relevant to specifying the behavior of the software

- Require external resources (devices or external software subsystems) to acquire or influence the information

The set of boundary classes is derived from the likely changes list and the CoRE information model as described in Section 3.3.1, which are based on corresponding *Component, CriticalIssue, Decision,* or *SystemParameter* elements in the RDD-100 model.

3.4.1.3 Monitored and Controlled Variables

CoRE environmental variables are physical quantities of interest to the software. Monitored variables are environmental variables measured by the software. Controlled variables are controlled by the

software. Monitored and controlled variables appear on the context diagram as arrows between the system transformation and terminators (boundary classes).

Candidate environmental variables can be found in the CoRE information model (see Section 3.3.1). Typically, environmental variables will appear as attributes of entities in the CoRE information model. Specifically, when RDD-100 is used, monitored variable candidates can be derived from RDD-100 *DiscreteItem* and *TimeItem* elements whose *Source* is a *Component* with a corresponding entity in the CoRE information model. Controlled variable candidates can be derived from RDD-100 *DiscreteItem* and *TimeItem* elements whose *Target* is a *Component* with a corresponding entity in the CoRE information model. CoRE recommends using the environmental constraints (see Section 3.3.4) and the likely changes list (see Section 3.3.3) as a guide in identifying and specifying the definitions of monitored and controlled variables.

3.4.2 INPUT AND OUTPUT VARIABLE DEFINITIONS

IN relations record how the software can use input variables to approximate the values of monitored variables. OUT relations record how the software can use output variables to set the values of controlled variables. Input variables are descriptions of physical interfaces to the environment that allow software to determine the values of monitored variables. Output variables are descriptions of physical interfaces to the environment that allow software to set the values of controlled variables.

Section 3.2.4 specifies that system component interface specifications should be contained by *TimeItem*, *DiscreteItem*, or *Interface* elements that are shared by the *System Component* element and other *Component* elements, particularly those of type *ExternalSystem* or *Environment*. Those *Component* elements of type *ExternalSystem* or *Environment*, whose behavior should be specified using behavior diagrams, are useful in specifying IN and OUT relations. The *TimeItem*, *DiscreteItem*, *ItemLink*, or *Interface* elements referred to by Section 3.2.4 are useful in specifying the corresponding input and output variables.

IN and OUT relations for the HAS Buoy case study are contained in Sections App.3.3.4.1 and App.3.3.4.3, respectively.

3.4.3 TIMING AND ACCURACY CONSTRAINTS

CoRE timing and accuracy constraints define the allowable tolerance in terms of timing and accuracy associated with CoRE's mathematical relations. These constraints, which are sometimes based on requirements information relating to system performance (see Section 3.2.5), may be derived from *PerformanceIndex* elements when using RDD-100 for system design. In other cases, timing and accuracy constraints will be determined later in the CoRE process after the RDD-100 model has been completed.

Section App.3.3.6 identifies the timing and accuracy constraints for the HAS Buoy case study that were captured in the RDD-100 model.

This page intentionally left blank.

4. EXTENDING THE RDD-100 SCHEMA TO SUPPORT CoRE

The RDD-100 Extender (Ascent Logic Corporation 1991a) allows systems engineers to tailor the underlying database schema of RDD-100 for their own needs. Extending the schema such that CoRE-specific elements, such as monitored variables and input variables, are recognized by RDD-100 will allow systems engineers to specify designs that are more consistent with the needs of CoRE.

Table 3 identifies those elements and relationships systems engineers might add to the RDD-100 database schema in support of CoRE as Figure 7 shows. Making these additions to the standard, underlying database schema of RDD-100 allows it to recognize CoRE concepts and, therefore, facilitates mapping from an RDD-100 model to a CoRE model.

Table 3. Extended Schema for CoRE

Source Element	Relationship	Target Element
Controlled Variable	Is_A_Kind_Of	Environmental Variable
	NAT Inverse	Monitored Variable
	REQ Inverse	Monitored Variable
	OUT Inverse	Output Variable
Monitored Variable	Is_A_Kind_Of	Environmental Variable
	NAT	Controlled Variable
	REQ	Controlled Variable
	IN	Input Variable
Input Variable	Is_A_Kind_Of	Input/Output Variable
	IN Inverse	Monitored Variable
Output Variable	Is_A_Kind_Of	Input/Output Variable
	OUT	Controlled Variable

In experimentation with the RDD-100 Extender for the purpose of supporting CoRE, the Consortium concluded that the benefit obtained by modifying the RDD-100 database schema for CoRE use is marginal. By adding CoRE-specific elements to the schema, new elements are created that parallel the purposes of existing elements. The benefit obtained is the capability to refer to RDD-100 elements by the names of their equivalent CoRE elements (e.g., monitored variable, input variable).

Ascent Logic Corporation has delivered Release 4.0 of RDD-100 (this report assumes version 3.0.2 of RDD-100), which contains Real World and Abstract Object Editors that obviate the need to extend

the database schema for CoRE using the Extender. The same benefits provided by the Extender can be obtained using the Object Editors of Release 4.0 of RDD-100 without the drawbacks. The Object Editors allow systems engineers to overlay a modified schema onto the predefined RDD-100 schema so that database elements are not replicated for the modified schema. Therefore, Table 3 is included in this report because it useful for identifying the objects and relations that might be added to the schema using the Object Editors. In a future version of this report, this section will describe in detail the use of the Real World and Abstract Object Editors instead of the Extender.

5. GENERATING A CoRE REPORT FROM RDD-100

This section describes a CoRE-specific report that can be automatically generated by RDD-100 to simplify the process of building an initial CoRE specification from an RDD-100 model. Generating such a report requires that systems engineers use RDD-100's Report Writer (Ascent Logic Corporation 1992b) to define the contents of the report.

RDD-100's Report Writer provides templates ready for use, including the following reports: SEN, data dictionary, component interface, requirements allocation, requirements traceability, and 2167A-compliant reports (Interface Requirements Specification, System/Segment Specification, System/Segment Design Document, and, soon, the Software Requirements Specification).

The predefined report that is most helpful to a CoRE analyst is the SEN (Ascent Logic Corporation, 1991b). Section 5.1 describes the contents of the SEN. Section 5.2 describes various options for generating a CoRE-specific report.

5.1 THE RDD-100 SYSTEM ENGINEERING NOTEBOOK

This section describes the RDD-100 SEN. For each section of the SEN, there is a corresponding subsection containing a short discussion of its contents and applicability to CoRE.

5.1.1 SYSTEM TOP-LEVEL DESCRIPTION

This section is very useful to CoRE: it describes a high-level view (a *TimeFunction*) of the system and identifies the major *Components* from which the system is built. It also identifies all external interfaces, performance requirements, sources of system requirements, etc. related to the high-level view of the system. Finally, it contains the behavior diagram illustrating system functionality from the highest level. This behavior diagram is very useful in mapping to a CoRE context diagram.

5.1.2 SYSTEM-LEVEL ("ORIGINATING") REQUIREMENTS

This section contains the hierarchy of system requirements in alphabetical order, including identification of relationships between them and other RDD-100 database elements. These system requirements are useful to many CoRE activities.

5.1.3 DESIGN CONSTRAINTS

This section identifies RDD-100 *Constraint* elements and their relationships to other RDD-100 database elements. These *Constraints* are useful in identifying CoRE NAT relations.

5.1.4 ISSUES & DECISIONS

This section identifies RDD-100 *CriticalIssue* and *Decision* elements, which are important to CoRE when identifying likely changes and candidate environmental variables.

5.1.5 HIERARCHICAL FUNCTION LIST

This section identifies the RDD-100 *TimeFunctions* that have been decomposed into lower level *TimeFunctions* or *DiscreteFunctions* (excluding those that are performed by *Components*). These functions generally represent a lower level of detail than is necessary for CoRE and, therefore, are not likely to be of interest to CoRE. Section 5.1.6 describes the details of these functions.

5.1.6 SYSTEM FUNCTIONAL BEHAVIOR DESCRIPTION

This section contains the behavior diagrams for the RDD-100 *TimeFunctions* identified in Section 5.1.5. These functions generally represent a lower level of detail than is necessary for CoRE and, therefore, are not likely to be of interest to CoRE.

5.1.7 PERFORMANCE INDICES

This section describes the *PerformanceIndex* elements that appear in the RDD-100 database, which are of interest to CoRE when specifying timing and accuracy constraints.

5.1.8 ITEM DICTIONARY

This section describes each *TimeItem* and *DiscreteItem* in the RDD-100 database. *TimeItem* and *DiscreteItem* elements are important to the CoRE practitioner when specifying the CoRE information model, monitored and controlled variables, and input and output variable definitions.

5.1.9 COMPONENTS

This section of the SEN describes the characteristics of *Components* in the RDD-100 database, which are important throughout the CoRE process. This section illustrates the hierarchy of system *Components*. This hierarchy is useful in understanding the structure of the RDD-100 model but is not required by CoRE.

5.1.10 INTERFACES BETWEEN COMPONENTS

This section describes the interfaces between *Component* elements, including elements of types *TimeItem, DiscreteItem, Interface,* and *ItemLink*. This information is important to the CoRE practitioner when specifying the CoRE information model, monitored and controlled variables, and input and output variable definitions.

5.1.11 SYSTEM "OPERATIONAL" PARAMETERS

This section describes the *SystemParameter* elements that appear in the RDD-100 database, which may be of interest to CoRE when identifying candidate environmental variables and likely changes.

5.2 A CoRE-SPECIFIC REPORT

There are two ways to generate CoRE-specific reports using RDD-100's Report Writer:

- Predefined RDD-100 reports can be modified after being generated using a text editor.

- The contents of predefined RDD-100 reports are specified using hierarchies of behavior diagrams, identified by *ReportNet* elements in the RDD-100 database. Systems engineers can add or delete *ReportNet* elements or modify the behavior diagrams that define *ReportNets* so that the Report Writer produces a report describing those database elements of interest to them.

Two ways to modify predefined RDD-100 reports to generate CoRE specific reports using the Report Writer are:

- Begin with the predefined RDD-100 SEN:

    — Modify the headings of individual sections to indicate the potential for mapping from RDD-100 elements to CoRE elements (e.g., modify section headings for the sections containing *ExternalSystem* and *Environment Components* to indicate the potential for mapping to CoRE environmental variables).

    — Remove those sections that are not of interest to CoRE (i.e., the sections containing the Hierarchical Function List and System Functional Behavior Specification).

- If an RDD-100 schema modification has been made to support CoRE (see Section 4), a special CoRE report could be generated that represents all information assigned to CoRE-specific schema elements. To do this, you could begin with the *ReportNets* and behavior diagrams used by the Report Writer that specify a similar predefined report and modify them such that they specify a report containing CoRE-specific schema elements.

The best approach to generating a CoRE-specific report using RDD-100's Report Writer for specific needs depends on the amount of time systems engineers are willing to invest and their expected pay-back in terms of the amount of time they expect to save using the Report Writer. Systems engineers must trade off these parameters to determine which of the above approaches best suits their needs.

This page intentionally left blank.

6. USING THE RDD-100 BRIDGE TO TEAM*WORK*

Ascent Logic provides a bridge that allows systems engineers to translate RDD-100 behavior diagrams to team*work* context diagrams, data flow/control flow diagrams, and data dictionary entries. Cadre's team*work* (Cadre Technologies, Inc. 1990) can be used to develop CoRE work products, although it does not currently support CoRE directly. If systems engineers are using team*work* to record a CoRE specification, RDD-100's bridge to team*work* can provide a head start in building their CoRE specification. Some RDD-100 elements can be automatically translated to team*work* objects that are useful when building a CoRE specification.

Section 6.1 provides an overview of how the RDD-100 bridge to team*work* works. Section 6.2 provides an overview of how team*work* can be used to support CoRE. Section 6.3 describes how to use the RDD-100 bridge to Cadre's team*work*/RT tool to support the CoRE method.

6.1 THE RDD-100 BRIDGE TO TEAM*WORK*

Ascent Logic Corporation (1993b) describes the facility that allows systems engineers to translate RDD-100 behavior diagrams to team*work* context diagrams, data flow/control flow diagrams, and data dictionary entries. This bridge automates the transfer of certain elements from an RDD-100 database to the team*work* database by automatically generating CASE Data Interchange Format (CDIF) files from the RDD-100 model. CDIF files can be loaded into the team*work* database using the twk_put command, which is part of team*work*'s standard tool kit (Cadre Technologies, Inc. 1990).

The mapping from RDD-100 elements to team*work* objects is implemented according to the mapping shown in Table 4. The automated mapping to team*work* assumes that the system design is developed using RDD-100 behavior diagrams that adhere to some constraints defined in Ascent Logic Corporation (1993b, 2–4).

Table 4. RDD-100 Elements Mapping to Team*work*

RDD Element	Team*work* Object
System	Model
Component	Model
TimeFunction	Process Bubble or Terminator
DiscreteFunction	P-Spec (process specification)
TimeItem	Data Flow or Control Flow
DiscreteItem	Data Flow or Control Flow
DataStore	Data Store

Note that only a subset of the elements supported by RDD-100 map to team*work* objects. Also, there is no mapping to team*work* control specifications, including: state-transition diagrams, state-event matrices, process activation tables, and decision tables.

Data dictionary entries are also generated for each generated teamwork data flow, control flow, and data store. Ascent Logic Corporation (1993b, 2–5) describes the contents of the generated data dictionary entries as follows:

- Data dictionary entries for flows mapped from RDD-100 *TimeItems* list all items in the *TimeItem*'s current decomposition.

- The *Description* attribute of RDD-100 elements is included in the comments section of data dictionary entries. All other filled-in attributes of those elements are listed as teamwork extended attributes (i.e., at the end of the data dictionary entry following a dashed line).

6.2 USING TEAM*WORK* WITH CoRE

Much of the information in a CoRE specification can be recorded in teamwork in a straightforward manner. Teamwork can graphically record much of the information that specifies CoRE functional requirements using teamwork's decision tables and state transition diagrams. The rest of the CoRE specification can be recorded as text in teamwork's data dictionary.

Users can capture CoRE specifications in teamwork because CoRE was designed to use notations and formalisms that are common to available tools (e.g., data flow/control flow diagrams, finite state machines, etc.). Because CoRE departs from some conventions assumed by teamwork, the tool does not support CoRE as completely as it supports real-time structured analysis. In particular, the checks provided by teamwork are not useful to the CoRE user, and some of the CoRE specification is recorded as uninterpreted text.

The mapping from CoRE elements to teamwork objects is shown in Table 5.

Table 5. Teamwork's Support for CoRE Elements

CoRE Elements		Teamwork Objects
Information model	Entity	Entity in entity relationship diagram
	Relationship	Relation in entity relationship diagram
	Attribute	Defined in data dictionary entry for class
Boundary class	Class	Process bubble in data flow diagram
		Mode class control bar in data flow diagram
	Class interface	Control flows from process bubbles or from control bars that represent mode classes
Environmental variable	Monitored variable	Data flow from terminator on context diagram
		Data dictionary entry
	Controlled variable	Data flow to terminator on context diagram
		Data dictionary entry
Input/output variable	Input variable	Data dictionary entry
	Output variable	Data dictionary entry
Four-variable relations	REQ function	Control bar in data flow diagram
		Decision table

Table 5, continued

CoRE Elements		Teamwork Objects
	NAT relation*	Text specification in data dictionary entry for controlled variable
	IN relation*	Text specification in data dictionary entry for input variable
	OUT relation*	Text specification in data dictionary entry for output variable
Mode class	Mode class	Control bar in data flow diagram
		State transition diagram in control specification

* A commonly used mapping for teamwork four-variable relations is to a control bar and a corresponding decision table.

It is not obvious from Table 5 how CoRE makes use of context diagrams. A CoRE context diagram is used in much the same way as with teamwork: the central transformation represents the system to be built, and terminators represent external entities that have an effect on the system. The most significant difference is, however, that on the CoRE context diagram, arrows between terminators and the transformation represent environmental variables, not necessarily the flow of data.

6.3 USING THE RDD-100 BRIDGE TO TEAM*WORK* FOR CoRE

Of all the objects in the teamwork model generated by RDD-100's teamwork filter, the most useful part, in CoRE's perspective, is the context diagram. However, it probably will need modification for use by CoRE. Terminators on the teamwork context diagram are mapped from RDD-100 *Component* elements of type *ExternalSystem*, whose behavior is described using *TimeFunctions*. Although this mapping is useful for CoRE, it will not always be a precise mapping to CoRE terminators, which represent either sources of monitored variables or targets of controlled variables. Data flows on the CoRE context diagram are equivalent to monitored and controlled variables. The context diagram generated by RDD-100 may not adhere to this convention. The HAS Buoy Context Diagram of the HAS Buoy case study (Section App. 3.2.1) provides an example of the context diagram generated by RDD-100.

The data dictionary generated by RDD-100's teamwork filter provides useful definitions for CoRE. For example, when a data flow on the teamwork context diagram generated by RDD-100 maps to a CoRE environmental variable, the data dictionary entry for the data flow is useful as the specification of the environmental variable.

Attributes of the source RDD-100 elements are provided in the generated data dictionary entries (see Table 6). Most of these attribute values are useful for a CoRE specification. For example, the minimum value, maximum value, and units fields in Table 6 are useful when defining CoRE's input and output data items. However, some of these attributes may not be useful, such as RDD_Type, shown in Table 6.

Table 6. Sample Generated Data Dictionary Entry

```
Air_Temperature_DI (data flow, pel) =
   * An environmental variable describing the atmosphere. *
   ---------
          Author                System User;
          Creation Date         11 August 1993;
          Modification Date     7 October 1993;
          Modification Time     3:36:00 pm;
          Minimum Value         -40.0;
          Maximum Value         60.0;
          Mean Value            20.0;
          Units                 degree Celsius;
          Size                  1;
          Item Type             physical;
          RDD Type              message;
```

When teamwork objects are created using RDD-100's filter, RDD-100 is forced to generate labels for certain objects, such as transformations and data flows. RDD-100 has adopted a convention for identifying sources of those teamwork objects in the RDD-100 database. For example, the suffix "_DI" is added to the labels of teamwork objects mapped from DiscreteItems, and the suffix "_TF" is added to the labels of teamwork objects mapped from TimeFunctions. These labels are generally effective in conveying the purposes of the objects and traceability back to the RDD-100 model. However, maintaining these labels may become burdensome for the CoRE practitioner using teamwork, and it may be a useful exercise to strip those suffixes from the teamwork objects after the traceability from RDD-100 to teamwork is understood and recorded.

The lower level (below level 0) data flow/control flow diagrams and p-specs generated by RDD-100 are based on the hierarchy of TimeFunctions in the RDD-100 model. These diagrams indicate the relationships between TimeFunction, DiscreteFunction, TimeItem, DiscreteItem, and DataStore elements in the RDD-100 model according to the mapping in Table 4. Lower level data flow/control flow diagrams in CoRE are used to specify requirements class hierarchies, organized into classes. Any effective use of the RDD-100's automatically generated data flow/control flow diagram hierarchy for CoRE would be coincidental because criteria for building a hierarchy of RDD-100 TimeFunctions are unrelated to the criteria for building a CoRE requirements class hierarchy.

APPENDIX: HAS BUOY CASE STUDY

This appendix contains examples from the HAS Buoy system case study. The examples were selected with the intent to illustrate application of the guidelines contained in this report.

Section App.1 contains the document that records the HAS Buoy problem statement upon which the case study was based. Section App.2 contains the RDD-100 system design of the HAS Buoy built from the problem statement of Section App.1. Section App.3 contains the CoRE model for the HAS Buoy that was derived based on the RDD-100 model in Section App.2.

APP.1 HAS BUOY PROBLEM STATEMENT

This section contains the document that records the HAS Buoy problem statement upon which the case study was based. The problem statement was adapted from *Software Engineering Principles* (Naval Research Laboratory 1980).

App.1.1 INTRODUCTION

The Navy intends to deploy HAS buoys to provide navigation and weather data to air and ship traffic at sea. The buoys will collect wind, temperature, and location data and will periodically broadcast summaries. Passing vessels will be able to request more detailed information. In addition, HAS buoys will be deployed in the event of accidents at sea to aid sea search operations.

App.1.2 HARDWARE

Each HAS buoy will contain a small computer, a set of wind and temperature sensors, and a radio receiver and transmitter. The temperature sensors take air and water temperature (Centigrade). Each buoy will have one or more wind sensors to observe wind magnitude in knots and one or more wind sensors to observe wind direction. Buoy geographic position is determined by use of a radio receiver link with the Omega navigation system.

Some HAS buoys are also equipped with a red light and an emergency button. The red light may be made to flash by a request radioed from a vessel during a sea search operation. If the sailors are able to reach the buoy, they may press the emergency button to initiate SOS broadcasts from the buoy.

App.1.3 SOFTWARE REQUIREMENTS

The software for the HAS buoy must satisfy the following requirements:

- Maintain current wind and temperature information by monitoring sensors regularly and averaging readings.

- Calculate location via the Omega navigation system.

- Broadcast wind and temperature information every 60 seconds.

- Broadcast more detailed reports in response to requests from passing vessels. The information broadcast and the data rate will depend on the type of vessel making the request (ship or airplane). All requests and reports will be transmitted in the RAINFORM format.

- Broadcast weather history information in response to requests from ships or satellites. The history report consists of the periodic 60-second reports from the last 48 hours.

- Broadcast an SOS signal in place of the ordinary 60-second message after a sailor presses the emergency button. This should continue until a vessel sends a reset signal.

- Cause the red light to start and stop flashing in response to requests from passing vessels.

- Accept external update data. Although HAS buoys calculate their own position, they must also accept correction information from passing vessels. The software must use the information to update its internal database.

App.1.4 SOFTWARE TIMING REQUIREMENTS

In order to maintain accurate information, readings must be taken from the sensing devices at the following fixed intervals:

temperature sensors:	every 10 seconds
wind sensors:	every 30 seconds

App.1.5 PRIORITIES

Since the buoy can transmit only one report at a time, conflicts will arise.

If the transmitter is free and more than one report is ready, the next report will be chosen according to the following priority ranking:

SOS	1	highest
Periodic	1	
Airplane Request	2	
Ship Request	3	
Weather History	4	lowest

App.1.6 HAS BUOY STABILITIES AND VARIABILITIES

This section identifies additional requirements imposed onto the HAS Buoy system. Each of these requirements is classified as either stable (not likely to change) or variable (likely to change).

The HAS Buoy requirements that are expected to change are as follows:

1. The number of sensors of each type with which it is equipped.

2. The types of sensors with which it is equipped. In addition to different temperature and wind speed sensors, a Buoy may be equipped with sonar sensors, wave spectra sensors, and other sensors that monitor the ocean environment.

3. The range, resolution, and response time of the sensors used.

4. The frequency with which sensors are sampled.

5. The equipment used to determine location. Some buoys may use the Omega navigation system and its successors. Others may use inertial measurement systems.

6. The sources of external messages that the buoy may receive. In addition to U.S. Navy ships, some buoys may receive messages from satellites.

7. The format of the messages transmitted and received by the buoy.

8. The history interval, i.e., the length of time of the history .

9. The computer system used, including the speed, primary memory size, and availability of secondary memory.

10. The number of computers used.

11. The number and type of radio transmitters and receivers used.

12. The frequency with which wind and temperature data will be transmitted. The expected frequency is once per minute.

13. For some buoys, the position of some sensors may have to be recorded, e.g., water temperature sensors may be deliberately deployed at different depths.

14. The frequency with which various types of BIT are performed, the types and frequencies of occurrence of sensor and computer malfunction that require recovery strategies to be invoked, and the strategies for recovery from resource failure.

The HAS Buoy requirements that are considered to be stable are as follows:

1. The buoy is equipped with a set of sensors that monitor environmental conditions. The value of a particular environmental condition at a given time is a function of the readings of sensors that can measure, directly or indirectly, the condition. (A typical function used is the average.) The number and types of sensors onboard a particular buoy are fixed once the buoy begins operation.

2. The buoy monitors at least air and water temperature and wind speed. It monitors them at its location.

3. The buoy can determine its location to within a specified tolerance.

4. The buoy maintains a finite history of the environmental data it has collected, a history of its location, and a correlation between the two. Included in the history is the time at which data were collected. The required length of the history does not change once the buoy begins operation.

5. The buoy is equipped with at least one radio transmitter and at least one receiver that enable it to receive and transmit messages. It shall at least be able to receive messages from passing U.S. Navy ships in the standard RAINFORM format.

6. The buoy transmits messages containing current wind and temperature information periodically. The period does not change once the buoy begins operation.

7. The buoy will respond to requests that it receives, via radio, to transmit more detailed reports on environmental conditions and to transmit weather history information, including both weather data and the location and time at which the weather conditions occurred.

8. The buoy is equipped with an emergency switch, which, when flipped, causes the buoy to transmit an SOS signal in place of its periodic wind and temperature reports. The SOS signal ceases when the buoy receives a reset message.

9. When the location as determined by the buoy is significantly different from the location supplied externally, the buoy will use self-diagnostics to attempt to determine and eliminate the source of the error. The criteria for significantly different are fixed once the buoy begins operation.

10. The buoy shall function without noticeable degradation with damage to up to 20% of its sensors. If more than 20% are improperly functioning, both periodic and request reports shall be marked suspect. In the event that data are considered unusable, a defective report shall be sent in place of the suspect data.

APP.2 HAS BUOY RDD-100 MODEL

This section contains the RDD-100 system design of the HAS Buoy built from the problem statement of Section App.1. The RDD-100 model of the HAS Buoy system is most easily represented in a report by providing the RDD-100 SEN for the system.

The RDD-100 SEN provides a report including a complete description of all of the information in the RDD-100 database. However, much of this information is not applicable for the software requirements engineer using CoRE (see Section 5). Therefore, only those parts of the SEN that are of interest to CoRE users are included in this section. In particular, the following portions of the SEN are not included in this section:

- Section 5, Hierarchical Function List

- Section 6, System Functional Behavior Description

Note that the RDD-100 notational conventions have been adhered to as closely as possible. That is:

- Major subsections are numbered (e.g., App.2.1 System Top-Level Description).

- Minor subsections are in *boldfaced italicized serif* font.

- Relationships and attributes of elements are in *italicized serif* font.

An additional convention has been adopted for this report: RDD-100 database elements that are not expected to be of interest to CoRE are named with all lower case letters (e.g., "sensor interface box" instead of "Sensor Interface Box").

App.2.1 System Top-Level Description

HAS Buoy

Purpose: This is the set of all requirements for the overall buoy system.

Built From Components:

```
External System:  Air
System:  HAS Buoy
        1.1     sensors package
                Subsystem:     1.1.1   Sensors
                               1.1.2   sensor interface box
        1.2     communications package
                               1.2.1   comm interface box
                Subsystem:     1.2.2   transmitter
                Subsystem:     1.2.3   receiver
        1.3     other packages
External System:  Light
External System:  Omega System
External System:  Sailor
External System:  Vessel
External System:  Water
```

External Interfaces

System-Level *Performance Requirements:*

```
Air Temperature Accuracy
Period Accuracy to 10%
Water Temperature Accuracy
```

Source Document: HAS ADARTS Problem Statement

Performs Top-Level Function:

HAS Buoy (see Figure 8). Note: Although the Abstract Object Editor is provided in Release 4.0 of RDD-100 (not Version 3.0.2), Figure 9, which requires the Abstract Object Editor, is included because it illustrates the effectiveness of the new tool for supporting CoRE. Note that Figure 9 identifies those *DiscreteItems* that may map to CoRE environmental variables.

Figure 8. HAS Buoy Behavior Model

Figure 9. HAS Buoy Behavior Model Including Abstract Object Editor
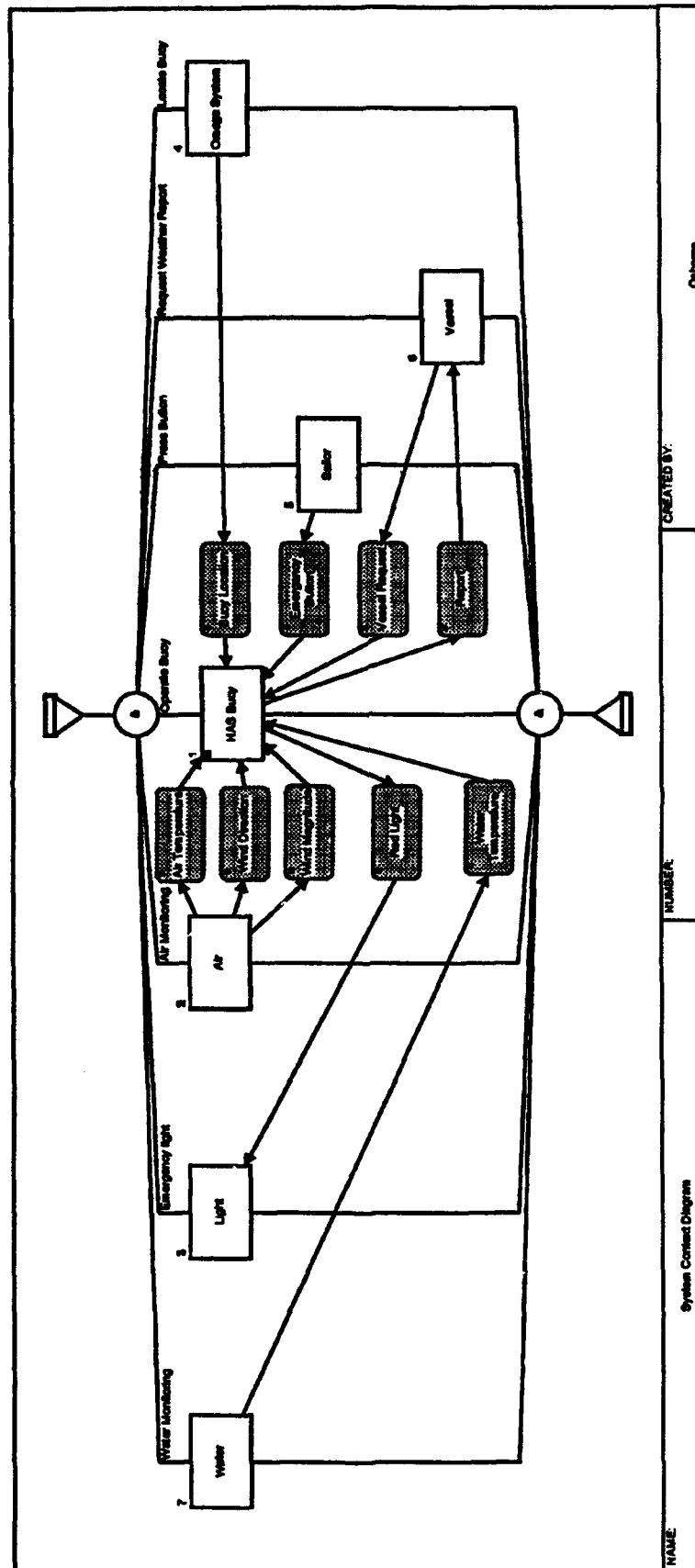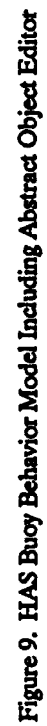
Inputs:

        Air Temperature *source: ExternalSystem: Air*
        Buoy Location *source: ExternalSystem: Omega System*
        Emergency Button *source: ExternalSystem: Sailor*
        Vessel Request *source: ExternalSystem: Vessel*
        Water Temperature *source: ExternalSystem: Water*
        Wind Direction *source: ExternalSystem: Air*
        Wind Magnitude *source: ExternalSystem: Air*

Outputs:

        Red Light *destination: ExternalSystem: Light*
        Report *destination: ExternalSystem: Vessel*

First-Level System Functions:

        HAS Buoy *TimeFunction: 0 Buoy Performance*

App.2.2 SYSTEM-LEVEL ("ORIGINATING") REQUIREMENTS

In order to reduce the size of this document only a few system-level requirements are provided in their entirety for exemplary purposes. Only the names of the remaining system-level requirements are provided.

Air Temperature

Description:

        The buoy monitors at least air and water temperature and wind speed. It monitors them at its location.

Traces To: PerformanceIndex: Air Temperature Accuracy

Source Document: HAS Stabilities

Communications

Description:

        The buoys will collect wind, temperature, and location data and will periodically broadcast summaries. Passing vessels will be able to request more detailed information.

Incorporates System Requirement(s):

        History Interval
        Message Format
        Message Source
        Number of Transceivers
        Response to Requests

Traces To: PerformanceIndex: Period Accuracy to 10%

Source Document: HAS ADARTS Problem Statement

Computer

Description:

> 9. The computer system used, including the speed, primary memory size, and availability of secondary memory.

Source Document: HAS Variabilities

Emergency Switch

Description:

> 8. The buoy is equipped with an emergency switch, which, when flipped, causes the buoy to transmit an SOS signal in place of its periodic wind and temperature reports. The SOS signal ceases when the buoy receives a reset message.

Source Document: HAS Stabilities

External Location

Description:

> 9. The buoy can accept location data from external sources, such as passing ships, via radio messages. When the location as determined by the buoy is significantly different from the location supplied externally, the buoy will use self-diagnostics to attempt to determine and eliminate the source of the error. The criteria for significantly different are fixed once the buoy begins operation.

Incorporates System Requirement(s): Self Diagnosis

Source Document: HAS Stabilities

Fixed Periodicity

Description:

> 6. The buoy transmits messages containing current wind and temperature information periodically. The period does not change once the buoy begins operation.

Source Document: HAS Stabilities

Graceful Degradation

Description:

> 10. The buoy shall function without noticeable degradation with damage to up to 20% of its sensors. If more than 20% are improperly functioning, both periodic and request reports shall be marked

suspect. In the event that data are considered unusable, a defective report shall be sent in place of the suspect data.

Incorporates System Requirement(s): Suspect Reports

Source Document: HAS Stabilities

Hardware

Description:

Each HAS buoy will contain a small computer, a set of wind and temperature sensors, and a radio receiver and transmitter. The temperature sensors take air and water temperature (Centigrade). Each buoy will have one or more wind sensors to observe wind magnitude in knots and one or more wind sensors to observe wind direction. Buoy geographic position is determined by use of a radio receiver link with the Omega navigation system.

Some HAS buoys are also equipped with a red light and an emergency button. The red light may be made to flash by a request radioed from a vessel during a sea search operation. If the sailors are able to reach the buoy, they may press the emergency button to initiate SOS broadcasts from the buoy.

Incorporates System Requirement(s):

Communications
Computer
Emergency Switch
Graceful Degradation
History Interval
Malfunctions
Message Format
Message Source
Number of Computers
Number of Transceivers
Response to Requests
Suspect Reports

Source Document: HAS ADARTS Problem Statement

History Interval

Description: 8. The history interval, i.e., the length of time of the history.

Source Document: HAS Variabilities

History Maintenance

Description:

4. The buoy maintains a finite history of the environmental data it has collected, a history of its location, and a correlation between the two. Included in the history is the time at which data were collected. The required length of the history does not change once the buoy begins operation.

Source Document: HAS Stabilities

The remaining system-level requirements are listed by name only.

Location

Location Tolerance

Malfunctions

Message Format

Message Source

Minimum Communications

Number of Comp

Number of Transceivers

Priorities

Response to Requests

Self Diagnosis

Sense Environment

Sensor Number

Sensor Period

Sensor Position Recording

Sensor Range

Sensor Type

Sensors

Software Requirements

Software Timing

Suspect Reports

Transmission Period

Water Temperature

Wind Speed

App.2.3 Design Constraints

Cost

Description: The buoy's parts shouldn't cost more than $2,000.

Constrains:

> Decision: Available Hardware
> Decision: Two Formats

Traced From: Decision: Two Formats

RAINFORM Format

Description:

> The buoy ... shall at least be able to receive messages from passing U.S. Navy ships in the standard RAINFORM format.
>
> Domain Stabilities, #5

Constrains:

> CriticalIssue: How Many Formats?
> Decision: Two Formats

Traced From:

> CriticalIssue: How Many Formats?
> SystemRequirement: Message Format
> SystemRequirement: Message Source
> Decision: Two Formats

App.2.4 Issues & Decisions

Decisions that trace from Critical Issues

Issue: Error Correction?

Originator: System User

Origination Date: 23 August 1993

Description:

> How is the required error correction to be performed?
>
> From #9, "HAS Stabilities": "When the location as determined by the buoy is significantly different from the location supplied externally, the buoy will use self-diagnostics to attempt to determine and eliminate the source of the error. The criteria for significantly different are fixed once the buoy begins operation."

Source Document: HAS Stabilities

Traced From:

> SystemRequirement: Location Tolerance
> SystemRequirement: Malfunctions

Traces To: Source: HAS Stabilities

Issue: How Many Formats?

Originator: System User

Origination Date: 23 August 1993

Description:

> At least one format must be recognized for message passing (standard RAINFORM format). Are others required by the existing vessels' communications gear? How many others are desirable?

Traced From: Decision: Two Formats

Traces To:

> Source: HAS Stabilities
> Constraint: RAINFORM Format

Issue: How Much History?

Originator: System User

Origination Date: 23 August 1993

Description:

What is the longest history trail required by any HAS buoy?

> #4 in Source: "The buoy maintains a finite history of the environmental data it has collected, a history of its location, and a correlation between the two. Included in the history is the time at which data were collected."

Source Document: HAS Stabilities

Traced From: Decision: Available Hardware

Traces To: Source: HAS Stabilities

Decisions that do not trace from Critical Issues

Decision: Available Hardware

Approved By: System User

Approval Date: 23 August 1993

Description:

> History will be kept based on the available RAM memory provided in the onboard computer. No specific added provision will be added, to contain costs.

Problem:

> The buoy maintains a finite history of the environmental data it has collected, a history of its location, and a correlation between the two. Included in the history is the time at which data were collected. The required length of the history does not change once the buoy begins operation.

Alternatives:

> 1) If a long string of history is not a necessity, history should be kept based only on the available RAM memory provided in the onboard computer.

> 2) If a long string of history is important, extra RAM memory should be provided in the onboard computer.

Choice: No. 1 is chosen, pending further notice.

Source Document: HAS ADARTS Problem Statement

Traces To: CriticalIssue: How Much History?

Decision: Two Formats

Approved By: System User

Approval Date: 23 August 1993

Description: RAINFORM format will be supported, and a standard SOS format will be supported.

Problem:

> The buoy is equipped with at least one radio transmitter and at least one receiver that enable it to receive and transmit messages. It shall at least be able to receive messages from passing U.S. Navy ships in the standard RAINFORM format.

Alternatives:

> 1) The explicit requirem⌐ only for RAINFORM.

> 2) But, the critical nature of SOSes suggests recognizing older formats as well.

> 3) Flexibility would be improved if an onboard mechanism to vary selection of formats was provided.

Choice: #2 was chosen, the 3rd running afoul of cost constraints.

Source Document: HAS Stabilities

Traces To:

>   Constraint: Cost
>   CriticalIssue: How Many Formats?
>   SystemRequirement: Message Format
>   SystemRequirement: Message Source
>   SystemRequirement: Minimum Communications
>   Constraint: RAINFORM Format

App.2.5 PERFORMANCE INDICES

Air Temperature Accuracy

Description: Air temperature must be kept to within 1 degree Centigrade.

Units:

Value:

Category: Measurement

Exhibited By:

>   Component: HAS Buoy
>   System: HAS Buoy
>   Component: 1.1.1 – Sensors

Traced From:

>   SystemRequirement: Air Temperature
>   SystemRequirement: Sensors

Period Accuracy to 10%

Description:

>   The broadcast cycle must be accurate to within 3 seconds for the wind sensors, and
>   1 second for the temperature sensors.

Units:

Value:

Category: Measurement

Exhibited By:

>   Component: HAS Buoy
>   System: HAS Buoy

Component: 1.2 – communications package
Component: 1.2.2 – transmitter

Traced From:

SystemRequirement: Communications
SystemRequirement: Software Timing

Water Temperature Accuracy

Description: Water temperature must be kept to within 1 degree Centigrade.

Units:

Value:

Category: Measurement

Exhibited By:

Component: HAS Buoy
System: HAS Buoy
Component: 1.1.1 – Sensors

Traced From:

SystemRequirement: Sensors
SystemRequirement: Water Temperature

App.2.6 Item Dictionary

acknowledgement

Output From:

Component: 1.2 communications package
Component: 1.2.1 comm interface box
DiscreteFunction: 2.1.1 receive message
  (Allocated Onto: Component: comm interface box)

Input To:

Component: 1.1 sensors package
Component: 1.1.2 sensor interface box
DiscreteFunction: 1.2.2 wait for ack
  (Allocated Onto: Component: sensor interface box)

Air Temperature

Description: An environmental variable describing the atmosphere.

Output From: ExternalSystem: Air

Input To:

> System: HAS Buoy
> Component: 1 simple WB
>
> System: HAS Buoy
> Component: 1.1 sensors package
> Subsystem: 1.1.1 Sensors
> DiscreteFunction: 1.1.1 gather periodic samples
> *(Allocated Onto: Subsystem: Sensors)*

Allocated to Component: External System: Air

Buoy Location

Description: An environmental variable describing where the HAS buoy is situated.

Output From: ExternalSystem: Omega System

Input To:

> System: HAS Buoy
> Component: 1 simple WB
> System: HAS Buoy
> Component: 1.2 communications package
> Subsystem: 1.2.3 receiver
> DiscreteFunction: 2.3.1 receive navigation signal
> *(Allocated Onto: Subsystem: receiver)*

Allocated to Component: External System: Omega System

collected data

Output From:

> Subsystem: 1.1.1 Sensors
> DiscreteFunction: 1.1.1 gather periodic samples
> *(Allocated Onto: Subsystem: Sensors)*

Input To:

> Component. 1.1.2 sensor interface box
> DiscreteFunction: 1.2.1 send message
> *(Allocated Onto: Component: sensor interface box)*

Emergency Button

Description:

> An environmental variable describing whether a sailor has gotten to the HAS buoy and pressed the
> button to cut on the emergency beacon.

Output From: ExternalSystem: Sailor

Input To:

> System: HAS Buoy
> Component: 1 simple WB
> System: HAS Buoy
> Component: 1.1 sensors package
> Subsystem: 1.1.1 Sensors
> DiscreteFunction: 1.1.1 gather periodic samples
> *(Allocated Onto: Subsystem: Sensors)*

Allocated to Component: External System: Sailor

internal message

Output From:

> Component: 1.1 sensors package
> Component: 1.1.2 sensor interface box
> DiscreteFunction: 1.2.1 send message
> *(Allocated Onto: Component: sensor interface box)*

Input To:

> Component: 1.2 communications package
> Component: 1.2.1 comm interface box
> DiscreteFunction: 2.1.1 receive message
> *(Allocated Onto: Component: comm interface box)*

Red Light

Description:

> An environmental variable describing whether the emergency beacon is turned on or not.

Output From:

> System: HAS Buoy
> Component: 1 simple WB
> System: HAS Buoy
> Component: 1.2 communications package
> Subsystem: 1.2.2 transmitter
> DiscreteFunction: 2.2.2 transmit one singular

Input To: ExternalSystem: Light

Allocated to Component: External System: Light

Report

Description:

> An environmental variable embodying the weather and location report previously requested.

Output From:

> System: HAS Buoy
> Component: 1 simple WB
>
> System: HAS Buoy
> Component: 1.2 communications package
> Subsystem: 1.2.2 transmitter
> DiscreteFunction: 2.2.1 transmit one periodic
> *(Allocated Onto: Subsystem: transmitter)*

Input To: ExternalSystem: Vessel

Allocated to Component: External System: Vessel

transmit item

Output From:

> Component: 1.2.1 comm interface box
> DiscreteFunction: 2.1.1 receive message
> *(Allocated Onto: Component: comm interface box)*

Input To:

> Subsystem: 1.2.2 transmitter
> DiscreteFunction: 2.2.1 transmit one periodic
> *(Allocated Onto: Subsystem: transmitter)*

Vessel Request

Description:

> An environmental variable describing a vessel's desire to get a current weather and/or location report.

Output From: ExternalSystem: Vessel

Input To:

> System: HAS Buoy
> Component: 1 simple WB
> System: HAS Buoy
> Component: 1.1 sensors package
> Subsystem: 1.1.1 Sensors
> DiscreteFunction: 1.1.1 gather periodic samples
> *(Allocated Onto: Subsystem: Sensors)*

Allocated to Component: External System: Vessel

Water Temperature

Description: An environmental variable describing the surrounding ocean.

Output From: ExternalSystem: Water

Input To:

> System: HAS Buoy
> Component: 1 simple WB
> System: HAS Buoy
> Component: 1.1 sensors package
> Subsystem: 1.1.1 Sensors
> DiscreteFunction: 1.1.1 gather periodic samples
> *(Allocated Onto: Subsystem: Sensors)*

Allocated to Component: External System: Water

Wind Direction

Description: An environmental variable describing which way the wind is blowing.

Output From: ExternalSystem: Air

> Input To:
> System: HAS Buoy
> Component: 1 simple WB
> System: HAS Buoy
> Component: 1.1 sensors package
> Subsystem: 1.1.1 Sensors
> DiscreteFunction: 1.1.1 gather periodic samples
> *(Allocated Onto: Subsystem: Sensors)*

Carried by interface link: Wind Direction

Allocated to Component: External System: Air

Wind Magnitude

Description: An environmental variable describing how big the wind is.

Output From: ExternalSystem: Air

Input To:

> System: HAS Buoy
> Component: 1 simple WB
> System: HAS Buoy
> Component: 1.1 sensors package
> Subsystem: 1.1.1 Sensors

DiscreteFunction: 1.1.1 gather periodic samples
(Allocated Onto: Subsystem: Sensors)

Allocated to Component: External System: Air

App.2.7 COMPONENTS

In order to reduce the size of this document, this section describes only a subset of the *Components* in their entirety. All of the *Components*, however, are identified. Figure 10 illustrates the component hierarchy described in this section.
Air

Component Type: External System

Description: HAS Buoy must perform measurements of the atmosphere.

Builds Higher-Level Component/System:

> ExternalSystem: Air
> System: HAS Buoy

Allocated Items:

> DiscreteItem: Air Temperature
> DiscreteItem: Wind Direction
> DiscreteItem: Wind Magnitude

HAS Buoy

Component Type: System

Description:

> HAS Buoy is the internal part of the system, comprising all hardware and its embedded software.

Builds Higher-Level Component/System: System: HAS Buoy

Built From Components:

> 1.1   sensors package
> 1.2   communications package
> 1.3   other packages

Allocated Functions:

> TimeFunction: 0   Buoy Performance
> TimeFunction: 1   HAS Buoy

Component-Level Performace Requirements:

> Air Temperature Accuracy
> Period Accuracy to 10%
> Water Temperature Accuracy
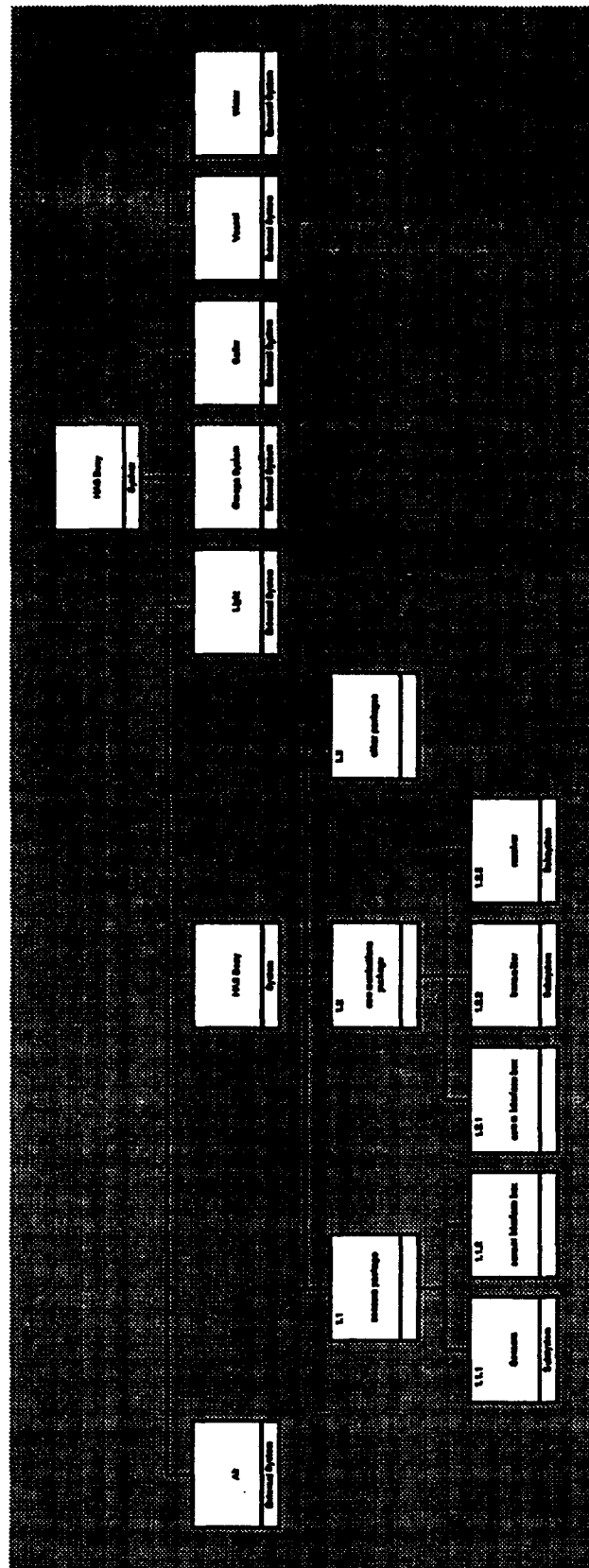
Figure 10. HAS Buoy Component Hierarchy

Light

Component Type: External System

Description: HAS Buoy must turn on a red emergency light in this external medium.

Builds Higher-Level Component/System:

> System: HAS Buoy
> ExternalSystem: Light

Allocated Items: DiscreteItem: Red Light

Omega System

Component Type: External System

Description: HAS Buoy must be located via information from this external medium.

Builds Higher-Level Component/System:

> System: HAS Buoy
> ExternalSystem: Omega System

Allocated Items: DiscreteItem: Buoy Location

Sailor

Component Type: External System

Description:

HAS Buoy must take emergency button pushes from this personal, but external, medium.

Builds Higher-Level Component/System:

> System: HAS Buoy
> ExternalSystem: Sailor

Allocated Items: DiscreteItem: Emergency Button

Vessel

Component Type: External System

Description: HAS Buoy must receive requests from, and send reports to, this external medium.

Builds Higher-Level Component/System:

> System: HAS Buoy
> ExternalSystem: Vessel

Allocated Items:

> DiscreteItem: Report
> DiscreteItem: Vessel Request

Water

Component Type: External System

Description: HAS Buoy must perform measurements of this external medium.

Builds Higher-Level Component/System:

> System: HAS Buoy
> ExternalSystem: Water

Allocated Items: DiscreteItem: Water Temperature

1 simple WB

Component Type:

Built From Components:

> 1.1 sensors package
> 1.2 communications package
> 1.3 other packages

1.1 sensors package

Component Type:

Builds Higher-Level Component/System:

> Component: HAS Buoy
> Component: 1 simple WB

Built From Components:

> Subsystem: 1.1.1 Sensors
> 1.1.2 sensor interface box

Allocated Functions: TimeFunction: 1 sensor functions

The remaining components are listed only by name.

1.1.1 Sensors

1.1.2 sensor interface box

1.2 communications package

1.2.1 comm interface box

1.2.2 transmitter

1.2.3 receiver

1.3 other packages

App.2.8 INTERFACES BETWEEN COMPONENTS

Derived Interfaces

Items Flowing "**from**" Component: comm interface box
"**into**" Component: sensor interface box

DiscreteItem: acknowledgement
  Output From: 2.1.1 - receive message
  Input To: 1.2.2 - wait for ack

Items Flowing "**from**" Component: comm interface box
"**into**" Component: sensors package

DiscreteItem: acknowledgement
  Output From: 2.1.1 - receive message
  Input To: 1 - sensor functions

Items Flowing "**from**" Component: comm interface box
"**into**" Subsystem: transmitter

DiscreteItem: transmit item
  Output From: 2.1.1 - receive message
  Input To: 2.2.1 - transmit one periodic

Items Flowing "**from**" Component: communications package
"**into**" Component: sensor interface box

DiscreteItem: acknowledgement
  Output From: 2 - communication functions
  Input To: 1.2.2 - wait for ack

Items Flowing "**from**" Component: communications package
"**into**" Component: sensors package

DiscreteItem: acknowledgement
  Output From: 2 - communication functions
  Input To: 1 - sensor functions

Items Flowing "**from**" Component: sensor interface box
"**into**" Component: comm interface box

DiscreteItem: internal message
  Output From: 1.2.1 - send message
  Input To: 2.1.1 - receive message

Items Flowing "from" Component: sensor interface box
"into" Component: communications package

DiscreteItem: internal message
Output From: 1.2.1 - send message
Input To: 2 - communication functions

Items Flowing "from" Subsystem: Sensors
"into" Component: sensor interface box

DiscreteItem: collected data
Output From: 1.1.1 - gather periodic samples
Input To: 1.2.1 - send message

Items Flowing "from" Component: sensors package
"into" Component: comm interface box

DiscreteItem: internal message
Output From: 1 - sensor functions
Input To: 2.1.1 - receive message

Items Flowing "from" Component: sensors package
"into" Component: communications package

DiscreteItem: internal message
Output From: 1 - sensor functions
Input To: 2 - communication functions

Database Interface Elements

communicates

Connects to:

System: HAS Buoy
ExternalSystem: Vessel
External System: Vessel

controls

Connects to:

System: HAS Buoy
ExternalSystem: Light
External System: Light

locates

Connects to:

System: HAS Buoy
ExternalSystem: Omega System
External System: Omega System

monitors

Connects to:

> ExternalSystem: Air
> External System: Air
> System: HAS Buoy
> External System: Water
> ExternalSystem: Water

switches

Connects to:

> System: HAS Buoy
> External System: Sailor
> ExternalSystem: Sailor

App.2.9 SYSTEM "OPERATIONAL" PARAMETERS

This section did not apply to this case study.

APP.3 HAS BUOY CoRE MODEL

This section contains the CoRE model for the HAS Buoy that was derived based on the RDD-100 model in Section App.2. Section App.3.1 describes how RDD-100 database elements of the HAS Buoy model were mapped to CoRE elements. Section App.3.2 contains the results of filtering the RDD-100 model into *teamwork* format. Section App.3.3 contains the remaining CoRE work products for the HAS Buoy system.

App.3.1 MAPPING FROM RDD-100 TO CoRE

Table 7 summarizes how RDD-100 database elements map to CoRE inputs for the HAS Buoy model, according to the mapping specified in Section 3.2. For each expected input to CoRE, Table 7 identifies:

- The RDD-100 database elements providing the necessary information

- An estimation of the degree of completeness provided by the mapping

Table 7. Mapping From RDD-100 to CoRE

Expected CoRE Inputs	RDD-100 Element Type (HAS Buoy Instances)	Degree of Mapping
Mission statement	*Source* • HAS ADARTS Problem Statement • HAS Stabilities • HAS Variabilities	Complete
System model	*FNet* • System Context Diagram *Component* • HAS Buoy (system) • Air (external system) • Water (external system) • Vessel (external system) • Light (external system) • Omega System (external system) • Sailor (external system)	Complete

Table 7, continued

System requirements specification	*SystemRequirement* • Air Temperature • Communications • Computer • Emergency Switch • External Location • Fixed Periodicity • etc. (see Section App.2.2)	Complete
System component interface specifications	*Interface* *ItemLink* *DiscreteItem* *TimeItem*	Complete
Requirements information relating to system performance	*PerformanceIndex* • Air Temperature Accuracy • Period Accuracy to 10% • Water Temperature Accuracy	Incomplete

Table 8 summarizes how the initial set of CoRE work products were derived from RDD-100 database elements for the HAS Buoy model, according to the mapping specified in Section 3.3.

Table 8. Mapping From RDD-100 to Initial CoRE Products

CoRE Products	RDD-100 Element Type (HAS Buoy Instances)	Degree of Mapping
CoRE information model	*Interface* • communications • controls • locates • monitors • switches *ExternalSystem* • Air • Water • Vessel • Light • Omega System • Sailor *Component* • HAS Buoy	Complete
Candidate environmental variables	*DiscreteItem* • Vessel Request • Report • Air Temperature • Water Temperature • Wind Magnitude • Wind Direction • Buoy Location • Emergency Button • Red Light	Complete

Table 8, continued

| Likely changes list | *CriticalIssue*
• Error Correction?
• How Many Formats?
• How Much History?
Decision
• Available Hardware
• Two Formats | Complete |
| Environmental constraints specification (NAT) | *Constraint*
• Cost
• RAINFORM Format | Incomplete |

Table 9 summarizes how the additional CoRE work products were derived from RDD-100 database elements for the HAS Buoy model, according to the mapping specified in Section 3.4.

Table 9. Mapping From RDD-100 to Additional CoRE Products

CoRE Products	RDD-100 Element Type (HAS Buoy Instances)	Degree of Mapping
Context diagram	*FNet* • System Context Diagram	Complete
Monitored and controlled variable definitions	*DiscreteItem* • Vessel Request • Report • Air Temperature • Water Temperature • Wind Magnitude • Wind Direction • Buoy Location • Emergency Button • Red Light	Incomplete

App.3.2 TEAMWORK-FILTERED VERSION OF HAS BUOY

This section contains those teamwork work products that were generated by RDD-100's teamwork filter from the RDD-100 model of the HAS Buoy system. Some parts of the generated teamwork model were not useful for CoRE and have been omitted.

Section App.3.2.1 contains the generated teamwork process index, and Section App.3.2.2 contains the generated teamwork data dictionary.

App.3.2.1 Teamwork Process Index

The teamwork process index generated by RDD-100 contained a context diagram, a data flow/control flow diagram hierarchy, and a number of p-specs. The context diagram was derived from the behavior diagram for the RDD-100 *System Component*. Each data flow/control flow diagram in the hierarchy was derived from a corresponding RDD-100 *FNet* or *TimeFunction*. The data flow/control flow diagrams contained a data transformation for each *DiscreteFunction* contained by the *FNet* or *TimeFunction*. Data flows in and out of data transformations were derived from *DiscreteItems* that were inputs and outputs of the *DiscreteFunctions*.

The generated context diagram in Figure 11 turned out to be ideal for CoRE.

Context-Diagram;1
System_Context_Diagram_FN
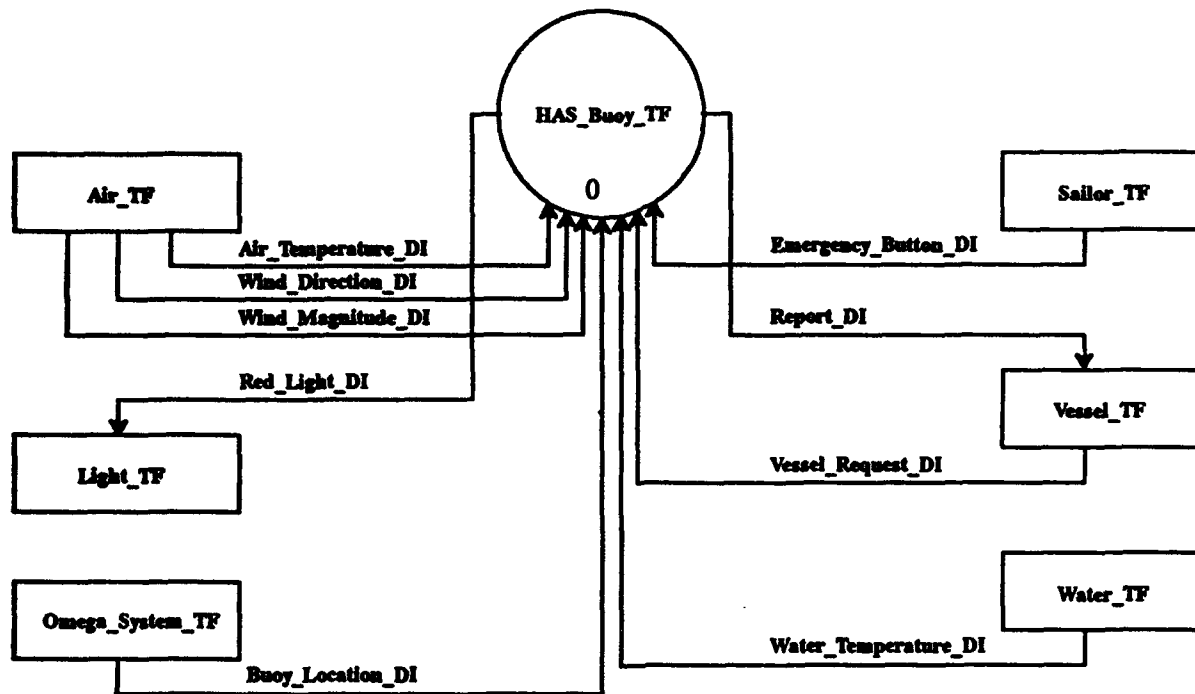
Figure 11. HAS Buoy Context Diagram

Neither the generated data flow/control flow diagrams nor the generated p-specs were useful for CoRE because they provide too low a level of detail. The generated p-specs contained only input and output lists.

App.3.2.2 Teamwork Data Dictionary

This section contains the teamwork data dictionary entries generated by RDD-100.

```
acknowledgement_DI (data flow, pel) =
        ----------
                Author               System User;
                Creation Date        31 January 1992;
                Modification Date    7 October 1993;
                Modification Time    3:35:59 pm;
                Size                 1;
                RDD Type             message;

Air_Temperature_DI (data flow, pel) =
        * An environmental variable describing the atmosphere. *
        ----------
                Author               System User;
                Creation Date        11 August 1993;
```

```
        Modification Date      7 October 1993;
        Modification Time      3:36:00 pm;
        Minimum Value          -40.0;
        Maximum Value          60.0;
        Mean Value             20.0;
        Units                  degree Celsius;
        Size                   1;
        Item Type              physical;
        RDD Type               message;
```

Buoy_Location_DI (data flow, pel) =
    * An environmental variable describing where the HAS buoy is situated. *

```
        Author                 System User;
        Creation Date          11 August 1993;
        Modification Date      7 October 1993;
        Modification Time      3:36:00 pm;
        Units                  characters;
        Size                   10;
        Item Type              data;
        RDD Type               message;
```

collected_data_DI (data flow, pel) =

```
        Author                 System User;
        Creation Date          31 January 1992;
        Modification Date      7 October 1993;
        Modification Time      3:36:01 pm;
        Size                   1;
        RDD Type               message;
```

Emergency_Button_DI (data flow, pel) =
    * An environmental variable describing whether a sailor has gotten to
      the HAS buoy and pressed the button to cut on the emergency beacon. *

```
        Author                 System User;
        Creation Date          11 August 1993;
        Modification Date      7 October 1993;
        Modification Time      3:36:01 pm;
        Minimum Value          0.0;
        Maximum Value          0.0;
        Units                  Bits;
        Size                   1;
        Item Type              digital;
        RDD Type               message;
```

internal_message_DI (data flow, pel) =

```
        Author                 System User;
        Creation Date          31 January 1992;
        Modification Date      7 October 1993;
        Modification Time      3:36:02 pm;
```

.

```
            Size                    1;
            RDD Type                message;


Red_Light_DI (data flow, pel) =
    * An environmental variable describing whether the emergency beacon is
      turned on or not. *
      ---------
            Author                  System User;
            Creation Date           11 August 1993;
            Modification Date       7 October 1993;
            Modification Time       3:36:02 pm;
            Minimum Value           0.0;
            Maximum Value           0.0;
            Units                   Bits;
            Size                    1;
            Item Type               digital;
            RDD Type                message;


Report_DI (data flow, ┌ 1) =
    * An environmental variable embodying the weather and location report
      previously requested.*
      ---------
            Author                  System User;
            Creation Date           11 August 1993;
            Modification Date       7 October 1993;
            Modification Time       3:36:02 pm;
            Units                   characters;
            Size                    100;
            Item Type               data;
            RDD Type                message;


transmit_item_DI (data flow, pel) =
      ---------
            Author                  System User;
            Creation Date           31 January 1992;
            Modification Date       7 October 1993;
            Modification Time       3:36:03 pm;
            Size                    1;
            RDD Type                message;


Vessel_Request_DI (data flow, pel) =
    * An environmental variable describing a vessel's desire to get a
      current weather and/or location report.*
      ---------
            Author                  System User;
            Creation Date           11 August 1993;
            Modification Date       7 October 1993;
            Modification Time       3:36:03 pm;
            Units                   characters;
            Size                    1;
            Item Type               data;
            RDD Type                message;
```

```
Water_Temperature_DI (data flow, pel) =
    * An environmental variable describing the surrounding ocean. *
        _____
            Author              System User;
            Creation Date       11 August 1993;
            Modification Date    7 October 1993;
            Modification Time    3:36:04 pm;
            Minimum Value        0.0;
            Maximum Value        50.0;
            Units                Degree Celsius;
            Size                 1;
            Item Type            physical;
            RDD Type             message;

Wind_Direction_DI (data flow, pel) =
    * An environmental variable describing which way the wind is blowing. *
        _____
            Author              System User;
            Creation Date       11 August 1993;
            Modification Date    7 October 1993;
            Modification Time    3:36:04 pm;
            Minimum Value        0.0;
            Maximum Value        359.9;
            Units                degrees of arc;
            Size                 1;
            Item Type            physical;
            RDD Type             message;

Wind_Magnitude_DI (data flow, pel) =
    * An environmental variable describing how big the wind is. *
        _____
            Author              System User;
            Creation Date       11 August 1993;
            Modification Date    7 October 1993;
            Modification Time    3:36:05 pm;
            Minimum Value        0.0;
            Maximum Value        300.0;
            Mean Value           10.0;
            Units                km/hr;
            Size                 1;
            Item Type            physical;
            RDD Type             message;
```

App.3.3 THE REMAINING CoRE WORK PRODUCTS

This section contains the remaining CoRE work products that complete the CoRE specification for the HAS Buoy system. The CoRE specification was recorded using Cadre's teamwork.

App.3.3.1 CoRE Information Model

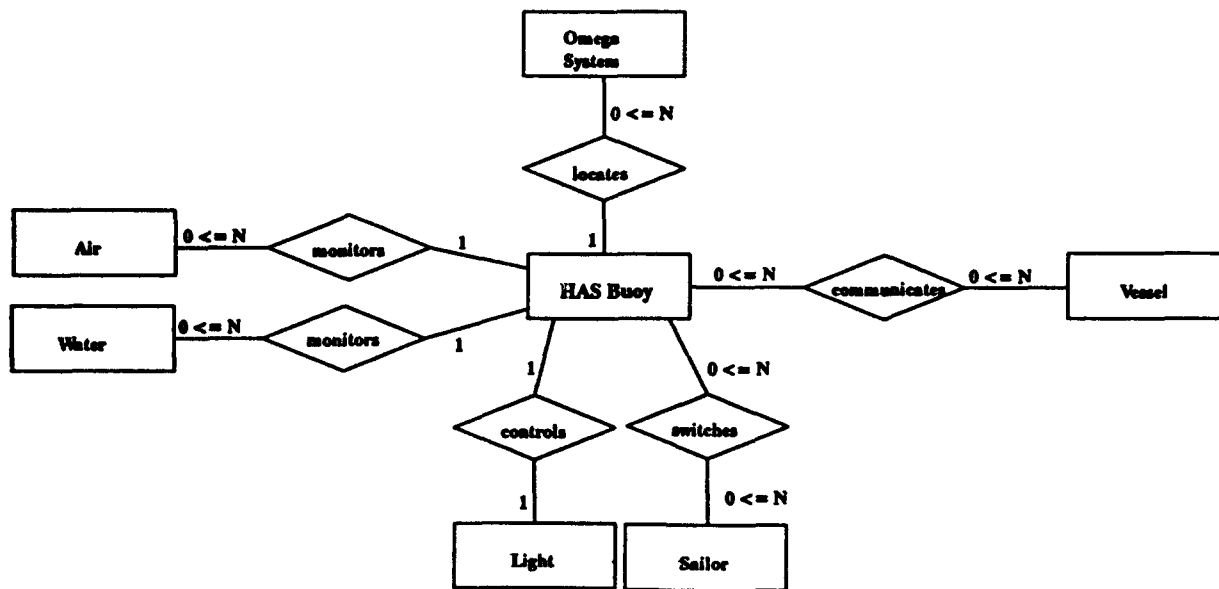The CoRE information model in Figure 12 was recorded using a teamwork entity-relationship diagram.

Figure 12. CoRE Information Model

App.3.3.2 Environmental Variable Definitions

Monitored variables are identified on the system context diagram as system input data flows. Controlled variables are identified on the system context diagram as system output data flows. The data dictionary entries associated with the data flows define the environmental variables. Some examples of environmental variable definitions are shown below.

```
WaterTemperature (data flow) =
    Temperature.
    ----------------------------
    Variable                WaterTemperature (MonitoredVariable)
    PhysicalInterpretation  The temperature of the water four feet below the
                            surface of the water in degrees centigrade.
    Type                    TEMPERATURE
    Values                  0 .. 100

WindDirection (data flow) =
    Direction.
    ----------------------------
    Variable                WindDirection (MonitoredVariable)
    PhysicalInterpretation  The direction the wind is blowing in degrees
                            (where 0 is due north and 90 is due east),
                            measured 10 feet above the surface of the water.
    Type                    DIRECTION
    Values                  0 <= WindDirection < 360 (0 = north, 90 = east,
                                                      180 = south, 270 = west)

Report (data flow) =
    Report_Type
    + ASCII_Report .
    ----------------------------
```

```
Variable                  Report (ControlledVariable)
PhysicalInterpretation
        while Report_Type=SOS_Report
            broadcasting data defined by DDE SOS_Data
        while Report_Type=Wind_and_Temperature_Report
            broadcasting data defined by DDE Wind_and_Temperature_Data
        while Report_Type=Weather_History_Report
            broadcasting data defined by DDE Weather_History_Data
        while Report_Type=Airplane_Detailed_Report
            broadcasting data defined by DDE Airplane_Detailed_Data
    .   while Report_Type=Ship_Detailed_Report
            broadcasting data defined by DDE Ship_Detailed_Data
        while Report_Type=None, no broadcasting
Type                      Enumeration + ASCII Text
Values                    for Enumeration, see PhysicalInterpretation
                          for ASCII Text, standard 8-bit ASCII character
                                        set
```

App.3.3.3 Dependency Graph

The CoRE dependency graph in Figure 13 was recorded using a *teamwork* level 0 data flow diagram. Classes are illustrated using data transformations. Terms and environmental variables on class interfaces are illustrated using data flows between transformations. Input and output variables are illustrated using external input and output data flows.

Data flow diagrams were also used to identify the IN, REQ, and OUT relations contained by each class. There is a lower level data flow diagram (see Figure 14) attached to each class on the dependency graph containing those relations.

App.3.3.4 Relations

IN, REQ, and OUT relations were defined using *teamwork* process activation tables. NAT relations were defined textually in the data dictionary. The following subsections contain examples of these relations.

App.3.3.4.1 IN Relations

Figure 15 illustrates an example of an IN relation for the monitored variable Wind (IN_Relation_for_Wind). WindVectorX and WindVectorY are terms defined in the data dictionary. WindSensors is the input variable used to derive the Wind monitored variable.

IN relations are also needed for the following monitored variables:

- Air_Temperature

- Water_Temperature

- Buoy_Location

- Vessel_Request

- Emergency_Button
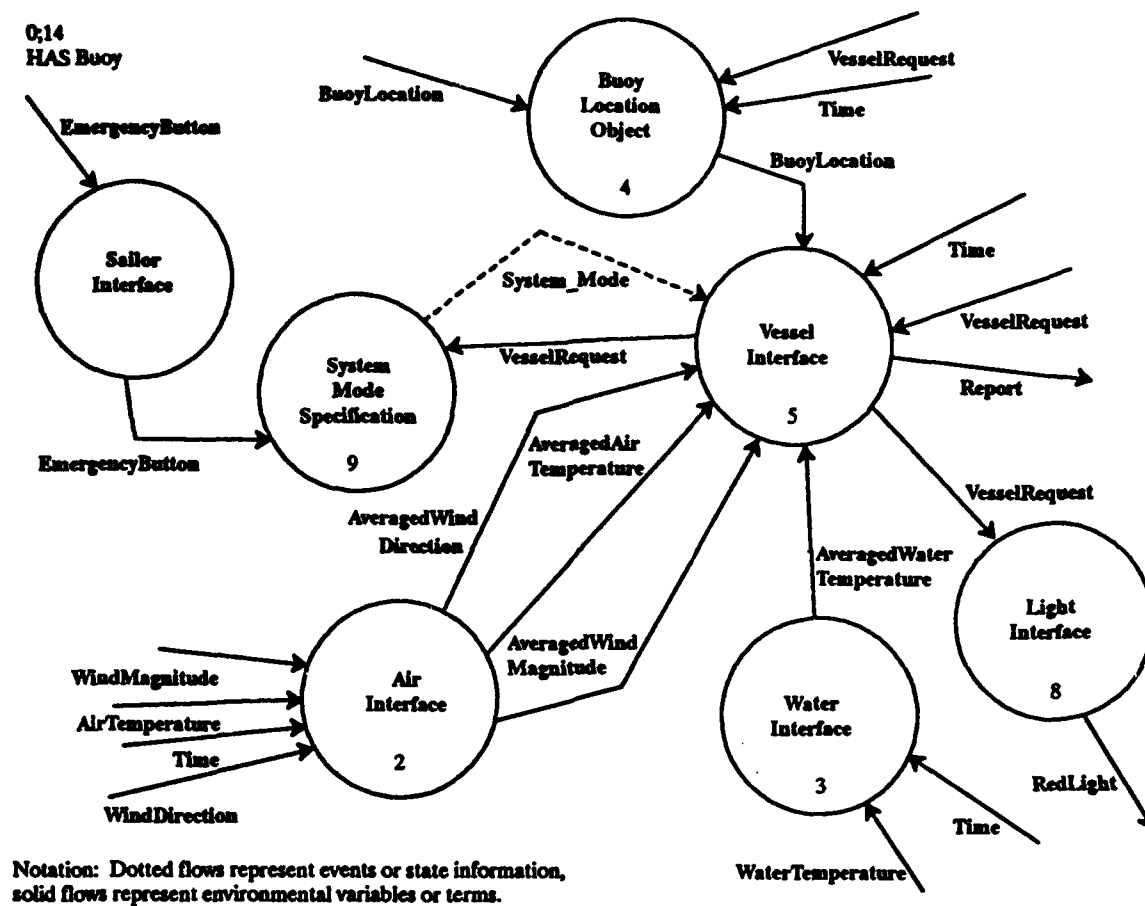
Notation: Dotted flows represent events or state information,
solid flows represent environmental variables or terms.
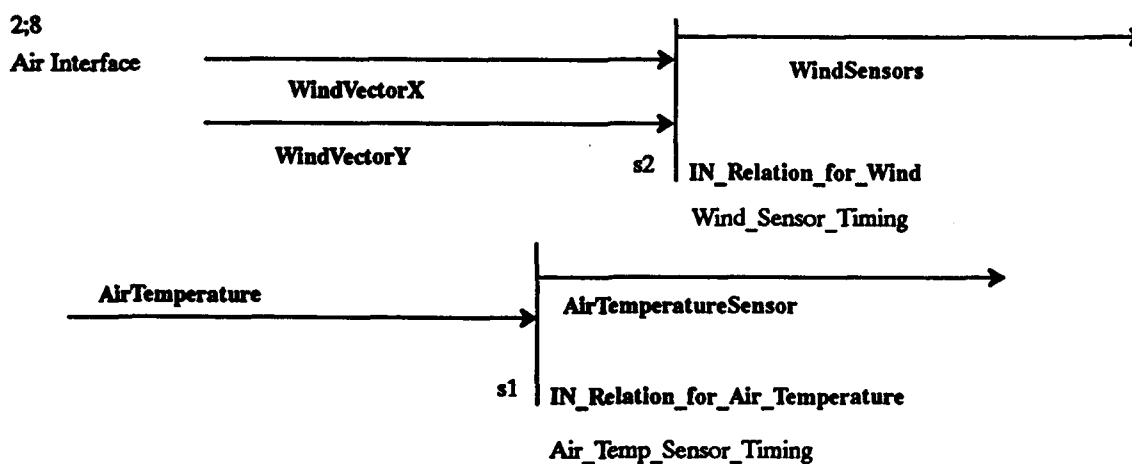
Figure 13. Dependency Graph

Figure 14. Example of a Lower Level Data Flow Diagram for a Class

2—s2;8
IN_Relation_for_Wind

Condition 1	Condition 2	WindSensors(C1) =	WindSensors(C2) =	WindSensors(C3) =	WindSensors(C4) =
WindVectorX >= 0	WindVectorY >= 0	WindVectorY	0	WindVectorX	0
WindVectorX >= 0	WindVectorY < 0	0	ABS(WindVectorY)	WindVectorX	0
WindVectorX < 0	WindVectorY >= 0	WindVectorY	0	0	ABS(WindVectorX)
WindVectorX < 0	WindVectorY < 0	0	ABS(WindVectorY)	0	ABS(WindVectorX)

Figure 15. Example of an IN Relation

App.3.3.4.2 REQ Relations

Figure 16 illustrates an example of a REQ relation (REQ_Relation_for_Report) that maps multiple monitored variables to the controlled variable Report.

5—s1;10
REQ_Relation_for_Report

Event	Report.Report_Type	Report.ASCII_Report
@T([Time MOD 60] = 0) when InMode(SOS)	"SOS_Report"	ASCII(SOS_Data)
@T([Time MOD 60] = 0) when InMode(Normal)	"Wind_and_Temperature_Report"	ASCII(Wind_and_Temperature_Data)
@T(VesselRequest = "Airplane_Detailed_Report_Request")	"Airplane_Detailed_Report"	ASCII(Airplane_Detailed_Data)
@T(VesselRequest = "Ship_Detailed_Report_Request")	"Ship_Detailed_Report"	ASCII(Ship_Detailed_Data)
@T(VesselRequest = "History_Report_Request")	"Weather_History_Report"	ASCII(Weather_History_Data)

Figure 16. Example of an REQ Relation

A REQ relation is also needed for the controlled variable RedLight.

App.3.3.4.3 OUT RELATIONS

Figure 17 illustrates an example of an OUT relation (OUT_Relation_for_RedLight). An OUT relation is also required for the controlled variable Report.

8—s2;6
OUT_Relation_for_RedLight

Light_Switch	RedLight
Light_On	"On"
Light_Off	"Off"

Figure 17. Example of an OUT Relation

App.3.3.4.4 NAT Relations

Examples of NAT relations for the HAS Buoy system are shown below:

```
-100 <= AirTemperature <= 100 (degrees centigrade)

  0 <= WindMagnitude  <= 250 (nautical miles per hour)
```

App.3.3.5 Terms

Terms were specified using data dictionary entries. Some examples of term definitions are shown below:

```
AveragedAirTemperature (data flow) =
    ─────────────────────────────────
    AveragedAirTemperature : TEMPERATURE
        := ROUND [ (SUM i: 0 <= i <= 5 :: AirTemperature(t-10*i) / 6 ) ]

Wind_and_Temperature_Data (data flow) =
    * Wind_and_Temperature can be expressed as a function of time t, where
      Wind_and_Temperature_Data (t) = the set including each of the
      following at time t: *

    { BuoyLocation
      + AveragedWaterTemperature
      + AveragedAirTemperature
      + AveragedWindDirection
      + AveragedWindMagnitude } .
```

App.3.3.6 Timing and Accuracy Constraints

Timing and accuracy constraints were specified using data dictionary entries attached to each IN, OUT, and REQ relation. Each relation has a body of text with the suffix "_Timing" attached to the appropriate team*work* process activation table. The data dictionary entries associated with these bodies of text specify timing and accuracy constraints. Examples are shown below:

```
Wind_Sensor_Timing (data flow) =
    Periodic (sensor_offset,
              30.0 second period,
              1.0 second delay).
    ─────────
    The wind sensor values are updated every 30 seconds, but the value
    is up to 1 second old by the time the software gets it.

Report_Timing (data flow) =
    Periodic (startup_offset + 30 seconds,
              60.0 second period,
              5.0 second delay)
        and
    Demand (startup_offset,
            report_request (Report_Type),
            reporting_delay (Report_Type)).
```

```
---------
    Reports are generated regularly, but special reports can also be
    requested.

reporting_delay (data flow) =
    when Airplane_Detailed_Report => 2.0 minutes;
    when     Ship_Detailed_Report => 5.0 minutes;
    when   Weather_History_Report => 6.0 minutes;
    ---------

    Each kind of report requires a different response time based on
    the time the platform will be in range.
```

App.3.4 INPUT AND OUTPUT VARIABLE DEFINITIONS

Input and output variables were specified using data dictionary entries. Some examples of input and output variable definitions are shown below:

```
AirTemperatureSensor (data flow) =
    ---------------------
    DataItem                AirTemperatureSensor
    Hardware                Air temperature sensor
    Values                  -128 <= AirTemperatureSensor <= 127
    DataTransfer            Port B
    DataRepresentation      8-bits, two's-complement integer

Button_Indicator (data flow) =
    ---------
    DataItem                Button_Indicator
    Hardware                Emergency button on buoy
    Values                  Pressed (1) or released (0)
    DataTransfer            Port E
    DataRepresentation      Most significant bit (bit 0) of the 8-bit port:
                                    2#1xxxxxxx# = Pressed,
                                    2#0xxxxxxx# = Released.

Outgoing_Radio_Message (data flow) =
    ---------
    DataItem                Outgoing_Radio_Message
    Hardware                Radio transmitter
    Values                  SOS_Report,
                            Wind_and_Temperature_Report,
                            Airplane_Detailed_Report,
                            Ship_Detailed_Report,
                            Weather_History_Report,
                            None.
    DataTransfer            Port G
    DataRepresentation   512 bytes:
        Byte 1: 2#10000001# means bytes 3-512 contain a page of an SOS_Report
                2#10000010# means bytes 3-512 contain a page of a
                    Wind_and_Temperature_Report
                2#10000011# means bytes 3-512 contain a page of an
                    Airplane_Detailed_Report
```

```
2#10000100# means bytes 3-512 contain a page of a
    Ship_Detailed_Report
2#10000101# means bytes 3-512 contain a page of a
    Weather_History_Report
2#0xxxxxxx# means None - no message is be transmitted (ignore
    bytes 2-512)
```
Byte 2: Bits 0-3:   4-bits range 1 .. 16 representing total number of
                            pages in message
          Bits 4-7:   4-bits range 1 .. 16 representing number of page
                            being transmitted
          Bytes 3-512:   Message represented by ASCII characters.
                            End of message represented by 16#FF#.

App.3.5 MODE CLASSES

Mode classes were defined using *teamwork* state-transition diagrams. There was only one mode class for the HAS Buoy problem (see Figure 18).

9—s1;14

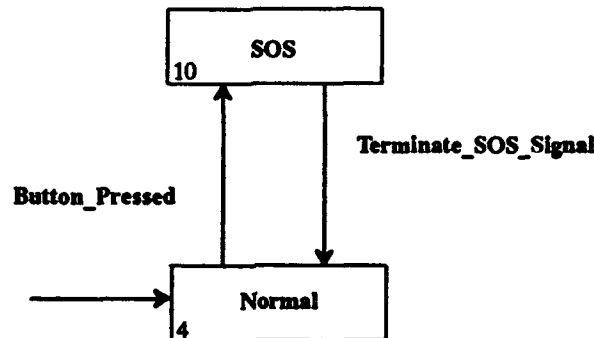Mode_Class_for_System_Mode

Figure 18. HAS Buoy Mode Class

LIST OF ABBREVIATIONS AND ACRONYMS

ADARTS   Ada-based Design Approach for Real-Time Systems

CASE   computer-aided software engineering

CDIF   CASE Data Interchange Format

CoRE   Consortium Requirements Engineering

ERA   entity-relationship-attribute

HAS   Host-at-Sea

IN   input (CoRE's abbreviation for "input" relations)

NAT   natural (CoRE's abbreviation for "natural" relations)

n.d.   no date

OUT   output (CoRE's abbreviation for "output" relations)

p-spec   process specification

RDD   Requirements Driven Design

REQ   required (CoRE's abbreviation for "required" relations)

SEN   System Engineering Notebook

This page intentionally left blank.

REFERENCES

Alford, Mack
n.d.

Behavior Based System Test Planning. San Jose, California: Ascent Logic Corporation.

Ascent Logic Corporation
1991a

RDD-100 Extensible Database Manager, release 3.0. San Jose, California: Ascent Logic Corporation.

1991b

RDD-100 System Engineering Notebook, release 3.0. San Jose, California: Ascent Logic Corporation.

1992a

RDD-100 User's Guide, release 3.0.2. San Jose, California: Ascent Logic Corporation.

1992b

RDD-100 Report Writer Manual, release 3.0. San Jose, California: Ascent Logic Corporation.

1993a

RDD-100 Customer Education. San Jose, California: Ascent Logic Corporation.

1993b

*RDD-100 Bridge to team*work. Beta 1. San Jose, California: Ascent Logic Corporation.

Cadre Technologies, Inc.
1990

*team*work *Toolkit Utilities User's Guide*, release 4.0. Providence, Rhode Island: Cadre Technologies, Inc.

Naval Research Laboratory
1980

Software Engineering Principles. Washington, D.C.: Naval Research Laboratory.

O'Rourke, Joel
1993

RDD-100: A System Engineering Support Tool. Review Draft, Rev. 02. San Jose, California: Ascent Logic Corporation.

Software Productivity
Consortium
1993

Consortium Requirements Engineering Guidebook, SPC-92060-CMC, version 01.00.09. Herndon, Virginia: Software Productivity Consortium.

This page intentionally left blank.