Strategies for Real-Time System Specification

by Derek J. Hatley
and Imtiaz A. Pirbhai

Foreword by Tom DeMarco

STRATEGIES FOR REAL-TIME SYSTEM SPECIFICATION

This page intentionally left blank

STRATEGIES FOR REAL-TIME SYSTEM SPECIFICATION

By

Derek J. Hatley   Imtiaz A. Pirbhai

Dorset House Publishing
353 West 12th Street
New York, New York 10014

Library of Congress Cataloging-in-Publication Data

```
Hatley, Derek J., 1934-
     Strategies for real-time system specification.

     Bibliography:  p.
     Includes index.
     1.  Real-time data processing.  2.  System design.
I. Pirbhai, Imtiaz A., 1953-             .  II.  Title.
QA76.54.H38   1987        004'.33                87-50801
ISBN 0-932633-04-8
```

CREDITS
Cover Design: Jeff Faville, Faville Graphics
Back Cover: Hatley photo: Bultman Studios, Inc.; Pirbhai photo: Stewart Auer.

The text of the book was set by the authors using the LaTEX document preparation system in cooperation with Smiths Industries. The graphics were prepared by the authors using Macintosh Plus and MacDraft.

 Published by Dorset House Publishing Co., Inc., 353 West 12th Street, New York, NY 10014.

Distributed in the United Kingdom, Ireland, Europe, and Africa by John Wiley & Sons Ltd., Chichester, Sussex, England.

Printed in the United States of America

Library of Congress Catalog Card Number 87-50801

ISBN: 0-932633-04-8

Digital release by Pearson Education, Inc., June, 2013

To Sue

DJH

To my father,
Baba Anwar Shah Taji,
and in memory of my mother

IAP

Acknowledgments

First and foremost, we thank Wendy Eakin, our editor and publisher, who gently but firmly led these two neophyte authors through the rigors of writing a book. You too should thank her, dear Reader: For most of our careers we have worked on or with engineering specifications and standards, and if you are at all familiar with those kinds of documents, you will understand that she is all that stands between you and total boredom.

Many people have contributed in many ways to the work presented here, and while we can mention only a few, we thank you all. Bruce Chubb, Angelo Dimitriou, Rex Morin, and Rod Wierenga gave management support for development and introduction of the requirements method. Dave Bulman sat in on the method's original conception, and, with his great knowledge and experience, helped us sift out the good ideas from the bad. The original Methods Team consisted of Kathy Hornbach, Bill Kelly, Martie Lorenz, John McCreary, and George Wood. Don Morrow remains the preeminent practitioner of the method itself, while John TenCate is master of the art of transforming the resulting model into a real system. Linda Goggins succeeded in the enormous task of coordinating and compiling the first (manual) application of the method. Kent McPherson performed some miracles with LaTeX, the document preparation system for this book, to make it work the way we wanted.

Tim Petersen, Irv Reese, Dick Schoenmann, and Peter Sutcliffe provided the management support needed for the development of the architecture method, as well as support for the further expansion and acceptance of both the requirements and architecture methods in the user community.

Dan Bacon, Rodney Bell, James Bouhana, Bob Doering, Peter Hruschka, Lou Mazzucchelli, Don Morrow, Rick Swanborg, Tony Wasserman, and Bob Weisickle reviewed various drafts of the book and improved it immeasurably with their insights.

Finally, we give a very special acknowledgment to Paul Gartz. Paul has been an enthusiastic champion of these methods, and an inspiration to the rest of us working on them. His aggressive canvassing on behalf of the methods has been responsible for their acceptance by management and the industry.

Contents

This page intentionally left blank

List of Figures

Foreword

"Most systems people use the term *real-time* rather loosely," the young manager said. We were seated over dinner with three members of her staff and some other managers who took part in the day's seminar. "They say they've got a real-time constraint when they're worried about impatient insurance brokers or bankers sitting in front of their terminals. A real-time system, in their minds, is just one that needs to be 'quick as a bunny.' If they fail to meet that constraint, their users might be inconvenienced or even annoyed. When we use the term, it means something rather different."

Her co-workers began to smile, knowing what was coming. "We build systems that reside in a small telemetry computer, equipped with all kinds of sensors to measure electromagnetic fields and changes in temperature, sound, and physical disturbance. We analyze these signals and transmit the results back to a remote computer over a wide-band channel. Our computer is at one end of a one-meter long bar and at the other end is a nuclear device. We drop them together down a big hole in the ground and when the device detonates, our computer collects data on the leading edge of the blast. The first two-and-a-quarter milliseconds after detonation are the most interesting. Of course, long before millisecond three, things have gone down hill badly for our little computer. We think of *that* as a real-time constraint."

I had been lecturing that day on the use of data flow modeling techniques for system specification. There had been the odd question about the use of such techniques for real-time systems, and my (typically glib) answer was that the techniques were applicable, though perhaps not totally sufficient. After all, my own earliest work with data flow modeling had been at Bell Labs and at La CEGOS Informatique, in both cases working on real-time systems. Or were those real-time systems? Maybe they were really just systems that needed to be 'quick as a bunny'? The young manager's graphic example had left me in some doubt.

It has always been clear that something more than the basic tools of structured analysis are needed to specify timing and synchronization requirements.

In the simplest case, that "something" could be as trivial as a set of textual annotations, perhaps directly on the data flow diagrams. But for even slightly more complicated cases, there is the possibility of linked timing and ordering constraints that may involve a dozen or more system components. There has been a need for a systematic way to deal with such constraints.

Over the last few years, I began to hear favorable reports of some real-time modeling techniques called the Hatley/Pirbhai Extensions. This multi-perspective approach combined data flow decomposition with model components constructed in control- and information-space. The result appealed to me as a specification technique that was not only applicable, but for most cases sufficient for real-time systems. I contacted the developers and started to try out their extensions.

The act of writing a Foreword is a kind of endorsement. It says, if nothing else, "Read this book, it couldn't hurt." In this case, I can be considerably more positive than that. I can tell you that I learned valuable new techniques from Hatley and Pirbhai, and that I apply them regularly on real-world real-time applications.

October 1987
Camden, Maine

Tom DeMarco
The Atlantic Systems Guild

Preface

This book describes two methods for specifying, respectively, the requirements for and the design structure of software-based systems. Although the methods grew up around real-time embedded systems, systems of all types and sizes have benefited from them, largely because of their flexibility and adaptability. The methods can be viewed as an integrated toolkit from which all the tools are compatible, but only those that are useful for the particular job need be used.

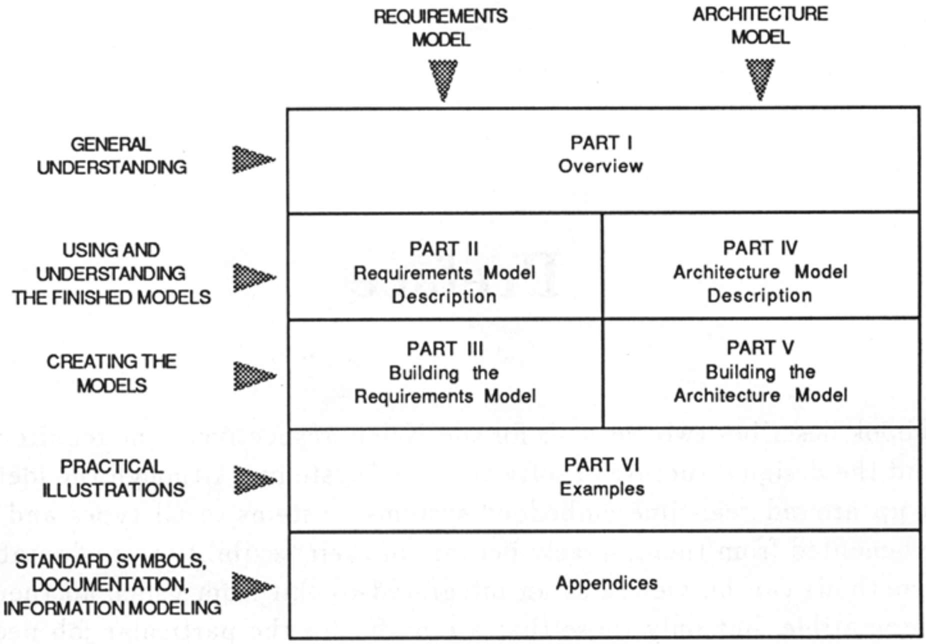
The subject of this book is neither computer science nor software. Although the vast majority of the systems to which the methods are applied will in fact be implemented using software in digital computers, the methods do not address how to write that software or how to design that hardware. What the methods do address is how to specify the *problems* that the hardware and software must solve.

Organization and audience of the book

The book is intended for a wide range of readers. We assume that you are involved, or at least interested, in software-based systems, but that your needs may range from just a general understanding to an in-depth working knowledge. We have tried to organize the book to make it easy for you to select those parts that are of specific interest to you and to avoid those that are not.

The figure on the following page shows the layout of the book. Part I provides an overview from which everyone will benefit. Parts II and IV describe *what* the methods are, and if you only need to understand the specifications resulting from their use, then these two parts are enough for you to read. If you need to *prepare* specifications, then you will also need to read Parts III and V, which describe *how* to use the methods. Part VI contains examples of the application of the methods, and we conclude the book with appendices and a brief bibliography.

Decide what your area of interest is—requirements analysis, design, or both—and what depth of understanding you need—general familiarity, use



A Map of the Book

of the completed specifications, or building the specifications—and it will be clear from the figure which parts of the book you should read.

Derek J. Hatley
Wyoming, Michigan

Imtiaz A. Pirbhai
Seattle, Washington

STRATEGIES FOR REAL-TIME SYSTEM SPECIFICATION

This page intentionally left blank

Chapter 3

Overview

Our strategy is to establish what the desired end product is, before describing how to produce it. Consequently, in Part II, we describe in detail the completed requirements model, as it would be received by the user, and in Part III, we describe *how* to construct the model. The principal tools of the requirements model are flow diagrams: graphical models of signal flows and processes acting on those flows. Flow diagrams consist of data flow diagrams and control flow diagrams. The former are essentially the same as the DFDs of DeMarco [4]; the latter are similar to DFDs but with some important and unique differences.

The other components of the requirements model are process specifications and the requirements dictionary (which are similar to DeMarco's mini-specifications and data dictionary), and control specifications (which are unique to this method).

You might wonder why we are describing here those parts of the method that have been described so often, and so well, before. The answer is, first, that we want the book to stand alone as a complete description of the whole method. Second, the earlier descriptions allowed a commendable degree of flexibility in the ways the method could be used. This has resulted in many different conventions being adopted in its use. It is therefore important to explain the particular conventions *we* have adopted in extending the method. If you have already been using "basic" structured analysis, you might very well find that some of our conventions are quite different from yours.

Key features of the method are

- It is a purely abstract model of the requirements, not a physical model of the design. It thus represents *what* has to be designed, not *how* to design it.

- It is diagrammatic, exploiting our ability to absorb visual information more effectively than textual or verbal information ("a picture is worth a thousand words").
- It hides unnecessary detail at any given level, and provides for progressive decomposition from that level down—the information hiding, abstraction, and decomposition principles put forward by Parnas [13]—thus allowing presentation of the requirements from a top-level overview down to the most intricate detail.
- It meets the needs of large, complex systems, including real-time systems, for representation of a process structure and a control structure. It does this by integrating DeMarco's structured analysis with finite state machine theory. Neither of the methods alone can meet these needs.
- It has built-in consistency checks within and between all its components. This feature of structured analysis has been extended to the whole structure, taking the guesswork out of the rigor and consistency of the model.
- It is self-indexing, another feature of structured analysis that has been successfully extended. It is easy to find your way around the resulting model without referring to page or paragraph numbers (although formal documentation requirements usually force us to keep these numbers).

The first of these features—the abstract nature of the model—deserves special emphasis, because it seems to be the one that newcomers to the method find hardest to grasp. Perhaps the best way to approach it is to understand the motivation for wanting such a feature, and this is best understood by reviewing the history of structured methods development.

Although structured methods generally represent their subject matter in a top-down fashion, the methods themselves have evolved in a bottom-up fashion. As software-based systems grew larger and more complex, the first obvious problem to be overcome was unmanageable and incomprehensible code. The need to solve this problem gave rise to structured programming in the form of constraints on module size, entry and exit points, branching, and so on. It also gave rise to high order languages, and, more recently, to program design languages.

Structured programming improved the quality of the code within individual modules, but still, as systems continued to grow in size and complexity, the interactions *between* the modules became as complex as the earlier code had been. Structured design methods, in which modules are constrained to communicate with each other in an organized way, were developed to deal with this complexity. These methods improved the quality of the software structure external to the modules, such that if both structured design and structured programming methods were conscientiously applied, the resulting system would work exactly as the designer intended.

Unfortunately, it still often happens that what the designer intended is not what the user wanted! The reason for this is that as the technology allows us to make systems that have ever larger capacity and complexity, the requirements themselves for such systems become large, complex, and difficult to describe and understand. Thus, the need was established for a method to represent the functional requirements of a system in an organized and understandable way, and in a way that is entirely independent of the eventual design and implementation of the system. Given such a model, various designs can be developed and compared without changing the requirements representation.

DeMarco's structured analysis method was designed to meet this need for data processing systems, and the extended structured analysis method described here was designed to meet the need for more general systems.

The model does, unfortunately, have a machine-like appearance; moreover, we are used to models that *are* intended to represent the actual design structure or implementation. As should now be clear, this is *not* the intent here. Again, it is a purely abstract model, with highly idealized properties, whereby each of its processes is independently data triggered and infinitely fast—unlikely characteristics for a real system. These properties are deliberately chosen for the very reason that we *can* ignore the constraints of the physical world, and concentrate exclusively on the required functionality.

As we build a requirements model, we follow this separation principle by including those characteristics that are of concern to the users, and deliberately excluding those that are not. The latter are left to the designer, and the moment we allow them to migrate into the requirements model, then we are including items that are *not* requirements; in so doing, we are placing unnecessary and artificial constraints on the designer.

We will remind you of this property at appropriate places throughout Parts II and III to emphasize its importance in each aspect of the model.

3.1 The Structure of the Model

Figure 3.1 is a composite chart, which enlarges on the diagram of Figure 2.7, and illustrates how the major components of the requirements model relate to each other. We will describe the major roles and relationships of these components before going into their individual details in the following chapters.

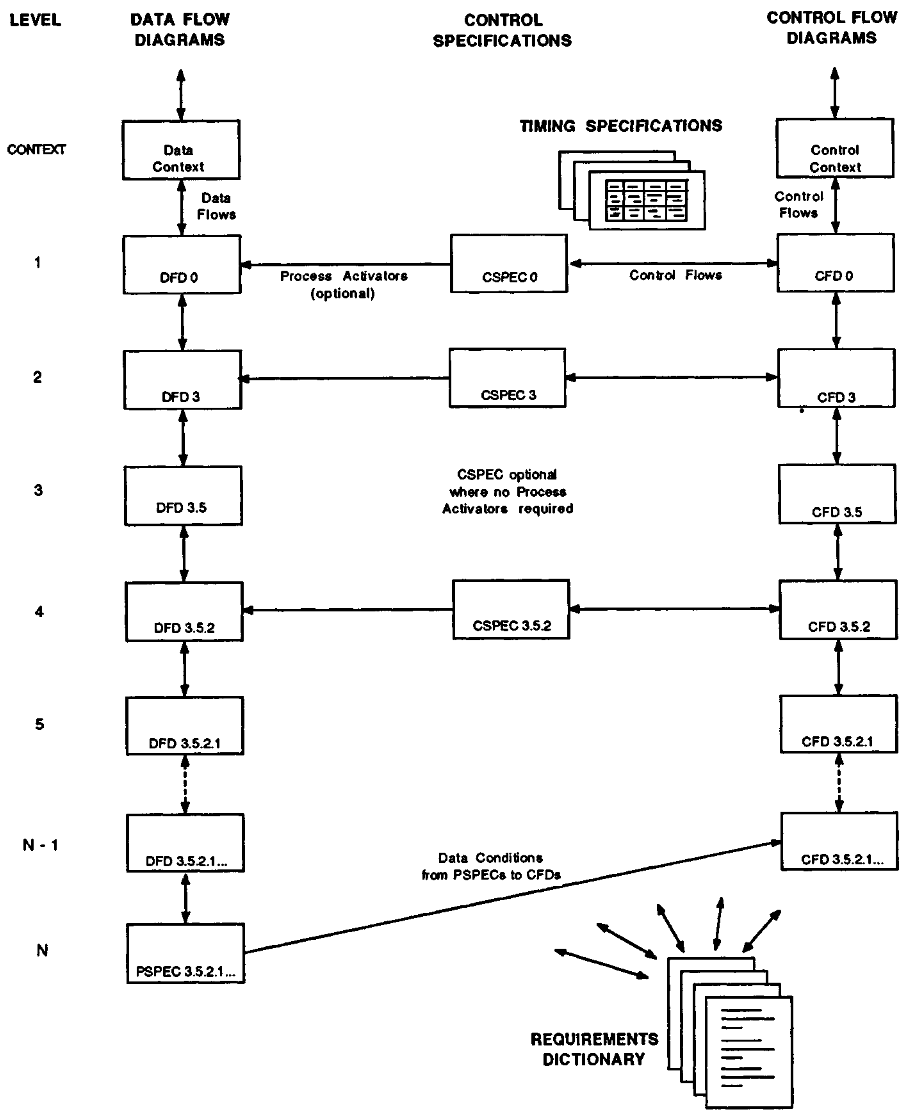


Figure 3.1. Composite chart of the requirements model.

At the top of Figure 3.1 are the data context diagram and the control context diagram. These show, respectively, the data and the control signals flowing between the system and the external entities with which the system must communicate. They also show the overall function of the system in the form of a single process. These diagrams represent the highest-level, or most abstract, view of the requirements, and show the central role the system is required to perform within its external environment.

Next (below the context diagrams in Figure 3.1) are two more flow diagrams—the level 1 DFD and CFD. They too represent the overall function of the system in graphical form, but in somewhat more detail than the context diagrams.

Below the level 1 diagrams are level 2 diagrams. For purposes of illustration, the figure shows just one DFD and one CFD at level 2, but, in fact, each of the several subprocesses on the level 1 diagrams may have its own level 2 DFD and CFD, so there will be several diagrams at this level. Just as the level 1 diagrams represent a more detailed statement of the system's purpose, so the level 2 diagrams represent more detailed statements of the purposes of the subprocesses on the level 1 diagram.

Similarly, level 3 diagrams decompose the processes and signal flows on the level 2 diagrams into more detailed statements, with a further multiplication of the number of diagrams. This decomposition continues level by level, each step revealing finer and finer details of the originally stated system's purpose.

The decomposition ends, for any given subprocess on the DFD side, with a concise statement of the purpose of a process using simple text, equations, tables, or diagrams. This statement constitutes a process specification, as illustrated at the bottom of the column of DFDs. A process that is described by a process specification is called a functional primitive, or just a primitive.

On the CFD side, the decomposition parallels that of the DFDs, but some of the control signals flow to and from control specifications, shown in the center column of Figure 3.1. The CSPECs describe the finite state machine functions of the system, which we will discuss in Chapter 6. Each CSPEC is associated with a specific CFD—the one adjacent to it in the diagram. It will usually control processes, but only those in the specific DFD adjacent to it in the figure. This close association of a DFD, CFD, and CSPEC is an important part of the structure and self-consistency properties of the requirements model.

The system input-to-output timing requirements are shown in the timing specification, shown between the context and level 1 diagrams (and detailed in Chapter 7). These specifications simply list events that are detected at the system inputs (the system stimuli), the corresponding events that are required to occur at the system outputs (the system responses), and the timing constraints within which the system must generate these responses.

Finally, Figure 3.1 shows the requirements dictionary. Although it is shown outside the main structure, the RD serves a vital role throughout the model since it contains precise definitions of all the data and control flows in all their levels of decomposition.

The remaining chapters of Part II describe each component of the method in detail, but it is important not to lose track of how they fit into the overall structure of Figure 3.1 as you study them.

For the purposes of this description, we refer to the DFDs and PSPECs as the *process model,* the CFDs and CSPECs as the *control model,* and the two of them with the requirements dictionary and timing specifications as the *requirements model.* This is in keeping with the evolution of the methods: The process model corresponds to the classical structured analysis model; the control model is the part we added, derived from finite state machine theory; and the requirements model is the complete, integrated combination of the two.

Index