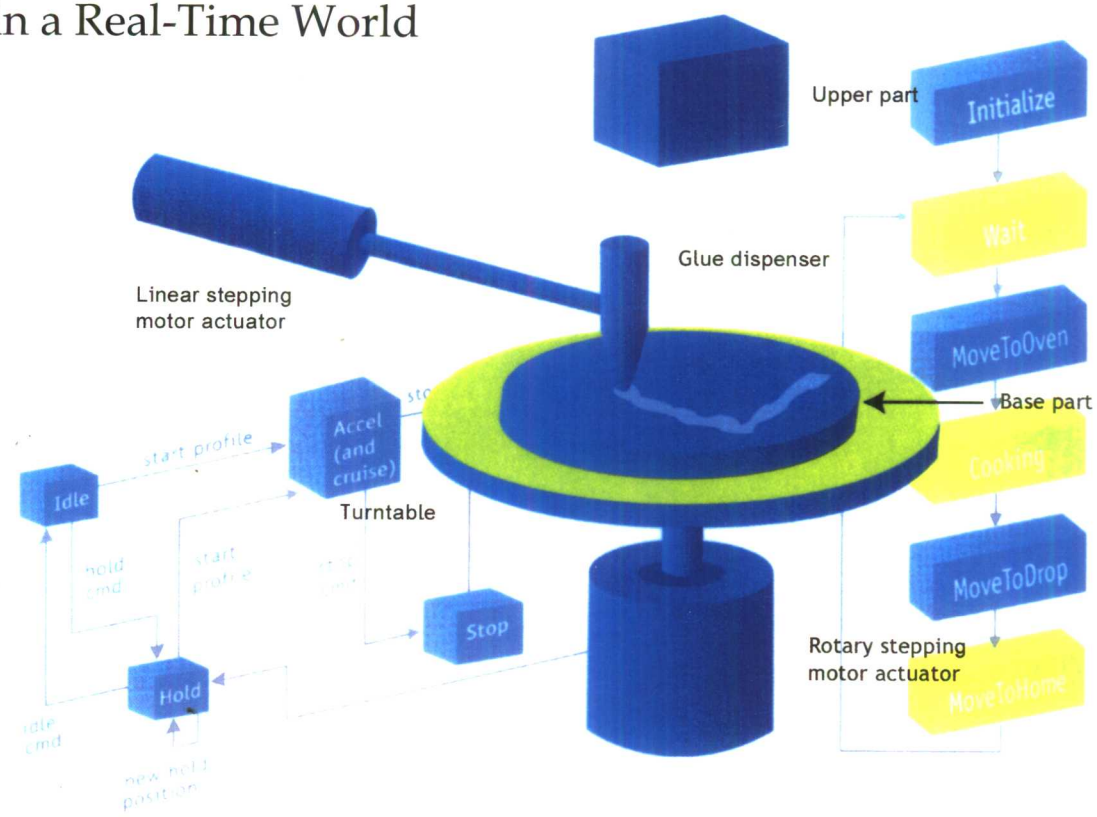
机械系统控制软件

——实时系统面向对象设计（影印版）

Control Software for Mechanical Systems:
Object-Oriented Design in a Real-Time World



[美] D.M. Auslander，J.R. Ridgely，J.D. Ringgenberg 著

Pearson Education 清华大学出版社

机械系统控制软件

——实时系统面向对象设计

（影印版）

Control Software for Mechanical Systems

Object-Oriented Design in a Real-Time World

D.M. Auslander
J.R. Ridgely
J.D. Ringgenberg
著

清华大学出版社

北京

图书在版编目(CIP)数据

机械系统控制软件：实时系统面向对象设计=Control Software for Mechanical Systems: Object-Oriented Design in a Real-Time World/（美）奥斯兰德，（美）瑞吉利，（美）瑞根伯格著. —影印本. —北京：清华大学出版社，2004.4
ISBN 7-302-08233-2

I. 机… II. ①奥… ②瑞… ③瑞… III. 机械工程—控制系统—应用软件—英文 IV. TP273

中国版本图书馆 CIP 数据核字（2004）第 017390 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社总机：010-6277 0175　　**客户服务**：010-6277 6969
责任编辑：郭福生　常晓波
封面设计：立日新
印 刷 者：清华大学印刷厂
装 订 者：三河市金元装订厂
发 行 者：新华书店总店北京发行所
开　　本：180×230　**印 张**：22.25
版　　次：2004 年 4 月第 1 版　2004 年 4 月第 1 次印刷
书　　号：ISBN 7-302-08233-2/TP • 5940
印　　数：1~3000
定　　价：36.00 元

前　言

在工程课程中，通常没有包括关于复杂机械系统的控制系统的内容，然而，这一主题在工业控制领域占有极其重要的位置。大多数机械课程和专业参考书都会详细介绍微型系统(使用微控制器)的嵌入式控制系统，也会详细介绍电子学和控制理论方面的内容。与众不同的是，本书只阐述与机械系统控制软件设计有关的问题，这些机械系统包含大量的传感器和致动器，人们必须协调这些部件的动作。这些机械系统通常包含一个基于计算机的操作界面，还包含有内部的、外部的网络连接。

在本书中，“机械系统”指的是必须有物理能量传递给目标物体的系统。这种能量的形式可以是动能、热能、力、压力，等等，十分广泛。所谓的“复杂系统”，是指“独立成套的机器”，即各部件之间经过很少的缓冲或完全没有缓冲而直接交换物理能量或物料的系统。这一定义，一方面是为了区分具体的机械系统和过程的控制，另一方面也是为了区分工作单元、整个工厂和工厂各部门的控制软件的设计。

本书的内容是根据本科生和研究生的课程编写的，可以供机械工程系的大多数学生使用。机械工程系的学生通常没有编程经验，因此，他们至少必须先行修完一门编程语言的课程，还必须学习有关设计方法的课程，并具备将这些知识应用于实验练习和项目的能力。虽然用 15 周的时间修完所有这些预备课程，时间有些紧，但学生对软件的内部工作原理似乎已经有了很好的理解。对于大多数学生而言，直接成功地进入本书介绍的工业应用领域是一个了不起的成就。本书也可用于自学，或者重点讲述设计或成功项目的跨学期课程。

本书的内容是从嵌入到设计层的控制工程理论的角度来讲述的，而不是从计算机代码的角度来介绍的。因此，我们重点讲述了可移植性、设计评审和内部通信问题。可运行的计算机代码的生成是一个独立的过程，这里要考虑实时约束问题、计算机软硬件成本问题以及维护的便利性问题。保持这种独立，是“机械系统控制软件的运行必须是一个可预测的工程过程”这一思想的关键。

本书介绍的设计规范所采用的方法，是以有限状态机和半独立任务为基础的。实践证明，用这种方法来解决机械系统的复杂度问题是适合的。这种方法中包含有一些更正规的方法(如 UML)中的某些常用的元素，但更加简单易学。方法的选择，主要是考虑它是否适合以能使广大读者易于理解的方式来描述控制行为，同时也要考虑是否便于使用计算机在任何语言或环境中进行实现。本书始终强调软件可移植性对于最大限度地保护软件投资的重要性。

在选择计算机语言时，我们将设计层作为本书的重点，而实际的实现则尽可能不拘泥于某一种语言。尽管如此，我们还是有必要选择一种语言来实现实际的控制系统。虽然 C 语言可能是目前最流行的语言，但在构建可重用的、使设计模型尽可能显而易见的软件基础结构方面，C++和 Java 对我们的帮助更大。因此，本书使用 C++和 Java 来实现实际的控制软件，书中的所有示例程序也是用 C++和 Java 编写的。由于无需向后兼容，Java 在可移植性和语言的清晰方面要优于 C++。例如，Java 的语法中包括图形用户界面(GUI)结构和 TCP/IP 网络结构，而 C++语法中不包括这些结构，因此，对于 C++程序来说，这些结构是不可移植的。Java 的类文件在任何支持 Java 的平台上都是可执行的，这大大简化了跨平台开发。但在另一方面，由于在执行时采用了虚拟机，Java 程序的运行速度相当慢；而在高速的实时环境中，像垃圾回收这样的特性的应用，需要很高的技巧。

然而，基本的编程模型稍加修改就可以用于其他编程语言。书中提供了使用 Matlab 进行模拟的示例软件，而其在 C 语言或其他纯算法语言中的特定实现，构建起来并不难。

我们提供了三个主要的软件包，用于支持开发，其中两个是用 C++开发的，另一个是用 Java 开发的。这三个软件包都支持基本的任务/状态设计模型，都可以在支持模拟、原型设计、甚至软件的生产版本的各种环境中运行。一个 C++软件包采用了一种非常简单的调度模块，所以，关于任务应该何时执行这种决策性任务大部分留给读者去完成。另外一个 C++软件包和 Java 软件包，支持一个功能更加丰富的调度模块。比较简单的那个 C++软件包和 Java 软件包，都以这样一种方式完全支持 TCP/IP 网络：允许通过十分简单的配置修改在多台计算机之间分配任务，而无须在控制软件中进行任何修改。

Windows NT 系列操作系统(NT、2000 和 XP)作为主要的开发环境，已经被人们使用了好多年。作为可用于许多目标环境的跨平台编译器，C++和 Java 编译器也是很容易得到的。在进行初期的原型设计和调试时，也要使用 C++和 Java 编译器作为实时执行环境。然而，各种 Windows 环境中的计时精度达不到大多数机械系统控制所必需的亚毫秒级精度，所以，为了获得更高质量的实现，必须使用其他的操作系统(尽管如果离开了 Windows 我们还能够完成多少工作，这一点会令人非常惊讶)。过去的几年我们一直使用 DOS、Windows NT RTX (来自 VenturCom 公司)和 QNX 作为实时平台[1]。目前，我们使用 Java 进行开发，而使用 QNX 作为实时平台，这是因为 DOS 和 NT/RTX 都不能有效地支持实时 Java。

对于许多商业性的控制系统，操作界面是其非常重要的组成部分。操作界面

[1] 从技术上讲，DOS 并不是一个实时平台，但它与我们开发的实时结构之间没有接口，因此，与 Windows 相比，其实时调度性能要高得多。

不仅对于目标系统的有效使用非常重要，而且通常会影响到安全性。尽管关于影响操作界面设计的人类因素和其他规范的全面讨论已经超出了本书的范围，但我们还是讨论了构建基于计算机的操作界面的几种方法。关于操作界面的示例我们使用 Bridgeview (Labview 的姊妹软件，由 National Instruments 公司开发)和 Visual Basic 来设计。从效率的角度来讲，这些方法和其他的简化 GUI 构建方法是非常重要的，控制工程师常常使用这些方法来构建原型操作界面，而不必将过多的时间浪费在过程之中。本书没有专门讨论如何用 Java 来构建 GUI，但提供了一个示例，用来演示如何使用 Java 的 Swing 组件快速开发 GUI。

本书提供了一个详细的案例分析，以演示如何使用本书中讨论的设计与实现方法，解决合理的复杂度问题。这完全是一个模拟实现，所以可以在任何安装了 Java 或 C++的平台上运行，因此，无论是课堂教学还是自学，这个案例都是一个很好的学习工具。针对我们开发的三种软件基础结构，我们提供了三种实现。

本书中使用的所有软件包和示例软件，可以从下列任何一位作者的站点下载：

```
www.me.berkley.edu/~dma
```

其他的 Java 信息可以从另一位作者的站点下载：

```
www.ugcs.caltech.edu/~joeringg/TranRunJ
```

还有相当多的教学资料可以从基于本书的课程站点下载：

```
www.me.berkley.edu/ME135
www.me.berkley.edu/ME230
```

其中包含每门课程的最新教学大纲、实验作业、教学笔记和其他相关信息。上述站点还提供大量课堂用 PowerPoint 演示文稿；也可以通过下列邮件地址请求演示文稿：

```
dma@me.berkley.edu
```

PREFACE

The control of complex mechanical systems often falls between the cracks of engineering curricula, but the topic occupies an extremely important place in the world of industrial control. Most courses and professional reference material cover the details of embedded control of very small systems using microcontrollers, as well as details of electronics and control theory. In contrast, this book addresses issues associated with the design of control software for mechanical systems consisting of significant numbers of sensors and actuators – systems whose activities must be coordinated. These systems often include a computer-based operator interface and internal as well as external network connections.

The term "mechanical system" in the context used here refers to a system in which real physical power must be delivered to a target object. The power could be in a form such as motion, heat, force, pressure, or many others, so the range of applicability is very large. The domain of complexity encompasses what we call a "unit machine," a mechanical system in which all parts either exchange physical power directly or exchange material with little or no buffering. This definition makes a distinction between the control of a specific unit machine or process on the one hand, and designing control software for work cells, entire process plants, or sections of a factory on the other.

The material we present was developed for teaching both undergraduate and graduate courses to classes consisting mostly of mechanical engineering students. Mechanical engineering students are often not experienced in computer programming and therefore have to learn an appropriate programming language (more on that in a moment), learn the design methodology, and develop the ability to apply both in a series of lab exercises and projects. While a 15 week semester makes for a somewhat compressed timetable to do all of this, the students seem to emerge with a good understanding of how the pieces fit together. It is not unusual for many students to progress directly to successful work in industrial applications based mostly on the exposure in this course. The material is usable in other configurations as well, including self study, multi-semester/quarter format with more emphasis on design or capstone projects, and so on.

The presentation in this text is based on a philosophy in which the control engineering insight is embedded in a design layer rather than in computer code. Thus, issues of portability, design review, and internal communication can be stressed. The generation of functioning computer code is a separate process where considerations of real time constraints, costs of computing hardware and software, and ease of maintenance can be considered. Maintaining this separation is crucial to the idea that production of control software for mechanical systems must be a predictable engineering process.

The methodology we use for design specification is based on finite state machines and semi-independent tasks. It has proved to be a methodology capable of handling the level of complexity needed for these systems. It has some common elements with more formal methodologies such as the universal modeling language (UML) but is considerably simpler. The choice of methodology is based on its ability to describe control behavior in a way that is easily explainable to a broad audience, as well as the complementary property of easy hand translation for computer implementation in almost any language or environment. Software portability to achieve maximum protection to software investment is emphasized throughout.

Choosing computer languages provokes as much vehement discussion as religion or politics! To neutralize some of that partisanship, we have kept the design layer as primary, with the actual software implementation kept as fluid as possible. Nonetheless, it is necessary to choose some language in which to implement actual control systems. While C is probably the most common language in use today, the structures of C++ and Java provide us with more assistance in building a reusable software infrastructure that makes the design model as obvious as possible. We have therefore used C++ and Java for most of the actual control implementation and for the example programs used throughout this text. Java has the advantage of greater portability and is a cleaner language than C++ because it does not need backward compatibility with C. Its syntax includes graphic user interface (GUI) constructs and TCP/IP networking, for example, which are not included in the C++ syntax and are thus not portable for C++ applications. Java's class files are executable on any platform supporting Java, so cross-development is greatly simplified. On the other hand, Java is considerably slower by virtue of its use of a virtual machine for execution; and features such as garbage collection are tricky to deal with in a high speed, real time environment.

The basic programming model, however, is easily adapted for other programming languages. Sample software for simulation using Matlab is given in the book and *ad hoc* implementations in C or other purely algorithmic languages are easily constructed.

We have three main software packages available to support development, two in C++ and one in Java. All of these support the basic task/state design model, and all can operate in a variety of environments to support simulation, prototyping, and even production versions of software. The two C++ packages differ in that one uses a much simpler scheduling module, so more of the decision-making about when tasks should execute is left to the programmer. The other C++ package, and the

Java package, support a richer scheduling module. The simpler C++ package and the Java package both support full TCP/IP networking in a manner that allows tasks to be distributed over multiple computers through very simple configuration changes, with no other changes needed in the control software.

The Windows NT family of operating systems (NT, 2000, and XP) have been used for a number of years as the main development environment. C++ and Java compilers are readily available as are cross-compilers for many target environments. They have also been used as real time execution environments for early prototyping and debugging. However, the timing in any of the Windows environments is not consistent to the sub-millisecond level needed for much mechanical system control, so other operating systems must be used for higher quality implementation (although it is surprising how much can actually be accomplished without leaving Windows). In the past several years we have used DOS, Windows NT modified with RTX (from VenturCom), and QNX as real time platforms.[1] We are currently working with Java and using QNX as the real time platform, since neither DOS nor NT/RTX supports real-time Java effectively.

The operator interface is a critical part of many commercial control systems. Not only does the interface contribute to efficient use of the target system, but it often affects safety in critical ways. While a full discussion of human factors and other disciplines affecting operator interface design is beyond the scope of this book, we do discuss several means of constructing computer-based operator interfaces. The book shows samples based on using Bridgeview (an industrial relative of Labview, from National Instruments) and Visual Basic. These and other simplified GUI construction methods are important from an efficiency point of view in that they can often be used by control engineers to construct prototype operator interfaces without having to devote inordinate amounts of time to the process. The construction of GUIs in Java is not specifically discussed in the book, but available sample Java code demonstrates the use of Java's Swing components in rapid GUI development.

A detailed case study is provided to illustrate the use of the design and implementation methodology on a problem with reasonable complexity. The implementation is entirely a simulation, so it can be run anywhere Java or C++ can be installed – and thus it makes a good vehicle for use by a class or as a self-study example. Implementations are given for all three of the software infrastructures we have developed.

All of the software packages and sample software supporting the book are available from the web site of one of the authors:

`www.me.berkeley.edu/~dma`

Additional Java information is available from the web site of another author:

`www.ugcs.caltech.edu/~joeringg/TranRunJ/`

[1] DOS is not technically a real time platform, but it does not interfere with the real time constructs we have created and thus allows much higher performance real-time scheduling than Windows.

There is considerable teaching material available on the class web sites for the courses based on this book:

`www.me.berkeley.edu/ME135` and `www.me.berkeley.edu/ME230`

This includes a current syllabus for each of these courses, lab assignments, notes, and other relevant information. There is also a large amount of class presentation material in PowerPoint format available; this can be requested by sending an email to `dma@me.berkeley.edu`.

Contents

1 MECHATRONICS 1

1.1 A History of Increasing Complexity 2
1.2 Mechatronic System Organization 3
1.3 Amplifiers and Isolation 3
1.4 Scope: The Unit Machine 4
1.5 Control 5
1.6 Real-Time Software 5
1.7 Nasty Software Properties 8
1.8 Engineering Design and Computational Performance 9
1.9 Control System Organization 10
1.10 Software Portability 10
1.11 Operator Interface 11
1.12 Multicomputer Systems: Communication 12
1.13 The Design and Implementation Process 12
1.13.1 Performance Specification 12
1.13.2 Design Documentation 13
1.13.3 Simulation 14
1.13.4 Laboratory Prototype 15
1.13.5 Production Prototype 16
1.13.6 Production System 17
1.13.7 Maintenance 17

2 TASKS **19**

2.1 Example: Task Selection in a Process System 20
2.2 Tasks and the Control Hierarchy 21
2.2.1 Intertask Communication 22
2.3 Task Structure Examples 22
2.3.1 Velocity Control of a DC Motor 23
2.3.2 Heater Control 24
2.3.3 Toaster Oven 26
2.3.4 Flexible Position Control of a DC Motor 27

2.4 Simulation 29
2.5 More Task Structure Examples 31
2.5.1 Coordinated, Two-Axis Motion 31
2.5.2 A Washing Machine 32

3 STATE TRANSITION LOGIC 35
3.1 States and Transitions 36
3.2 Transition Logic Diagrams 36
3.3 Tabular Form for Transition Logic 37
3.4 Example: Pulse-Width Modulation (PWM) 38
3.5 Transition Logic for the Process Control Example 39
3.6 Nonblocking State Code 41
3.7 State-Related Code 41
3.8 State Scanning: The Execution Cycle 42
3.9 Task Concurrency: Universal Real-Time Solution 43

4 DIRECT REALIZATION OF SYSTEM CONTROL SOFTWARE 45
4.1 Language 45
4.2 Time 47
4.3 Program Format 47
4.4 Simulation 48
4.5 Simulation in Matlab 48
4.5.1 Templates for Simulation Using Matlab 48
4.5.2 Simulation of PWM Generator 53
4.5.3 Simulation of Three-Tank Process System 57
4.6 Intertask Communication 61
4.7 Real-Time Realization 62
4.8 Real-Time Realization with Matlab 62
4.8.1 Heater Control Implementation in Matlab 63

5 SOFTWARE REALIZATION IN C++ 67
5.1 Simulation in C++ 67
5.2 Templates for Simulation in C++ (group-priority) 68
5.3 PWM Simulation Using C++ (group-priority) 80
5.4 Simulation in C++ (with TranRun4) 82
5.4.1 Components 82
5.4.2 The Master Scheduler 84
5.4.3 Process Objects and Task Lists 85
5.4.4 Task Objects 86
5.4.5 Tasks with No State Object 89
5.4.6 Creating Task Classes 89
5.4.7 State Objects 91
5.4.8 Creating State Classes 93
5.4.9 The Main File and UserMain() Function 94
5.5 Real-Time Realization with C++ 97

6 INTERTASK COMMUNICATION **99**
6.1 Communication Within a Process 100
6.1.1 Data Integrity 100
6.1.2 Design Rules 102
6.2 Communication Across Processes 105
6.2.1 Message Passing 105
6.2.2 Message Passing in the Group Priority Scheduler 106
6.2.3 Message Passing in the TranRun4 Scheduler 112
6.2.4 Distributed Database 115
6.2.5 Distributed Database in the Group Priority Scheduler 116
6.2.6 Distributed Database in the TranRun4 Scheduler 118

7 TIMING TECHNIQUES ON PC COMPATIBLES **121**
7.1 Calibrated Time 121
7.2 Free-Running Timer 122
7.2.1 Hardware Timers on the PC 123
7.2.2 Performance Timers in Unix and Windows 124
7.3 Interrupt-Based Timing 125

8 MULTITASKING: PERFORMANCE IN THE REAL WORLD **127**
8.1 Priority-Based Scheduling—Resource Shifting 127
8.1.1 Continuous vs. Intermittent Tasks 128
8.1.2 Cooperative Multitasking Modes 129
8.2 Matlab Template for Minimum-Latency Dispatcher 131
8.2.1 Example: Simulation of PWM-Actuated Heater 131
8.3 Cooperative Multitasking Using C++ 133
8.3.1 Inheriting Task Behavior—Two PWMs 137
8.4 Preemptive Multitasking Modes 138
8.5 Realization of Interrupt-Based Dispatching 140
8.5.1 How Many Priority Levels Are Necessary? 141
8.5.2 Which Interrupt Sources Will Be Used? 141
8.5.3 Interrupt-Based Dispatching Functions 142
8.5.4 Attaching Dispatching Functions to Interrupts 143

9 A CHARACTER-BASED OPERATOR INTERFACE **145**
9.1 Operator Interface Requirements 145
9.2 Context Sensitive Interfaces 146
9.3 User Interface Programming Paradigms 147
9.4 Mechatronics System Operator Interface 147
9.5 Operator Interface Programming 148
9.5.1 The Operator Screen 148
9.5.2 Programming Conventions in C++ 149
9.5.3 Heater Control Operator Interface 151

10 GRAPHICAL OPERATOR INTERFACES 155
10.1 Graphical Environments 156
10.1.1 Windowing Software: Events and Messages 156
10.1.2 Operator Interface vs. Standard Windowing Application 157
10.1.3 Simplified Programming for Windowing Systems 157
10.1.4 The Ease-of-Use Challenge 158
10.1.5 Methods of Simplifying Window-Style Programming 158
10.2 The Times-2 Problem 158
10.2.1 Times-2: Character-Based Interface 158
10.2.2 Times-2: Visual Basic 160
10.2.3 Times-2: Bridgeview 162
10.3 Screen Change 166
10.3.1 Screen Change in Visual Basic 166
10.3.2 Screen Change: Bridgeview 168
10.4 Heat Exchanger Control in Bridgeview 171
10.5 Interprocess Communication: DDE 173
10.5.1 DDE: The C++ Side 173
10.5.2 Communicating with Excel 176
10.5.3 A DDE Server in C++ 177
10.5.4 DDE Communication Between C++ and Visual Basic 178
10.5.5 DDE Communication Between C++ and Bridgeview 180
10.6 Putting It All Together 181

11 DISTRIBUTED CONTROL I: NET BASICS 185
11.1 Multiprocessor Architectures 186
11.1.1 Symmetric Multiprocessing (SMP) 186
11.1.2 Buses 186
11.1.3 Networks 187
11.1.4 Point-to-Point Connections 189
11.2 TCP/IP Networking 190
11.2.1 The Physical Context 190
11.2.2 Interconnection Protocols 191
11.2.3 TCP and UDP 191
11.2.4 Client/Server Architecture 192
11.3 Implementation of UDP 192
11.3.1 Sockets 192
11.3.2 Setting Up for Network Data Exchange 193
11.3.3 Nonblocking Network Calls 195
11.3.4 Receiving Information 195
11.3.5 Client-Side Setup 196
11.4 The Application Layer 197
11.4.1 Data Coding 197
11.4.2 Building the Packet 198
11.4.3 Parsing a Packet 200

12 DISTRIBUTED CONTROL II: A MECHATRONICS CONTROL APPLICATION LAYER **203**
12.1 Control System Application Protocol 203
12.2 Startup of Distributed Control Systems 207
12.3 Testing the Application Protocol 208
12.4 Using the Control Application Protocol 209
12.5 Compiling 212

13 JAVA FOR CONTROL SYSTEM SOFTWARE **213**
13.1 The Java Language and API 214
13.1.1 Networking 214
13.1.2 AWT/Swing 214
13.1.3 Multithreading 214
13.2 Preconditions for Real-Time Programming in Java 215
13.2.1 Deterministic Garbage Collection 215
13.2.2 Memory and Hardware Access 215
13.2.3 Timing 216
13.3 Advantages of Java for Control Software Design 216
13.3.1 Modularity 216
13.3.2 Distributed Control 217
13.3.3 Platform Independence and Prototyping 217
13.3.4 Operator Interface Design 217
13.4 Java and the Task/State Design Method 218
13.4.1 Inner Classes 218
13.4.2 Networking 218
13.4.3 Documentation 219
13.5 The Current State of Real-Time Java 219

14 PROGRAMMABLE LOGIC CONTROLLERS (PLCs) **221**
14.1 Introduction 221
14.2 Goals 222
14.3 PLC Programming 223
14.3.1 When to Use a PLC 223
14.3.2 Ladder Logic 224
14.3.3 Grafcet/Sequential Flow Charts 226
14.4 The Task/State Model 226
14.5 State Transition Logic for a PLC 227
14.5.1 State Variables 227
14.5.2 Ladder Organization 227
14.5.3 Transitions 228
14.5.4 Outputs 229
14.5.5 Entry Activity 229
14.5.6 Action Outputs 229
14.5.7 Exit (Transition-Based) Outputs 229
14.5.8 Common Exit Activities 230

14.6 PLC Multitasking 230
14.7 Modular Design 231
14.8 Example: Model Railroad Control 231
14.9 Simulation – Portability 232

15 ILLUSTRATIVE EXAMPLE: ASSEMBLY SYSTEM 235
15.1 The Assembly System 235
15.2 System Simulation 237
15.3 Development Sequence 237
15.4 Belt Motion Simulation (Glue00) 238
15.4.1 Modeling Belt Dynamics 238
15.4.2 Definition of Task Classes 239
15.4.3 Instantiating Tasks: the Main File 241
15.4.4 The Simulation Task 242
15.4.5 The Data Logging Task 244
15.4.6 Timing Mode 245
15.4.7 Compiling 246
15.4.8 Results 246
15.5 Oven Temperature Simulation (Glue01) 247
15.6 PID Control of Belt Position and Oven Temperature (Glue02) 247
15.6.1 Keeping Classes Generic 248
15.6.2 The PIDControl Class 248
15.6.3 Results 250
15.7 Better Control of Motion (Glue03) 250
15.7.1 Trapezoidal Motion Profile 251
15.7.2 Motion Profile Class 252
15.7.3 Profiler State Structure 253
15.7.4 Round-Off Error 257
15.7.5 Discretization Errors in Simulation 257
15.8 A Command Structure for Profiled Motion (Glue04) 260
15.8.1 Message-Based Command Structure 260
15.8.2 State Transition Audit Trail 261
15.8.3 Motion Results 263
15.9 Clamps (Glue05) 263
15.10 Robots (Glue06) 265
15.11 Cure/Unload (Glue07) 266
15.12 Making Widgets (Glue08) 271

16 THE GLUING CELL EXERCISE IN TRANRUN4 273
16.1 The Gluing System 273
16.2 Simulation and Prototyping 274
16.3 The Project Components 274
16.4 Glue00: Conveyor Simulation 275
16.4.1 The Dynamic Model 275
16.4.2 Creating the Conveyor Task 277

16.4.3 The Data Logging Task 280
16.4.4 Data Communication Between Tasks 284
16.4.5 The Main File 286
16.4.6 Glue00 Results 288
16.5 Glue01: An Oven Simulation 288
16.5.1 Configuration and Status Printouts 289
16.6 Glue02: PID Control 291
16.7 Glue03: The Operator Interface 292
16.7.1 Results 297
16.8 Glue04: Motion Profiling 299
16.9 Glue05: Belt Sequencing 306
16.10 Glue06: The Glue Application Machine 307
16.11 Glue07: Transport Task Supervision 309
16.12 Glue08: The Completed Assembly System 311

17 THE GLUING CELL EXERCISE IN TRANRUNJ 315
17.1 Getting Started 315
17.1.1 Program Entry Point 315
17.1.2 The userMain Method 316
17.2 Writing Custom Tasks and States 317
17.2.1 Creating a Task Class 317
17.2.2 Creating a State Class 319
17.3 Implementing State Transition Logic 320
17.4 Global Data and Intertask Messaging 321
17.4.1 Global Data Items 321
17.4.2 Task Messages 322
17.5 Continuous vs. Intermittent Tasks 323
17.6 Scheduler Internals 324
17.6.1 Operating System Processes vs. CProcess 324
17.6.2 Foreground vs. Background Execution Lists 325
17.6.3 Scheduling Modes 325
17.7 Execution Profiling 325
17.8 Intertask Messaging Across Different Processes 326
17.9 Tips And Tricks 328
17.9.1 Judicious Use of Execution-Time Profiling 328
17.9.2 Integer Labels for Global Data and Task Message Inboxes 328
17.9.3 The TaskMessageListener Interface 328
17.9.4 Scheduler Sleeping 329
17.9.5 Anonymous State Classes 329
17.10 Additional Information 330

BIBLIOGRAPHY 331

INDEX 333