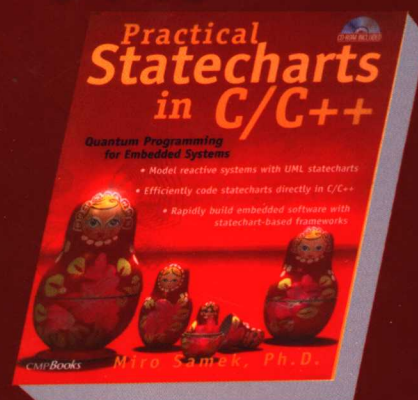


嵌入式系统译丛

[美] Miro Samek Ph.D. 著
敬万钧 陈丽蓉 译

嵌入式系统的 微模块化程序设计 ——实用状态图C/C++实现

Practical Statecharts in C/C++
Quantum Programming
for Embedded Systems



CMPBooks



北京航空航天大学出版社
<http://www.buaapress.com.cn>

内 容 简 介

建模反应式系统而不使用重量级的工具,作者的量子编程(QP)是一种新的范型。它把状态图作为一种设计方法,而不是作为特殊工具来使用。在本书的第1部分,给出相关概念清晰而明确的叙述,包括传统的有限状态机和状态图,以及基于状态图的设计模式;给出了可运行代码,使读者能通过实际操作来学习量子编程,以及学到状态嵌套如何导致行为继承和如何通过按差异编程而实现重用。第2部分完整地叙述了量子框架的实现,以及说明如何在应用中使用它,并移植到所选用的RTOS。本书适于嵌入式系统、实时系统及UML状态图的相关工程设计人员使用,并可作计算机科学和电气工程高年级学生的教学用书。所附光盘包含了作者的量子框架的全部源代码、散见于全书的所有练习的答案以及一个RTOS-32的评估板——X86处理器的32位实时操作系统。

图书在版编目(CIP)数据

嵌入式系统的微模块化程序设计:实用状态图C/C++
实现/(美)萨梅克(Samek, M.)著;敬万钧等译.
北京:北京航空航天大学出版社,2004.8
ISBN 7-81077-415-8

I. 嵌… II. ①萨…②敬… III. 模块化程序设计
IV. TP311.11

中国版本图书馆CIP数据核字(2004)第058172号

本书英文版原名: Practical Statecharts in C/C++ Quantum Programming for Embedded Systems
Copyright © 2002 by CMP Books, except where noted otherwise.
Published by CMP Books, CMP Media LLC.
4601 West 6th St, Suite B
Lawrence, KS 66049, USA
All rights reserved.

本书中文简体字版由美国CMP Books公司授权北京航空航天大学出版社在中华人民共和国境内(不包括香港特别行政区)独家出版发行。版权所有。

北京市版权局著作权登记号: 图字: 01-2002-4205

嵌入式系统的微模块化程序设计 ——实用状态图C/C++实现 Practical Statecharts in C/C++ Quantum Programming for Embedded Systems

[美] Miro Samek Ph. D. 著 敬万钧 陈丽蓉译
责任编辑 王 瑛
责任校对 戚 爽

*

北京航空航天大学出版社出版发行

北京市海淀区学院路37号(100083) 发行部电话:010-82317024 传真:010-82328026

http://www.buaapress.com.cn E-mail: bhpress@263.net

涿州市新华印刷有限公司印装 各地书店经销

*

开本: 787×960 1/16 印张: 21.5 字数: 482千字
2004年8月第1版 2004年8月第1次印刷 印数: 6 000册
ISBN 7-81077-415-8 定价: 48.00元(含光盘1张)

译者序

Miro Samek 博士是美国 Integri Nautics 公司的首席软件结构师。他在该公司以及此前在 GE 医疗系统都是从事实时软件开发,长期的工作使他在嵌入式实时软件开发方面积累了丰富的实践经验。他将其所积累的嵌入式实时软件开发的经验写成了这本理论和实践并重,而实践性更强的著作。

本书中除概念的叙述之外,Samek 博士还提供了完整的源代码,包括完整的软件基础结构和用以执行裁剪自多线程嵌入式应用的状态图。灵活的、有效的、可移植的、可伸缩的以及可维护的轻量级量子框架允许快速、直接地从 UML 状态图以 C 或 C++ 着手实时系统的工作。

量子框架是完整的实时嵌入式实现环境——一个设计用以与任何 RTOS 一起工作的活动对象框架。它支持快速原型构造,可在开发的任何阶段容易地进行状态的转换,以及可选择实现语言(C 或 C++),以与目标系统的资源约束相匹配;它最大的好处是非常紧凑,整个 QP 代码典型地要求少于 5 KB 的代码和数据。这些都降低了实现抽象概念的困难程度,对希望了解层次式状态机能力的嵌入式软件工程师是非常必要的。

关于本书的译名,为适合我国当前的具体情况,将其定为“嵌入式系统的微模块化程序设计”,而原作名应直译为“嵌入式系统



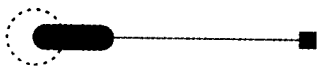
的量子程序设计”。但量子程序设计并不为我国读者所熟悉,经与有关专家商议,确定使用现译名。而在正文中,由于作者多处将 QP 与量子力学概念进行比拟,为维持叙述的一致性,故在正文中仍使用“量子程序设计”一词,请读者加以注意。

在本书的译出中,得到了电子科技大学“嵌入式实时系统研究中心”的大力支持,以及中心主任熊光棣教授的关心和帮助;同时在北京航空航天大学出版社马广云老师的鼎力帮助下,本书才得以及时面世。在此均表示由衷的感谢。

本书第 1~6 章由敬万钧翻译,第 7~11 章由陈丽蓉翻译。译者虽然尽心尽力,多次修改译稿,但疏忽欠妥之处定有,恳请业内同仁予以指正。

译 者

2003.8 于成都



第 1 部分 状态图

第 1 章 量子编程的快速浏览

1.1 终结钩子——GUI 应用程序的剖析	3
1.2 程序设计的更好办法——一个计算器的工作	5
1.2.1 传统的事件动作范型的缺点	5
1.2.2 计算器状态图	7
1.2.3 同 Windows 集成	11
1.2.4 状态处理器方法	13
1.3 面向对象相似性	15
1.3.1 状态层次和类分类学	15
1.3.2 进入/退出状态和初始化/结束类	16
1.3.3 按差异编程	16
1.3.4 行为继承作为基本的元模式	16
1.3.5 状态模式	17
1.3.6 重构状态模型	17
1.3.7 超越面向对象编程	18
1.4 量子类比	18
1.5 小 结	18

第 2 章 状态图速成

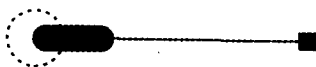
2.1 有限状态机的本质	21
--------------------	----



2.1.1	状 态	21
2.1.2	扩展状态	21
2.1.3	监测器	22
2.1.4	事 件	23
2.1.5	动作和转换	23
2.1.6	Mealy 和 Moore 自动机	24
2.1.7	执行模型——RTC 步	24
2.1.8	状态转换图	25
2.2	UML 状态图的本质	26
2.2.1	层次式状态	26
2.2.2	行为继承	28
2.2.3	正交区域	29
2.2.4	进入和退出动作	30
2.2.5	转换执行序列	32
2.2.6	内部转换	33
2.2.7	伪状态	33
2.2.8	细化事件处理	34
2.2.9	语义与表示法	34
2.2.10	状态图和流程图	35
2.2.11	状态图与自动代码综合	36
2.3	状态模型的例子	37
2.3.1	量子计算器	37
2.3.2	氢原子	41
2.4	小 结	44

第 3 章 标准状态机实现

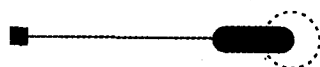
3.1	状态机接口	46
3.2	嵌套的 switch 语句	48
3.3	状态表	51
3.4	状态设计模式	56
3.5	优化的 FSM 实现	60
3.6	状态机和 C++ 异常处理	64
3.7	成员函数指针的作用	65
3.8	实现监测器、接合和选择点	66



3.9 实现进入和退出动作·····	67
3.10 处理状态层次·····	68
3.11 小 结·····	69

第4章 实现行为继承

4.1 结 构·····	71
4.1.1 事 件·····	74
4.1.2 状 态·····	76
4.1.3 进入/退出动作和初始转换·····	77
4.1.4 状态转换·····	79
4.1.5 top 状态和初始伪状态·····	80
4.2 一个注释性例子·····	81
4.2.1 枚举信号和子类构造 QHsm·····	82
4.2.2 定义状态处理器方法·····	83
4.2.3 初始化和分发事件·····	86
4.2.4 测试运行·····	87
4.3 试探和约定·····	89
4.3.1 构建状态机代码·····	89
4.3.2 选择正确的信号粒度·····	91
4.3.3 UML 一致的 HSM·····	91
4.4 事件处理器·····	93
4.4.1 初始化状态机: init() 方法·····	93
4.4.2 分发事件: dispatch() 方法·····	95
4.4.3 静态和动态状态转换: 宏 Q_TRAN 和 Q_TRAN_DYN()·····	96
4.4.4 动态状态转换: tran() 方法·····	97
4.4.5 静态状态转换: tranStat() 方法和 Tran 类·····	102
4.4.6 初始化 QTran 对象: tranSetup() 方法·····	104
4.5 C 实现·····	107
4.5.1 用“C+”的 QHsm 类·····	107
4.5.2 QHsm 构造函数与析构函数·····	109
4.5.3 状态处理器方法和成员函数指针·····	109
4.5.4 QHsm 方法·····	110
4.5.5 用 C 的状态图例子·····	110
4.6 防止误解的说明·····	113

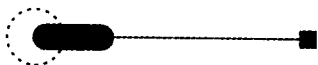




4.7 小 结	115
---------------	-----

第 5 章 状态模式

5.1 终结钩子	118
5.1.1 目 的	118
5.1.2 问 题	118
5.1.3 解决办法	118
5.1.4 样例代码	119
5.1.5 结 论	121
5.2 提示法	122
5.2.1 目 的	122
5.2.2 问 题	122
5.2.3 解决办法	122
5.2.4 样例代码	123
5.2.5 结 论	127
5.3 延迟事件	128
5.3.1 目 的	128
5.3.2 问 题	128
5.3.3 解决办法	128
5.3.4 样例代码	130
5.3.5 结 论	133
5.4 正交构件	133
5.4.1 目 的	133
5.4.2 问 题	133
5.4.3 解决办法	134
5.4.4 样例代码	135
5.4.5 结 论	141
5.5 转换到历史	143
5.5.1 目 的	143
5.5.2 问 题	143
5.5.3 解决办法	143
5.5.4 样例代码	143
5.5.5 结 论	147
5.6 小 结	147



第 6 章 继承状态模型

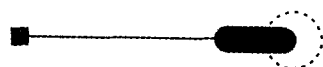
6.1 用 C++ 的状态图细化举例	149
6.2 用 C 的状态图细化举例	157
6.2.1 为多态性准备 C+ 状态机	157
6.2.2 继承和细化 C+ 状态机	159
6.3 防止误解的说明	160
6.3.1 静态与动态状态转换的比较	161
6.3.2 多重继承	161
6.3.3 继承和细化状态图的探索	163
6.4 小 结	165

第 2 部分 量子框架**第 7 章 量子框架介绍**

7.1 传统多线程方法	170
7.1.1 就餐哲学家——传统方法	171
7.1.2 Therac-25 的故事	174
7.2 QF 的计算模型	175
7.2.1 基于活动对象的多线程	176
7.2.2 量子类比	177
7.2.3 出版-订阅事件交付	179
7.2.4 通用结构	180
7.2.5 重访就餐哲学家问题	180
7.3 QF 的角色	183
7.3.1 概念完整的源泉	184
7.3.2 RTOS 抽象层	185
7.3.3 软件总线	186
7.3.4 高度并行计算平台	187
7.3.5 自动代码生成的基础	188
7.4 小 结	189

第 8 章 量子框架的设计

8.1 嵌入式实时系统	191
-------------------	-----





8.2 处理错误和异常条件	192
8.2.1 按契约设计	194
8.2.2 基于状态的异常条件处理	197
8.3 内存管理	198
8.3.1 堆问题	199
8.3.2 QF 中的内存管理	201
8.4 互斥和阻塞	203
8.4.1 互斥的危险	203
8.4.2 基于活动对象的系统中的阻塞	206
8.5 传送事件	207
8.5.1 动态事件分配	209
8.5.2 出版-订阅模型	212
8.5.3 组播事件	217
8.5.4 自动事件回收	219
8.6 活动对象	220
8.6.1 内部状态机	220
8.6.2 事件队列	221
8.6.3 执行线程	222
8.7 初始化和清除	224
8.7.1 初始化框架	224
8.7.2 启动 QF 应用	225
8.7.3 优美地终结 QF 应用	226
8.8 时间管理	226
8.8.1 QTimer 类	227
8.8.2 时钟节拍, QF::tick() 方法	228
8.9 QF API 快速参考	229
8.9.1 QF 接口	229
8.9.2 QActive 接口	231
8.9.3 QTimer 接口	232
8.10 小 结	233

第 9 章 量子框架的实现

9.1 作为一个 Parnas 家族的 QF	236
9.2 代码组织	236



9.2.1 目录结构	238
9.2.2 公共范围头文件	240
9.2.3 包范围头文件	241
9.3 公共元素	242
9.3.1 临界区	242
9.3.2 事件池	243
9.3.3 事件队列	247
9.4 DOS:不具备多任务内核的 QF	253
9.4.1 前台/后台处理	253
9.4.2 使用 QF 的前台/后台	254
9.4.3 DOS 特定的代码	258
9.4.4 QF 给前台/后台系统带来的益处	260
9.5 Win32:桌面上的 QF	261
9.5.1 临界区、事件队列和执行线程	261
9.5.2 时钟节拍	264
9.6 RTKernel-32:带有一个基于优先级的可抢占内核的 QF	266
9.6.1 临界区和事件池	266
9.6.2 事件队列和执行线程	269
9.6.3 RTKernel-32 的初始化和时钟节拍	271
9.6.4 使用 RTKernel-32 进行交叉开发	272
9.7 小 结	273

第 10 章 量子框架应用实例

10.1 产生一个 QF 应用	275
10.1.1 信号和事件	275
10.1.2 Table 活动对象	277
10.1.3 Philosopher 活动对象	280
10.1.4 配置 DPP	282
10.1.5 注 解	283
10.2 开发 QF 应用的规则	284
10.3 开发 QF 应用的启发	286
10.4 划分事件队列和事件池的大小	287
10.4.1 事件队列	287
10.4.2 事件池	290



10.5 系统集成.....	290
10.6 小 结.....	291

第 11 章 结束语

11.1 QP 的关键元素	292
11.1.1 是一个设计类型,而不是工具	293
11.1.2 是一个建模助手.....	293
11.1.3 是一个学习上的助手.....	294
11.1.4 是一个有用的隐喻.....	295
11.2 QP 的建议	296
11.2.1 量子编程语言.....	296
11.2.2 未来的 RTOS	297
11.2.3 硬件/软件协同设计	297
11.3 一个邀请.....	298

附录 A C+——用 C 作面向对象编程

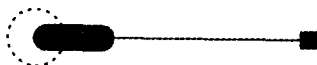
A.1 抽 象	301
A.2 继 承	303
A.3 多态性	305
A.4 费用和额外开销	312
A.5 小 结	313

附录 B 表示法指南

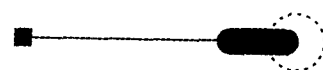
B.1 类 图	315
B.2 状态图	317
B.3 顺序图	318
B.4 时序图	318

附录 C CD-ROM

C.1 源代码结构	321
C.2 安 装	321
C.2.1 源代码	321
C.2.2 OnTime RTOS-32 评估套件	321
C.2.3 Adobe Acrobat Reader	322



C.3 练习的答案	322
C.4 资 源	323
参考文献.....	324



第 1 部分 状态图

必须对输入事件作出反映的系统就是事件驱动系统。这是当前一种流行的系统,而为了定义和实现事件驱动系统,状态机是一种很好的形式方法,UML 状态图代表了状态机理论和表示法的当前状况。

第 1 部分介绍状态图的概念,叙述直接用 C 和 C++ 编码状态图的具体技术,并且提供主要的基于状态图设计模式的一个小目录。读者将会了解到状态图是一种强有力的设计方法,可以使用它而不必借助于复杂的代码综合工具来进行设计。

第1章

量子编程的快速浏览

我发现,为了找到你喜欢或讨厌的人,除了同他一起旅行外,没有更可行的办法。

——Mark Twain

在过去的 30 年间的软件开发中,图形用户界面的成功是最为引人注目的^①。今天,这个概念已为大家所熟悉而不需要再加以赘述。虽然从一开始,窗口、图标、菜单以及指示器就已是直观的,并易于为用户接受的,但是对程序设计者来说,它们仍然是一种挑战。这是因为 GUI 内部结构对很多新手是困难的。他们经常发现它是陌生的、不合潮流的、思路不清或古怪的。GUI 编程之所以有这些不同之处,是因为它不像传统的数据处理。它是完全事件驱动的,事件能以任何顺序、在任何时刻出现,应用程序必须随时准备好去处理它们。GUI 是复杂反应式系统的一个例子。

没必要去寻找其他的反应式系统的例子,事实上,所有 PC,MAC 以及其他通用计算机的 CPU 仅仅消耗世界范围微处理器产品的 1% 左右,而其他 99% 的微处理器芯片都为各种嵌入式系统所用,嵌入式自然是最主要的反应式系统。尽管具有如此的普遍性,但绝大多数此类系统的编码都是非常难以理解、调整和维护的。建造此类软件的理论基础已经出现了 10 多年,但是,这些想法却并未成为主流。量子编程(QP)就是要使这些现代方法为程序员们所接受。QP 是直观设计模式、习惯方式、具体实现以及易于明白技术的集合,这使读者可立即着手使用而无需去研究各种复杂的工具。

1.1 终结钩子——GUI 应用程序的剖析

早期的 GUI 设计师面临着困难的任务:一方面,GUI 应用程序必须使得从开矩形窗口到划分屏幕,以及令人眼花缭乱的屏幕帮助等事情成为非常惯用的动作;另一方面,系统又必须保证观感一致性,而且具有这些标准性能的应用程序还必须简单。对这种矛盾

^① 窗口、图标、菜单和指示器(WIMP)界面的概念是由斯坦福研究所的 Douy Englebart 和他的队伍在 1968 年的西部联合计算机会议上首次公布的。



的要求如何来进行折衷呢？

现在这个问题似乎简单了——使用终结钩子的办法[Petzold, 1996]。其想法是非常简单的, GUI 系统(如 Windows)首先分发所有的事件到应用程序(Windows 调用在应用程序中的一个特殊函数)。如果应用程序没有处理,则各事件返回系统,这就建立了一个事件处理的层次顺序。应用程序在概念上处于层次的最低级,有机会对每一事件作出反应;因此,应用程序能够定制它的事件行为的各个方面,同时,所有未被处理的事件返回较高级层次(即回到系统),系统对它们按照标准的观感处理。这是一个“按差异编程”的例子,因为应用程序人员仅仅对不同于标准的系统行为进行编码。

David Harel 独立地将同样的想法应用到有限状态机形式方法[Herel, 1998]。大约在 1983 年,他发明了状态图。作为规定复杂反应式系统的一种强有力的方法,状态图较之传统有限状态机的主要创新之处在于层次式状态的引入。为了了解它的意思,将被嵌套状态(子状态)和包围状态(超状态)之间的关系示于图 1.1(a)。此状态图首先要处理子状态范围中的所有事件,如果子状态不能处理这些事件,状态图自动地送事件到下一个较高级状态(即超状态)。当然,状态的嵌套是可以进一步加深的,而处理事件的简单规则反复地被用于嵌套的任意级。

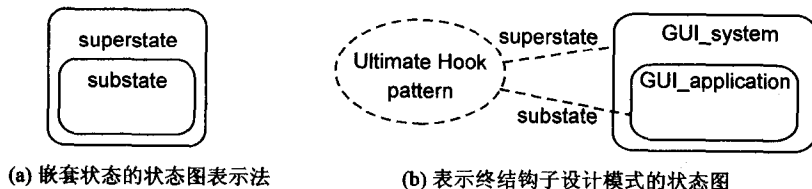


图 1.1 状态层次图

Harel 的状态层次语义学是构成 GUI 系统的终结钩子设计的核心。换句话说,状态图直接支持终结钩子模式。当其把超状态改名为 GUI_system,子状态改名为 GUI_application 时,这就变得非常明白了,如图 1.1(b)所示。现在有了一个非常有趣的结果:基本的终结钩子设计模式转化成了非常简单的状态图! 这个强有力的状态图是非常简单的,因为它在这个抽象级没有包含任何状态转换。

通常,在状态图中引入的状态层次已经证实下述是正确的:

结果,高度复杂的行为已不能用简单的、“平面的”状态转换图来方便地描述。其根源在于不可管理的大量的状态,它们可能导致无结构的和混乱的状态转换图。

——[Herel+, 1998]

当然,层次式图形经常比传统的平面图有更为简单和更好的结构,但是,这并不是状态层次重要性的主要理由,而只是副作用之一。状态层次即使在没有大量状态和转换的