计算机科学2001Vol.28№.12

实时系统设计方法

The Design Approach for Real-Time System

王 莲[1] 张云勇[2]

(科技部西南信息中心 重庆400013)[1] (电子科技大学计算机学院 成都610054)[2]

Abstract The real-time system is used widely since the 1990's. In the paper some typical design approaches are introduced. Then the object-oriented design approach for real-time system and the use of UML is analyzed.

Keywords Real-time system, Design approach, Time Petri Net, Object-oriented, UML, UML-RT

1. 引言

实时系统在工业、商业和军事等领域都有非常广泛的用途，并且已经有很多实际的应用。实时系统可以分为"硬实时系统"和"软实时系统"。二者的区别在于：前者如果在不满足响应时限、响应不及时或反应过早的情况下都会导致灾难性的后果；而后者则在不满足响应时限时，系统性能退化，但并不会导致灾难性的后果。实时系统一般应具有以下的特点：

实时性：软实时/硬实时，快速启动，随时就绪；

异步事件的并发处理：实时多任务机制；

应用/OS 支持一体化；

可固化：飞盘/DOC/DOM，代码精简；

鲁棒性：可靠/出错处理/自动恢复；

灵活性：可剪裁/可配置/可移植/可重用；

安全性：能应用在关键场合。

一般来说，实时系统是比较复杂的，因为它必须处理很多并发事件的输入数据流，这些事件的到来次序和几率通常是不可预测的，而且还要求系统必须在事先设定好的时限内做出相应的响应，所以实时系统的设计方法和普通系统有一定的差别。本文首先阐述实时系统中一些典型的设计方法，然后剖析实时系统面向对象的设计方法及 UML 的利用，并对 UML 的扩展进行了简单的介绍。

2. 实时系统中常见的设计方法

普通的系统设计方法主要有语言描述、数学分析、流程图、数据流图、结构图、伪代码、有限状态机、Petri 网络和面向对象的设计方法等。但是很多设计方法由于其自身的缺陷，都不适合于实时系统。如语言描述不够明确，数学分析难于应用，流程图没有并发处理，不能表现瞬时状态，结构图没有条件转移，没有并发处理，伪代码则容易出错。下面我们选择几种典型的、可以在实时系统中利用的设计方法加以阐述。

2.1 有限状态机(FSM)

有限状态机是实时系统设计中的一种数学模型，是一种重要的、易于建立的、应用比较广泛的、以描述控制特性为主的建模方法。它可以应用于从系统分析到设计的所有阶段。有限状态机的组成如下：(1)一个有限的状态集合 Q；(2)一个有限的输入集合 I；(3)一个变迁函数 $\delta: Q \times I \to Q$。其中变迁函数也是一个状态函数，在某一状态下，给定输入后，FSM 转入该函数产生的新状态。δ 的定义域内的某些数值可以是未定义的。

有限状态机通常用图的方式来表示，其节点代表状态。若在输入 i 下状态由 q1转变为状态 q2，则有一条标有输入 i 的弧线从状态 q1指向 q2。此时，其变迁函数 $\delta(q1,i)=q2$。图1示意了一个简单的有限状态机。

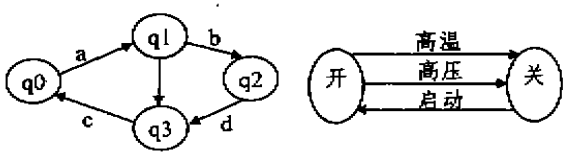

图1 简单有限状态机

图1中的左图表明该 FSM 有 q0、q1、q2、q3四个状态，输入集中有 a、b、c 三个元素；右图中各个状态之间的转换关系则更清晰。有限状态机非常适合于描述这样的系统：系统含有有限个状态，不同事件的发生可以用不同状态之间的转换来模拟。

一般来说，在进行有限状态机的设计时，其设计步骤为：

第一步：理解问题。将状态机的行为规范用语言文字描述出来，这一步必须注意的一点就是在讲解状态

机的描述时不能产生模棱两可的二义性，用词一定要准确，而且能够被一般人所理解；

第二步：FSM 的抽象描述。一旦理解清楚问题后，就要以某种形式对该问题进行抽象，这种形式应该容易为程序所操作，管理，以方便状态机的实现。表示形式有许多种可以选择，比如状态图，算法状态机，以及用伪代码语言进行描述等，但是通常人们选用的是状态图；

第三步：状态最小化。在第二步中，经常会生成很多的状态，该状态机上的某些路径实际上可以不要，因为有些其他功能等价的路径已经覆盖了它们的输入输出行为，所以我们要在一定理论基础下进行状态的最小化。但是，对于某些比较简单的状态机设计，这一步骤不是必需的；

第四步：状态机的实现，这一步应该完成的任务就是将已经得出的最简状态图在计算机中实现，包括状态的分配，状态的变换，输入的定义，输出及处理函数的实现，相对来说，这一步是比较烦琐的一步，在具体这一步工作时，其实现方式也有好几种。

总的来说，有限状态机的优点在于简单易用，状态间的关系能够直观看到。但应用在实时系统中时，其最大的缺点是：任何时刻系统只能有一个状态，无法表示并发性，不能描述异步并发的系统。另外，在系统部件较多时，状态数随之增加，导致复杂性显著增长。为了消除这些缺点，一些新的方法应运而生。下面介绍的 Petri 网就是其中一种，与 FSM 相比，它具有更强的建模能力，并能描述系统的并发、异步、同步等特性。

2.2 时间 Petri 网

Petri 网是一种使用图形方式对系统进行需求规格说明的技术，用来定义多进程、多任务系统的数学模型，易于描述系统的并发、竞争、同步等特征，并可用于设计、评价和改进系统。如今，Petri 网已经大量应用于各种系统的模型化。与 FSM 相比，Petri 网不仅能描述同步模型，更适合于相互独立、协同操作的处理系统。

Petri 网的组成成分包括：(1)一个有限的库所(place)集合 P，表示系统的状态；(2)一个有限的变迁(transition)集合 T，表示系统中的事件；(3)一个有限的连接库所到变迁或者反向的有向箭头的集合，又分输入 I 和输出 O；(4)标记 μ 是 Petri 网位置中一组令牌(tokens)的分配。图2为一 Petri 网示例，图中的库所用圆圈表示，变迁由矩形表示。Petri 网有如下特性：

(1)状态　通过标记 Petri 网的库所来给出其状态，标记 Petri 网的库所在图形中表现为对库所插入数目不同的令牌。

(2)状态变化规则　一个变迁可以有多个输入或输出库所，如果一条有向箭头是从库所到变迁，则该库所是该变迁的一个输入库所，反之则为输出库所。如果一个变迁中每一个输入库所都至少有一个令牌，则称该变迁是一个使能变迁。

(3)点燃　一个使能变迁可以被点燃，即从该变迁的每一个输入库所中移走一个令牌，在该变迁的每一个输出库所中增加一个令牌。

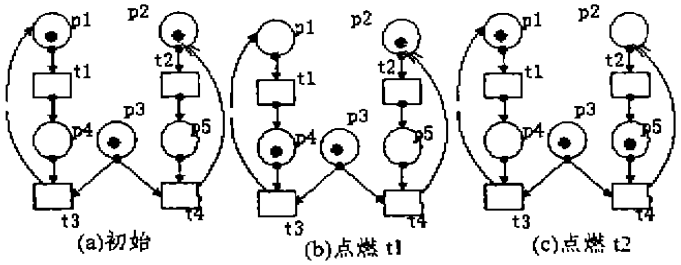

图2　Petri 网示例

图2中，库所中的实心代表令牌。由(a)中可以看出，变迁 t1和 t2都是使能的，(b)表示了点燃 t1的过程，(c)表示了点燃 t2的过程。从这个模型中可以看出，在给定初始令牌后，Petri 网的发展过程可能是不同的。在 Petri 网中，经常利用变迁来模拟一个事件，而点燃则用来表示事件的发生。这样，如果一个变迁所表示的事件的发生条件是满足的，那么这个变迁就是使能的，库所中的星号标记则说明某个条件是满足的。图2中的 Petri 网可分为两部分，一部分是由变迁 t1、t3组成，另一部分由 t2、t4组成，这相当于是两个独立的工作流，并且共享了资源 p3。开始时，双方可以互不干涉地、异步地进行，因为变迁 t1和 t2双方互不妨碍。点燃 t1后，t3处于使能状态，点燃 t2后，t4处于使能状态。在两个变迁都点燃后，两个工作流都有新的变迁处于使能状态，但是这两个变迁的点燃处于竞争状态，两个工作流中只能有一个获得 p3的资源从而得以继续变迁。如果资源满足的情况可以解决这种冲突，就可以设 p3的令牌有两个，两个工作流就互不干扰，可以并发执行。

应该说 Petri 网对分析设计实时系统起到了很大

的作用，但是为了进一步突出实时系统的实时特性，人们后来在传统 Petri 网的基础上引进了时间 Petri 网(Time Petri Net)的概念。时间 Petri 网的结构 Φ 由七部分组成，Φ(P，T，I，O，Min，Max，μ)，其中 Min 和 Max 分别是最小时间函数和最大时间函数，它们满足：

(1)Min：T→R，和 Max：T→R，R 为非负实数集；

(2)对所有 $t_i \in T$，$Min\ i \leqslant Max\ i$。

在时间 Petri 网中的点燃规则为：

(1)一个转移 t_i 在时刻 τ 是可点燃的当且仅当在时间区间[τ-Min(t_i)，τ]段内一直是使能的；

(2)可点燃的转移 t_i 在时间区间[τ-x，τ]段内是随时可点燃的，其中 x 满足 $Min(t_i) \leqslant x \leqslant Max(t_i)$；

(3)一个转移 t_i 在时刻 τ 是必须点燃的当且仅当在时间区间[τ-Max(t_i)，τ]段内一直是使能的。

时间 Petri 网通过转移的最大最小时间函数的约束，有效地控制令牌运动的时序性，这一特点使得时间 Petri 网在实时系统中的应用非常广泛。

当然 Petri 网也有缺点：Petri 网适合于多进程处理，对并行性有很好的表示，但应用于过小的系统时显得不必要。

2.3 DARTS(Design Approach for Real-Time Systems)

DARTS 是结构化分析/设计的扩展，它给出了划分任务的准则和任务间接口机制的定义。在需求分析的基础上，DARTS 设计方法的主要步骤如下：

2.3.1 数据流分析(Data Flow Analysis) 在系统需求分析的基础上，以数据流图作为分析工具，从系统的功能需求开始分析系统中的数据流，确定主要功能，扩展数据流图，并分解到足够的深度，识别出主要的子系统和每个子系统的主要成份。

每个数据流图都包含变换圈，表示系统完成的功能，箭头表示变换间的数据流动，数据存储区表示数据的存储场所。

数据字典定义了数据流和数据存储区所包含的数据项。

2.3.2 任务划分(Dicomposition into Tasks) 将一个软件系统分解成并行任务主要需考虑的是系统内功能的异步性：分析数据流图中的变换，确定哪些变换可以并行，而哪些变换在本质上是顺序的。通过这种方法，划分出任务：一个变换对应一个任务，或者一个任务包括几个变换。任务划分准则如下：

I/O 依赖性(Dependency on I/O)：如果变换依赖于 I/O，那么它运行的速度常常受限于与它互操作的 I/O 设备的速度。在这种情况下，变换应成为一个独立的任务。

功能的时间关键性(Time-critical functions)：具有时间关键性的功能需要以高优先级运行，因而，应成为一个独立的任务。计算需求(Computational requirements)：需要进行大量计算的功能(或功能集)可以作为较低优先级任务运行，消耗 CPU 的剩余时间。

功能内聚(functional cohesion)：完成的功能紧密相关的变换可以组成一个任务，因为这些功能间的数据通信较多，把它们作为一个个独立的任务会增加系统的开销；反之，把每个变换都作为同一任务中一个独立的模块，不仅保证了模块级的功能内聚，而且保证了任务级的功能内聚。

时间内聚(Temporal cohesion)：某些变换完成的功能是在同一时间执行的，这些变换可以组成一个任务，这样，每次任务接收到一个事件，它们都可以执行。

周期执行(Periodic execution)：一个需要周期执行的变换可以作为一个独立的任务，按一定的时间间隔被激活。

2.3.3 任务接口定义(Definition of Task Interfaces) 在 DARTS 中，定义两类任务接口模块来处理接口问题：任务间通信模块(TCM)和任务间互斥模块(TSM)。DARTS 支持两种 TCM：消息通信模块(MCM)和信息隐藏模块(IHM)。消息通信由一个称为消息通信模块的 TCM 进行处理，MCM 支持松耦合和紧耦合两类消息通信。而消息隐藏模块的 TCM 用来为共享资源(如查询数据、数据池等)定义数据存储区和对它进行访问的访问过程。

2.3.4 任务设计(Task Design) 画出每个任务的数据流图，然后使用结构化设计法，从数据流图导出任务的模块结构图，并且定义出各模块的接口。

最后，经过模块构筑，再经过任务和系统集成就实际实现了实时系统。

3. 实时系统面向对象的设计方法

随着面向对象编程(OOP)技术的日益普及，开发人员在分析和设计阶段也使用了面向对象的方法(OOA&OOD)以提高整体开发效率，而且由于实时系统开发的复杂性及对质量和可靠性的严格要求，往往需要面向对象方法和措施来保证软件的品质。面向对象的方法有很多种，这里所介绍的是利用统一建模语言(UML)的建模技术。

作为一种建模语言，UML 的定义包括 UML 语义和 UML 表示法两个部分。

UML 语义：描述基于 UML 的精确元模型定义。元模型为 UML 的所有元素在语法和语义上提供了简单、一致、通用的定义性说明，使开发者能在语义上取得一致，消除了因人而异的最佳表达方法所造成的影

响，此外 UML 还支持对元模型的扩展定义。

UML 表示法：定义 UML 符号的表示法，为开发者或开发工具使用这些图形符号和文本语法为系统建模提供了标准。这些图形符号和文字所表达的是应用级的模型，在语义上它是 UML 元模型的实例。

标准建模语言 UML 的重要内容可以由下列五类图（共9种图形）来定义：

·第一类是用例图（Use case diagram），从用户角度描述系统功能，并指出各功能的操作者。

·第二类是静态图（Static diagram），包括类图（Class diagram），描述系统中的主要类。

·第三类是行为图（Behavior diagram），即状态转移图（State Transition diagram）描述类的对象所有可能的状态以及事件发生时状态的转移条件。

·第四类是交互图（Interactive diagram），描述对象间的交互关系。其中顺序图（Sequence diagram）显示对象之间的动态合作关系，它强调对象之间消息发送的顺序，同时显示对象之间的交互；合作图（Collaboration diagram）描述对象间的协作关系，合作图跟顺序图相似，显示对象间的动态合作关系。除显示信息交换外，合作图还显示对象以及它们之间的关系。如果强调时间和顺序，则使用顺序图；如果强调上下级关系，则选择合作图。

·第五类是实现图（Implementation diagram）。其中部件图（Component diagram）描述代码部件的物理结构及各部件之间的依赖关系。一个部件可能是一个资源代码部件、一个二进制部件或一个可执行部件。它包含逻辑类或实现类的有关信息。部件图有助于分析和理解部件之间的相互影响程度。配置图（Deployment diagram）定义系统中软硬件的物理体系结构。它可以显示实际的计算机和设备（用节点表示）以及它们之间的连接关系，也可显示连接的类型及部件之间的依赖性。在节点内部，放置可执行部件和对象以显示节点跟可执行软件单元的对应关系。

从应用的角度看，采用面向对象技术设计系统时，首先是描述需求；其次根据需求建立系统的静态模型，以构造系统的结构；第三步是描述系统的行为。其中在第一步与第二步中所建立的模型都是静态的，包括用例图、类图、组件图和配置图等图形，是标准建模语言 UML 的静态建模机制。其中第三步中所建立的模型或者可以执行，或者表示执行时的时序状态或交互关系，它包括状态转移图、顺序图和合作图等三个图形，是标准建模语言 UML 的动态建模机制。因此，标准建模语言 UML 的主要内容也可以归纳为静态建模机制和动态建模机制两大类。

UML 的这些特性比较适合软实时系统的设计，在硬实时系统中，有时显得不是非常有效，为此以 Object Time 公司为首的一些厂家提出了 UML 的实时扩展 UML-RT，引进了 Connector、Ports、Capsule、SubCapsule 等元素，可以简单地认为 UML-RT 是 UML 的子集和 ROOM（Realtime Object Oriented Modeling）的组合。UML-RT 一经提出，得到众多厂商的响应，并开发出了相应的产品。如 Rational 公司的 Rational Rose realtime、ObjectTime 公司的 ObjectTime 以及 I-logic 公司的 Rhapsody 等，它们都能较好地体现软件工程的生命周期，能自动生成代码，大大提高了效率。

总结 本文对应用于实时系统的、常用的多种分析和设计方法进行了阐述，应该明确的一点是，没有一种分析或设计方法是可以应用于任何场合的，事实上，很多情况都是几种方法的混合使用。

二十一世纪是后 PC 设备时代，即非 PC 信息设备大显神通的时代，实时嵌入式系统正是非 PC 设备的主体。互联网技术在世界范围的扩展和通信事业的高速发展，已为开发实时嵌入式产品造就了广大市场。"工欲善其事，必先利其器"，只有采用了行之有效的设计方法才能方便而快捷地开发出实时系统，从而快速地占领市场。

参考文献

1 电子科技大学实时软件技术研究组. 实时系统及其软件技术，1999年5月

2 CESD. CRTOS x86/PM 嵌入式实时操作系统原理与应用设计. 电子科技大学计算机学院，1998年

3 袁崇义. Petri 网络原理. 电子工业出版社，1998年

4 何国伟，王纬. 软件可靠性. 国防工业出版社，1998. 257～267

5 张云勇. 面向 Agent 的软件工程：[电子科技大学计算机学院博士生专题研究报告]. 2000. 8

6 Boggs W，Boggs M. UML with Rational Rose. 电子工业出版社，2000. 3

7 Lyons A. UML for Real-Time Overview. April 1998 Available at: http://www.rational.com/products/whitepapers/100463.jsp