返回总目录

目　　录

第 8 章　高级动态建模：实时系统

用 UML 建模的大部分系统是实时系统。关于什么是实时系统，没有一个准确的定义，但是在实时系统中，时间是最重要的因素。具体来说，实时系统具有以下一些特点：系统必须在指定时间内处理外部事件，进程并发执行，以及需要对系统的性能进行优化。基于这些特点，可以将大多数系统看作是实时系统，因而需要在模型中描述时间限制。

图 8-1 显示了一个实时系统。

- 时间线是最重要的：系统必须在指定时间内(“响应时间”)完成指定功能。
- 事件驱动：系统必须连续响应外部环境产生的事件，这些事件驱动系统的执行。
- 软件的不同部分以线程的形式并发执行：并发线程可以在一个真正的并发系统中运行，在这样的系统中，不同的线程可以在不同的处理器上执行；并发线程也可以在一个模拟的并发环境中运行，在这样的环境中，只有一个处理器执行所有线程，但是并发操作系统可以合理调度这些线程共享同一个处理器。并发可以有效地利用硬件资源，在大量外部事件以异步形式出现的系统的建模时非常有用。
- 对非功能性的需求有较高的要求，如可靠性、容错、性能。
- 非确定性：因为并发的复杂性，要证明一个系统在所有条件下正确地工作是不可能的。

实时系统建模需要描述下列情况：时间需求，异步事件处理，通信，并发和同步。因为实时系统通常是分布式的，所以模型常常还要描述系统的分布。与实时系统最相关的图是动态模型图，如序列图和协作图。但是，有一点要说明的是，任何一个实时系统也会有静态结构，这些结构需要用类图和分布图来描述。

一个实时系统通常与其运行的物理环境紧密结合在一起，因为实时系统通常通过传感器和控制设备来接收事件和采集信息。因而，实时系统经常同一些特殊的硬件联系在一起，不得不处理低级中断和硬件接口。与特殊软、硬件紧耦合在一起的系统称为嵌入式系统。在汽车，家用电器，制造机械等系统中常带有嵌入式系统。嵌入式实时系统通常需要使用一个小的需要很少内存的、容易集成进系统的小实时操作系统。即使在最坏情况下(所有事件一起发生)，嵌入式系统也必须能够尽可能快地处理与控制硬件有关的事件。

通常将实时系统分为两类：硬实时系统和软实时系统。在硬实时系统中，出现迟缓的或不正确的响应是不可接受的，即将导致系统的终止。如，飞机控制系统，救生系统，火车自动控制系统等都是硬实时系统。而在软实时系统中，偶尔可以接收一个迟缓的或不正确的响应，如，在一个数字电话系统中，有时连接一个呼叫可能需要较长的时间，或连接可能会失败，对这两种情况，没有一种认为是严重的或危险的错误。

因而，设计实时系统的要求比较高，特别是在设计硬实时系统时。系统必须具有容错能力，即系统必须能够处理软件和硬件中的意外错误，不管在何种情况下，系统必须是可操作的。为了即使在最坏情况下，系统也能够处理事件，必须对系统性能进行优化，使得在必须的时候系统有足够的资源来处理发生的事件。

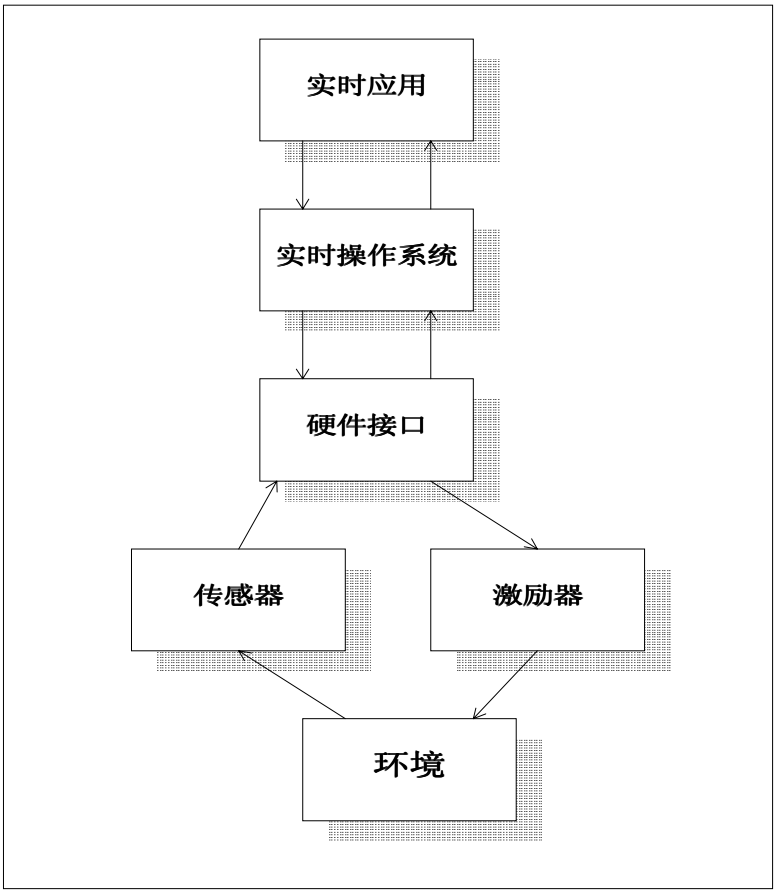

图 8-1　一个实时系统的例子

8.1　面向对象和实时系统

在进行面向对象的建模时，常问的一个问题是：面向对象是否真正能够模型化一个实时系统。因为实时系统对性能的高要求以及一些人对于面向对象的一些错误理解，即认为对象仅仅是存储在关系数据库中的数据而已，所以很多人对这一问题给出了否定意见。很明显，因为实时系统的复杂性，以及要求设计的模型准确和详细，因而比描述一般的系统要难得多。但是，只要模型语言支持传统的面向对象概念和实时概念，就可以将面向对象和实时性有机地结合起来。

另外，还有在实现时碰到的问题，如底层的环境必须完全支持实时结构(进程或线程，通信和同步机制)。有时还会碰到需要对模型实现进行优化的问题。比如，当模型中有太多的抽象层，导致层间对象间通信代价太大时。在这种情况下，实现就不能完全按照理想的模型去做。如果非要按照模型去做的话，则会得到一个非常复杂和很难维护的系统。

前面已提到过，面向对象的实时系统需要支持实时结构的操作系统。操作系统可以是一个标准的操作系统，如 Windows, OS/2，或 UNIX，也可以是一个专用于嵌入式系统的

实时操作系统。实时服务的质量(性能，可靠性，易于编程)起决于使用的操作系统，而不是起决于面向对象的语言或方法的特点。面向对象被用来建立系统模型，将建立的模型映射到操作系统提供的服务上。很自然地，面向对象的设计的质量对最终的系统的质量有很重要的影响。

当为有并发执行的系统建模时引入的一个基本概念是活动类。活动类的活动对象(active object)能够执行它自己的控制线程，因而，可以不需发送一条消息就可以开始执行一个动作。在一个有多个活动对象的系统中，多条控制线程将并发执行。这将引入新的问题，如活动对象间的通信，同享资源时的同步问题等。与活动类相对的是静态类。静态类就是我们通常意义上的类。一个静态类的对象只有在收到一个消息时(即，其操作被调用时)才执行，当操作返回时，控制返回给它的调用者。实时系统是活动对象和静态对象的混合体。

可以通过显式的或隐式的并发模型(在 M.Awad et al., 1996 中有详细描述)来描述面向对象系统中的并发性。在显式的并发模型中，将对象和并发性分开来描述，方法是在分析的早期阶段定义进程，然后将进程和对象当作不同的建模实体。系统被分解成很多进程，每一进程通过一个面向对象的系统来描述其内部结构(当然，类可以在几个进程中重用)。

隐式的并发模型则推迟并发性的设计。在分析阶段，将系统抽象成一组对象，所有对象都有自己的执行线程，也就是说，对象是活动对象(或，只有那些被标识为活动的对象才有自己的执行线程)。渐渐地，通过构架和详细设计，理想的模型被映射到得到底层实时系统的操作系统支持的实现。在最后的实现中，只有那些真正需要的对象被实现为活动对象。虽然 UML 既支持显式的并发模型也支持隐式的并发模型，但是它对隐式的并发模型有更好的支持。在早期就可以将活动类，异步通信和同步抽象成模型，然后渐渐地转换成实现环境中的服务和功能。

硬件的处理一般由硬件包装类来实现，硬件包装类提供访问硬件的接口。这些硬件包装类处理到设备的通信和设备产生的中断。通过这样的硬件类，可以隐藏低级协议的细节，并且很方便、快速地将低级中断转换成系统中其余部分的高级事件。包装类是属于活动的还是静态的类，主要取决于硬件的属性。

8.2 实时的概念

本节介绍用 UML 进行实时系统建模的一些基本机制。每一小节详细介绍这些机制是如何使用第 5 章中介绍的动态模型图。这些机制是活动类和对象(并发)的定义，活动对象间的通信、同步，如图 8-2 所示。

活动类被用来抽象现实世界中的并发行为，产生尽可能高效地使用系统资源的模型。活动类拥有自己的执行线程，且可以初始化控制活动。活动类(或活动类的实例，活动对象)和其它的活动对象并发执行。因而，活动类是并发执行“单元”。而只有当其它对象调用其操作时(发送消息给它)静态类的静态对象才能执行。

8.2.1 活动的类和对象

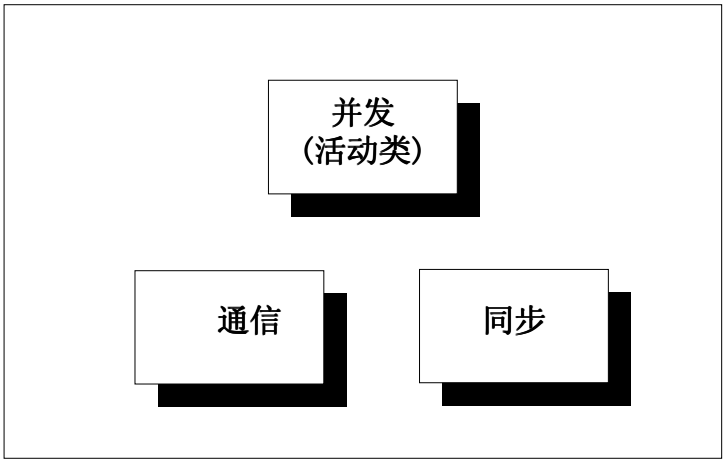

图 8-2　实时系统建模用到的机制

活动类被实现为进程或线程。进程是“重量级”的控制线程，而线程是轻量级的控制线程。进程和线程的重要区别是，进程有它自己的内存空间，而线程同其它线程一起共享同一个内存空间。共享的内存空间中包含了它们的共享信息(如共享对象)。进程从可执行程序启动。注意，在某些操作系统中，进程等同于线程。另外，可以将这些概念结合起来，在一个进程内，多条线程同享进程的内存空间。

活动类一般是通过类库来实现，类库中有一个超类，ActiveClass。所有的活动类均应该继承该超类中的下列操作：将进程或线程操作，如启动、停止、持起、恢复、优化级控制等，映射到底层操作系统中实现这些功能的操作系统调用。通常情况下，超类中的运行操作必须在具体的活动子类中实现。该操作的实现指定活动类的执行线程代码。典型情况下，运行操作在一个无限循环中执行。在该循环中，读输入信号或同步机制交互，执行活动类的工作。

活动对象通过通信和同步机制与其它对象交互。通信机制可以是对象间的普通的操作调用(在面向对象中的同步消息或是操作系统支持的某种通信方式，如邮箱或队列。同步机制主要是控制并发线程的执行，防止资源使用冲突)。

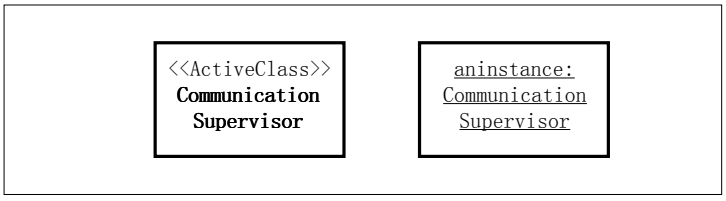

图 8-3　一个活动类及其对象

活动对象通常要比静态对象大，因为它们拥有或引用了许多其它对象。有时通过一个活动对象来控制一个包(UML 中的子系统)同其它包并发执行。但是，即使是在硬实时系统中，静态对象也要比活动对象多。创建线程或进程会带来一些开销，如果系统中定义了太多的活动对象将导致系统过于复杂，性能降低。

在 UML 中，活动类(或活动对象)用粗线方框来表示，如图 8-4 所示。如果方框内的

名字包含类名，则该符号表示活动类。如果方框内的名字是一个带下划线的对象名，则该符号表示一个活动对象。当类是版类<<ActiveClass>>时，在类图中，它应该是可见的，如图 8-3 所示。

经常采用嵌入的内部结构来描述活动类或活动对象，如图 8-4 所示。典型情况下，用几个其它的类来定义其内部结构。这些内部的类一般是静态类，但也可能是活动类。

活动类经常会根据其内部状态来改变它的行为(或，在某些情况下，根据系统的总的状态)。活动类的状态和它的行为可以用状态图来抽象，如在第 5 章中所述。状态图显示活动类的所有可能的状态，在每一个状态下，可以接收事件，可以根据发生的事件发生状态转换和执行一些动作，如图 8-5 所示。在状态图中没有显示时间约束。

8.2.2 通信

有许多机制来实现活动对象间的通信。活动对象必须能够并发执行，发送消息给其它活动对象而不需等待操作完成。最常用的通信机制主要有：

- 操作调用：调用对象中的操作。这相当于发送一条同步消息给一个对象，然后调用者等待操作完成并返回。
- 邮箱/消息队列：定义邮箱或消息队列的技术。发送者将消息放在邮箱中，接收者在同一点去读消息并处理消息。消息也可以是异步的，即发送者将消息放在邮箱中后，立即返回，继续执行。
- 共享内存：将一块内存专用作通信，两个或多个活动对象可以在这块内存上读、写信息。必须通过一些同步机制来保护这块内存，防止多个活动对象同时读、写这块内存。共享内存通常用于一些大的数据结构需要被几个对象处理时。

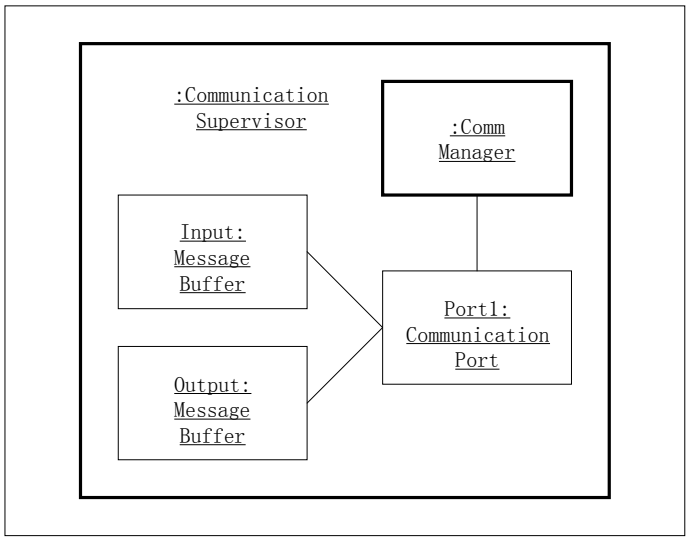

图 8-4　活动对象的内部结构图

- 同步：指定两条线程的同步点。当一条线程到达指定的同步点后，停下来等待另一条线程也到达对应的同步点。当两条线程都到达指定的同步点时，它们交换信息，然后又开始并发执行。
- 远程过程调用 RPC：RPC 技术允许将并发线程放在不同的计算机上实现和执行。

调用线程标识它想调用的一个对象的操作，然后将请求放入 RPC 库中。RPC 在网络上找到该对象，将请求打包并通过网络发送给目的对象。在接收方，将请求转换成本地格式，执行请求的调用。当调用完成时，将结果以同样的方式返回给发送方。因而，RCP 既是一种通信机制也是一种同步机制。

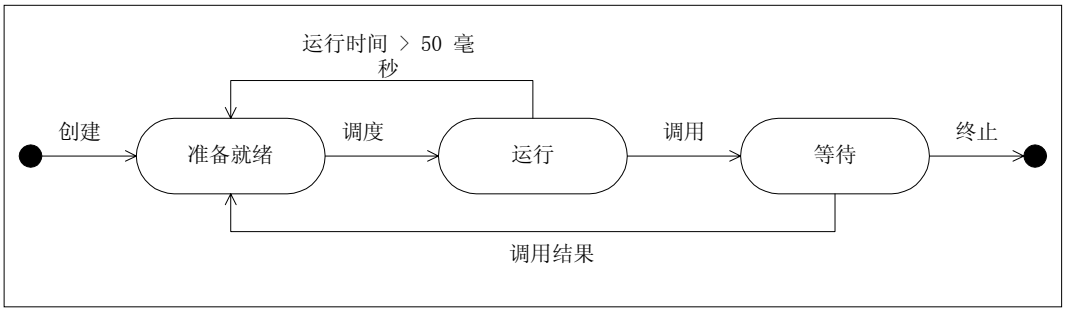

图 8-5　带有引起状态转移的事件的活动对象的状态

上面描述的通信机制都是与实现有关的技术。在建模时，活动对象间的通信用事件、信号和消息来描述。直到设计阶段，才需要选择真正的实现技术。

通信可以是同步的或异步的。异步通信是不可预测的，也就是说，事件可能在任何时刻出现，不可能知道一个指定事件在何时发送。同步通信则是可预测的，因为它在指定的时间出现。实时系统需要用到这种通信方式。一般来说，环境中的外部事件是以异步方式出现的，而系统中的内部通信则常以同步方式出现。在讨论对象间发送的消息时，异步通信和同步通信也用来指示发送消息的对象是否等待消息的处理。

(1) 事件

在实时系统中，事件是系统活动的驱动器。所有出现的事件必须是有定义的，且当事件出现时系统的行为也应该是有定义的。行为还与对象的状态有关，对象在不同的状态对同一事件作出的反应可能是不一样的。事件会引起状态转移。

在 UML 中，主要有四种不同类型的事件：

- 条件成真：即守卫条件成真是一种事件。
- 收到另一个对象中的信号：从一个对象发送一个信号给另一个对象。信号本身是一个信号类的对象，它包含属性和操作，也可以将信号看作消息。一般来说，信号是通过操作系统提供的机制来发送的，如邮箱或队列。可以异步发送信号。
- 收到另一个对象(或对象本身)的操作调用：一个对象调用另一个对象上的操作。可以将操作调用看作是同步消息。
- 经过指定时间间隔：经过一段时间间隔也是一种事件，有时也将其称为定期事件。一般用这种事件来描述下列情况：在处理过程中的延时或等待事件超时。通过 sleep 调用来实现这类事件，该调用挂起线程的执行到指定时间间隔，或当经过指定时间间隔后请求操作系统发送一个超时消息，并将该消息转发给请求的对象。

可以将事件分为逻辑事件和物理事件。物理事件是低级的，硬件级的事件；逻辑事件则是物理事件的高级形式。如，物理事件“端口 4H 上的中断”对应的逻辑事件可能是“红外传感设备上的告警”。因为逻辑事件处于较高的抽象级，模型更容易描述逻辑事件而不是物理事件。通过活动对象的状态图，将物理事件转换成逻辑事件。

可以通过描述事件的类别，优先级，处理时间请求或管理信息等特点来定义事件。为了实现这一点，可以将事件定义为类，将这些特点作为类的标志值，或对象的属性。

(2) 信号

在 UML 中，将信号定义为“可能发生的命名事件”。将信号描述成版类(<<signal>>)化的类，表示事件可能会出现在系统中。因为将信号描述成类，所以信号的属性和操作描述了事件的信息和行为。信号可以在系统中的对象间传递，以同步方式或以异步方式。

在类图中描述信号，实时系统中的所有信号均分层表示，如图 8-6 所示。从而可以将具有同一超类的所有对象组合起来。这样，信号的接收者就可以指定它接收指定超类的对象，即它接收指定超类的信号对象或超类的子类的信号对象。

信号是事件的一种特殊形式，因而，也可以将它们分为物理信号和逻辑信号，分别表示低级的硬件信息和事件的高级解释。

(3) 消息

对象通过消息来通信。消息的实现一般是一个简单的操作调用或放入邮箱或队列中的信号对象。在实时系统中，接收消息就是一个事件。可以在很多 UML 图中表示消息，如序列图、协作图、状态图和活动图。

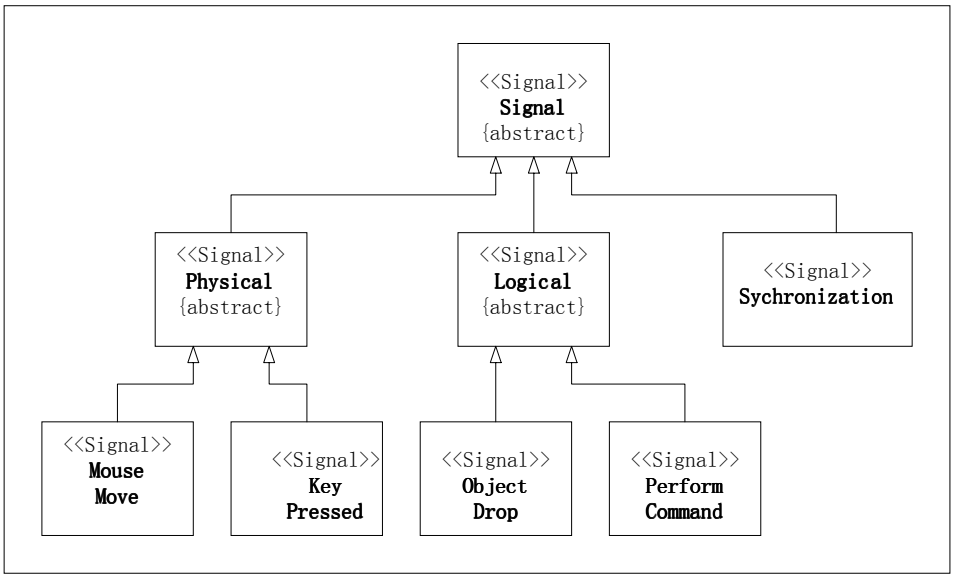

图 8-6　信号类的层次性

图 8-7 显示了 UML 中的消息类型。

- 简单消息：表示普通的控制流。它只是表示控制是如何从一个对象传给另一个对象，而没有描述通信的任何细节。这种消息类型主要用于通信细节未知或不需要考虑通信细节的场合。它也可以用于表示一个同步消息的返回；也就是说，箭头从处理消息的对象指向调用者表示控制返回给调用者。
- 同步消息：一个嵌套控制流，典型情况下表示一个操作调用。处理消息的操作在调用者恢复执行之前完成(包括任何在本次处理中发送的其它消息)。返回可以用一个简单消息来表示，或当消息被处理完毕隐含地表示。
- 异步消息：异步控制流中，没有直接的返回给调用者，发送者发送完消息后不需

要等待消息处理完成而是继续执行。在实时系统中，当对象并行执行时，常采用这类消息。

可以将一个简单消息和一个同步消息合并成一个消息，原同步消息的箭头和简单消息的箭头分别放在合并后的消息的两端。这样的消息意味着操作调用一旦完成就立即返回。

其它的消息类型是对基本的 UML 的扩展，如等待消息，只有当接收方准备好接收时才发送的消息)和超时消息，如果在指定时间内没有被处理就将被取消的消息)。这两种类型的消息和其它的变种在消息的版类中描述。

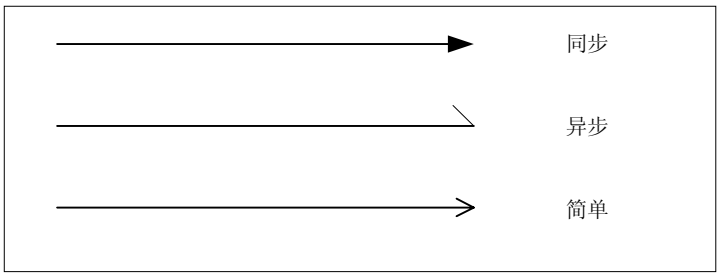

图 8-7　UML 中的消息类型

8.2.3　同步

同步的目的是为了协调并发线程的执行，即防止它们同时修改或访问共享资源，或按一定顺序交互。在任何并发系统中，同步是一种必须有的机制。

如果没有正确处理好同步问题，可能会导致下列问题：

- 不正确的共享访问：如果活动对象同时访问它们共享的资源或静态对象就可能导致不正确的结果。如果一个线程调用一个静态对象上的操作，该操作的功能是设置对象的属性。如果在执行操作的过程中，线程被操作系统挂起，调度器调度另一个线程执行。而另一个线程调用同一个对象上的同一个或另一个操作，则该对象就处于不安全状态(因为上一次调用还没有完成)。让我们假定，第二个线程调用的操作执行完成，然后被挂起的第一个线程恢复执行。此时，对象的属性已发生了改变，操作的结果是不确定的。为了解决这一问题，需要在这两个线程中使用互斥机制，使得一次只有一个线程执行该操作，其它的线程只有等到第一个线程完成操作调用才可以调用该操作。与共享访问有关的资源包括：对象和其它的共享资源(如打印机、通信端口等)。
- 低效率的资源利用：不同的执行线程间经常存在着相关性。一个线程可能需要另一个线程为其准备所需的数据。该线程不断地通过一个循环来查看数据是否准备好，这是对系统资源的极大浪费。一个更好的解决办法是，挂起等待数据准备好的线程，当它所需的数据准备好后通过一种机制通知调度器调度等待线程运行。
- 死锁：当许多线程相互等待对方时就出现了死锁。举例来说，假定有两个通信端口 A 和 B，通过某种同步机制保护使用。现在，两个线程都需要这两个通信端口。一个线程已得到了端口 A，正等待得到端口 B，而另一个线程则相反，得到了端口 B，正等待得到端口 A。而它们都不释放已得到的端口，就样，谁也执行不下去，出现了死锁。

- 饿死：饿死就是一个线程总是得不到执行。当一个线程的优先级很低使得它不可能或很难得到系统控制时，线程的任务总是得不到执行，从而导致其它的问题。

与同步机制有关的概念主要有：信号量，监视器和关键区。它们的共同点是保护一代码块，使得一次只有一个线程访问它。代码块包含保护或使用资源的代码。同步机制也包括等待资源释放的操作(而不是前面提面的循环等待)。同步机制需要操作系统的支持。

解决同步问题还与线程调度有关。很显然，优先级高的线程要比优先级低的线程更容易得到系统的控制权。在某些操作系统中，允许将调度参数化，甚至可以选择调度算法。最简单的调度控制策略是在活动对象的代码中，通过调用 sleep 等调用来主动放弃控制或等待通信或同步，如在邮箱上或信号量上等待。

UML 对同步的支持能力是有限的，但是可以通过版类或性质来扩展。一个类或一个操作可以通过给其并发性质赋值来指定其并发性要求，如给并发性质赋下列值：串行，守卫，同步。如果设置的是类的性质，则将影响类的所有操作，否则仅对指定操作有影响。这些值的含义说明如下：

- 串行：类或操作只能在单一控制线程中运行。
- 守卫：类或操作可以在多个控制线程存在的情况下使用，但是它需要线程的主动协作来完成互斥。通常情况下，线程在使用它们之前加锁，使用完毕后解锁。
- 同步：类或操作可以在多个控制线程存在的情况下使用，但类本身来处理同步。这一特点在现代编程语言中比较常见，如在 Java 中，可以将操作宣告为同步的，然后操作规程的同步处理是自动进行的，使得一次只有一个线程访问该操作。

如果必须更显式地表示同步，则需要定义信号量类，并在需要活动对象间同步的地方实例化该信号量类。另一种方法是定义一个版类信号量，所有类的实例均可共享并被信号量保护。另外，同步机制而不是信号量当然可以被抽象成类或版类。

如前所述，活动对象间同步的另一种技术是调度。调度决定选择哪一个线程执行，一般由操作系统通过分配处理器给线程来完成。

程序员也可以通过设置线程的优先级或设置调度算法的参数来部分控制线程的调度。在一个复杂的环境中，如果需要详细的调度控制，需要设计一个超级线程。超级线程有最高的优先级，根据一定策略给应用线程赋予不同的优先级。

在 UML 中，活动对象的优先级最适合表示成活动类的标签值。没有预定义的标签值，但很容易定义。值的范围起决于操作系统的分辩率，如可以分低、中、高，或从 1 到 32。通常情况下，在建模时设置的优先级需要在测试原型时或系统的早期版本中进行调整。

8.2.4 在 JAVA 中实现并发和同步

Java 语言支持线程。一般情况下，实时结构的质量和性能起决于执行 Java 程序的底层操作系统。下面的代码说明一个从预定义的 Thread 类继承来的活动类。作为继承的一部分，抽象的运行操作必须在类中实现。代码中定义了一个无限循环，在循环的内部，发送同步和异步消息。每循环一次就睡眠 10 秒钟。

线程对象在静态的主函数中被实例化。在主函数中，两个 DemoThread 类的对象被创建并被启动，它们将并发执行，由操作系统调度。在 Java 中，可能容易地实现前面所述的实时结构。

```
class DemoThread extends Thread
{
   public void run()
   {
      try {
         // Do forever
         for (;;)
         {
          // Synchronous message to System.out object
          System.out.println("Hello");
          // Asynchronous message placed in Global_mailbox
          // Needs definition of class Signal and Mailbox elsewhere
          Signal s = new Signal("Asynch Hello");
          Global_mailbox.Put(s);

          // Waits for 10 milliseconds.
          sleep(10);
         }
       }
       catch (InterruptedException e)
       {
       }
    }
    public static void main(String[] arg)
    {
       // Create an instance of the active class (thread)
      DemoThread  t1 = new DemoThread();
      // Start execution
      t1.start();
      // Create another instance of the active class
      DemoThread t2 = new DemoThread();
      // Start execution
      t2.start();
     }
   }
```

8.3 UML 的实时建模手段

本节介绍 UML 中的所有实时建模手段。在 UML 中，没有专门的图来描述实时性，但是可以将实时建模信息加到 UML 图中，特别是动态图中。UML 可以定义下列信息：

- 时间：最好在序列图中描述时间及时间约束。当然，也可以将时间说明作为注释加到其它的图中。

- 并发：用活动类来描述并发(用版类<<ActiveClass>>和粗线表示的类或对象方框)。活动类的属性(如，优先级)可以定义成类的标签值。可以通过组件版类<<Process>>和<<Thread>>将它们映射到实现环境中。
- 异步事件：UML 支持线程间发送异步消息。
- 同步：可以用类的属性或操作(并发性质)来描述同步，也可以定义信号量，监视器，或关键区的类/版类来描述同步。
- 分发：用分布图来描述线程在分布式系统的分布情况。

图 8-8 到图 8-11 说明一个房屋告警系统的例子。系统由一个主单元以及与主单元相连的许多传感器和告警器组成。传感器探测监控区内的移动，告警设备产生告警声音或光来警告入侵者。将监控区域分成一个小单元，每一个单元内包含一些传感器和告警器。当一个单元被激活时，告警功能启动，否则关闭监视。

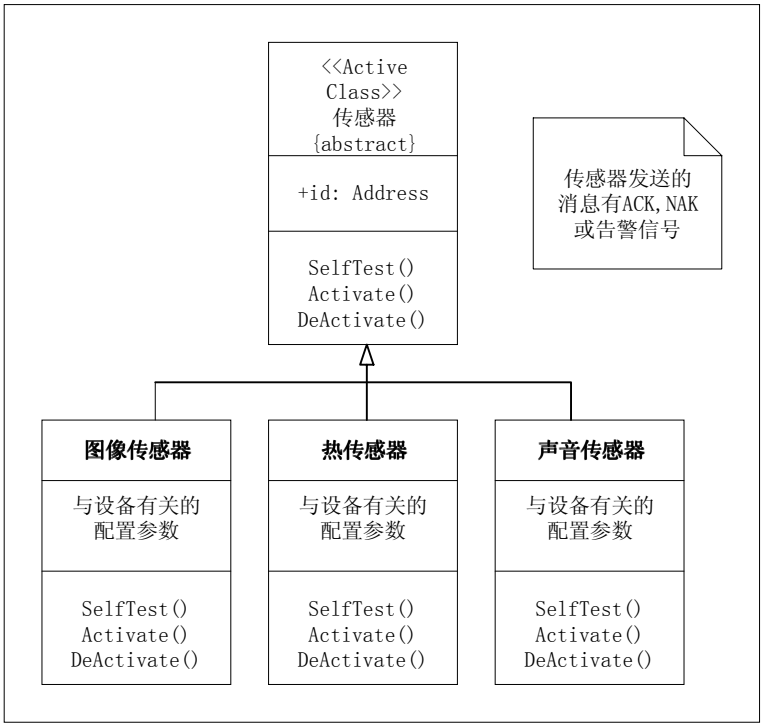

图 8-8　告警系统中的传感器

传感器是探测指定区域内的活动的设备，它们被抽象成 Sensor 类，这些类定义到传感器的接口。Sensor 类是活动的，它有自己的控制线程，主要功能是处理设备送来的低级中断。传感器可以被激活，关闭，测试。它还可以产生 ACK，NAK 和告警信号。

告警可被触发，关闭和测试，它产生的信号是 ACK 或 NAK。象传感器一样，将告警抽象成活动类，用来处理和设备的低级通信。

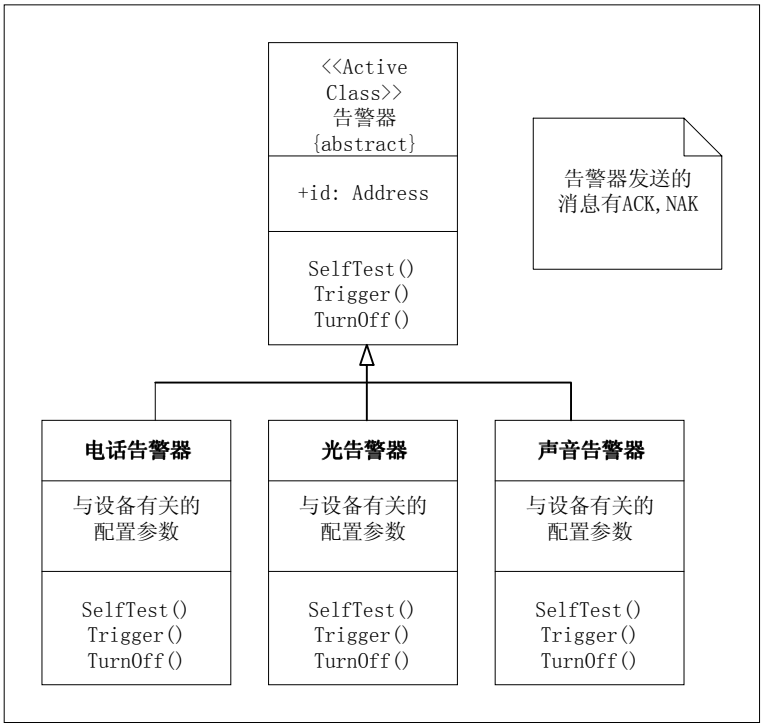

图 8-9　告警

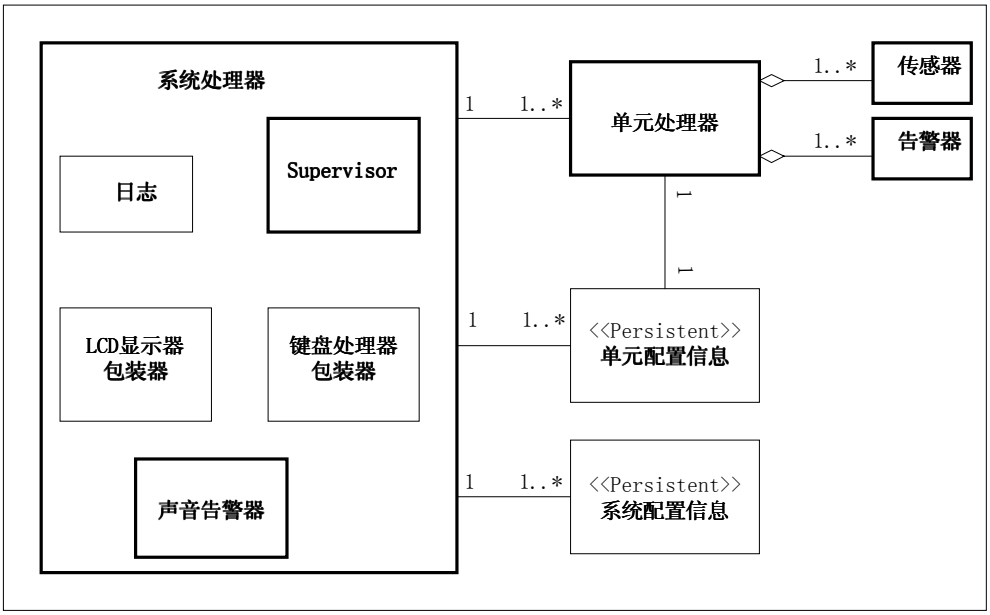

图 8-10　房屋告警系统中的活动对象和静态对象

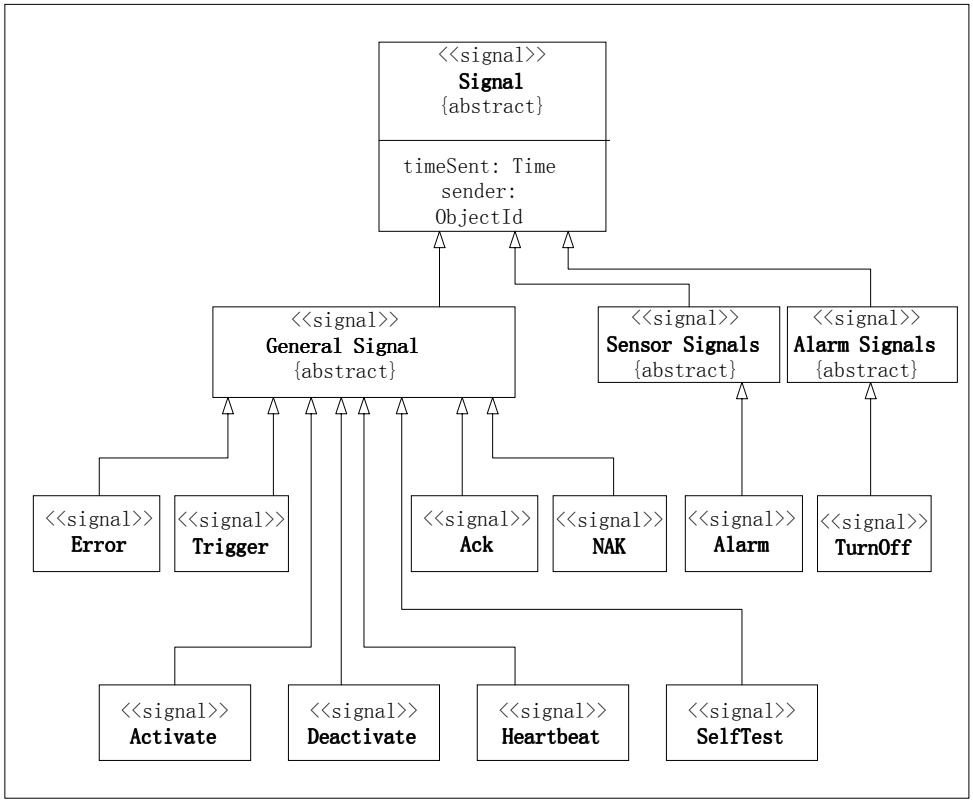

图 8-11　房屋告警系统中的信号的层次关系

系统的类图说明传感器和告警如何有机地结合起来完成监控功能。将传感器和告警连接到单元处理器。单元处理器是一个活动类，负责处理一个指定单元。一个单元处理器对象从保存单元当前配置信息的持续性对象中读出其配置信息。将单元处理器连接到系统处理器。系统处理器是一个活动类，负责处理通过带有 LCD 显示的控制面板与用户的交互。通过控制面板，可以配置、激活和关闭系统。所有的配置变化均保存在单元和系统配置信息类中。系统处理器还包含一个超级线程，它不断地同所有单元处理器通信。超级线程执行设备的测试，从单元处理器接收状态正常信号。它还接收单元的告警信号，并将收到的告警信号广播到其它单元中。系统处理器将所有事件保护在日志中。在系统处理器中，还有一个内部声音告警器，该告警器可以在任何情况下运行，即使与其它的告警器的连接被断开。

系统中的信号是房屋告警系统的静态结构的一部分。在本例中，有三类信号：一般的信号，传感器信号和告警信号。设备、系统处理器和单元处理器均可使用一般的信号。系统中的所有异步通信均通过信号来实现。

8.3.1　状态图

状态图的作用是标识对象的状态和行为，以及在对象生命期内在不同的状态下对象的行为变化。非并发性状态图在第 5 章中已有描述。可以将活动对象的状态细分为并发子状态，并发子状态下许多动作可以并发执行。子状态不必在它们自己的线程中执行，虽然常

有这种情况。

在状态图中，并发子状态的表示方法是：将状态框用虚线分隔成一些子区域，每一个子区域表示一个子状态，每一个子状态可以有一个可选的状态名，包含一个嵌套的状态图，如图 8-12 所示。

也可以用并发状态来抽象激活。子状态显示当前关心的告警，传感器和单元处理器的动作和状态。当且仅当所有的子状态都到达停止状态时，系统的总状态才被激活，否则，将系统放在激活失败状态。

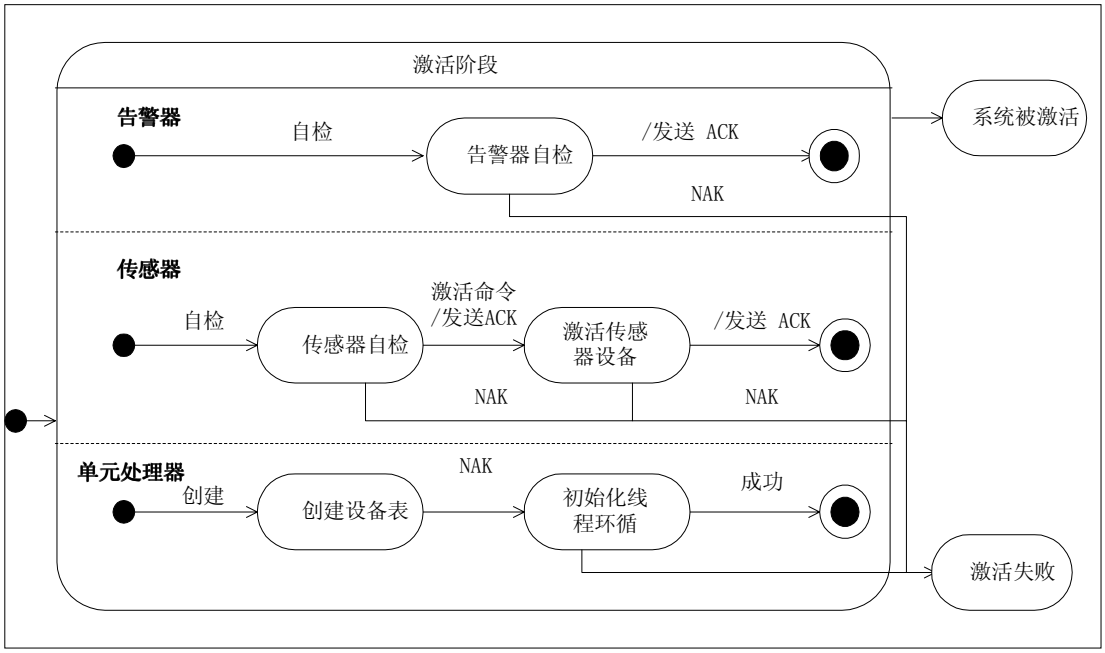

图 8-12　状态图中的并发子状态。图中还描述了系统的激活

也可以用复杂的状态转移来表示并发。复杂的状态转移可以有多个源和目的状态，可以将控制分成多个并发执行的线程或将多个并发线程合并成一条线程。在这种情况下，状态图的一部分可以并发执行。但是，同使用并发状态的区别是：复杂状态转移中的并发状态没有它们自己的状态机，它们是并发执行的同一级别上的状态。

用一条短的垂直的粗线条表示复杂的状态转移。可能有一条或多条实心箭头从某些状态(称为源状态)指向它的状态转移线，也可能有一条或多条实心箭头从状态转移线指向某些状态(称为目的状态)。在状态转移线的旁边可以有一个转移字符串。只有当对象处于某一源状态且转移守卫条件为真时，状态转移才发生，意味着并发执行开始或结束。

用复杂状态转移表示并发状态的例子如图 8-13 所示。在每一个子区域内的活动并发执行。

8.3.2　序列图

序列图描述对象的交互过程。在序列图中，最基本的元素是时间，从图的顶部向下看即可了解在某一时间对象的交互。因而，序列图是很适合描述实时系统中的时间和时间约束的。通常将时间说明放在图的边缘注释区。例如，两条消息之间最大时间间隔说明，一

个对象应该等待多长时间后才能给另一个对象发送另一条消息。作为一种辅助方法，可以在文档中定义时间标记(标记序列图中的某一点，如 A 或 B)，然后在后面的时间说明中引用它们(如 A - B < 10 秒)。

在序列图中，可以用不同的消息来表示对象间的不同类型的通信。简单消息表示消息类型未知或消息的类型不太重要。并发系统中的同步消息提供等待语义，也就是说，发送消息的对象等待接收消息的对象处理消息。异步消息则表示发送消息的对象不必等待消息的处理，而是在发送完消息后立即继续执行。传输时延可以用一条斜线箭头来表示，说明消息将在发送完成后的一段时间后收到。

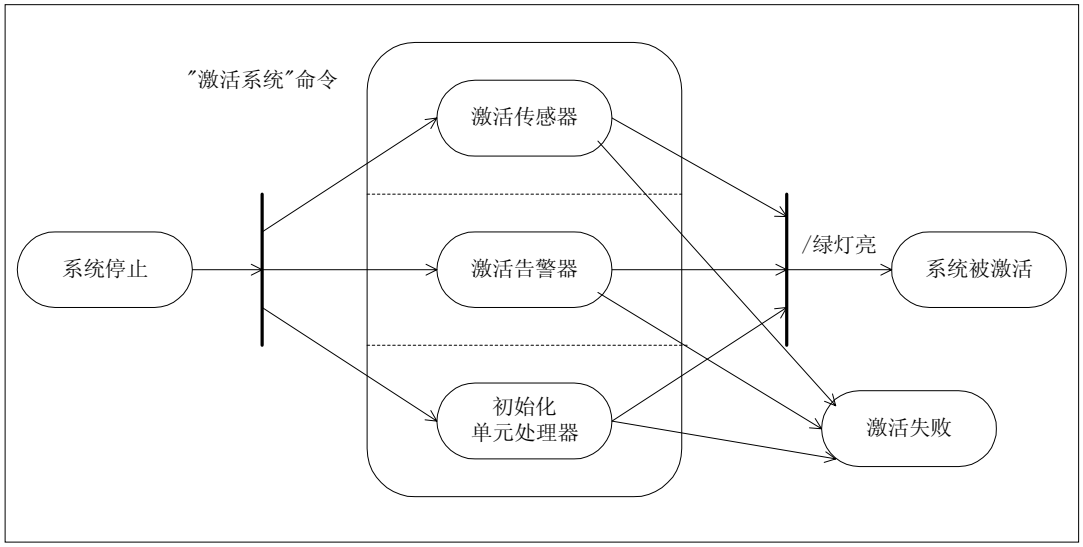

图 8-13　复杂状态转移图。图中控制流被分成并发执行的并发线程，并在后面又同步起来

活动对象用粗线方框来表示。激活用生命线上的一个窄方框来表示，方框的起点表示激活点，终点表示对象不再被激活。静态对象只有在处理消息的时候才被激活，当结果返回后激活完成。而活动对象一般在发送和接收其它对象时还保持控制和执行动作。

分枝、循环、递归、对象的创建和破坏的表示方法与非并发系统中的表示方法是一样的，这些内容在第 5 章中已经描述过。

图 8-14 描述房屋告警系统的激活序列图，图中系统处理器命令单元处理器激活它自己。单元处理器然后发送一条同步消息(操作调用)给单元配置对象，请求得到配置信息。收到配置信息后，单元处理器给所有设备发送自检信号，以确认它们是否工作正常。传感器也需要通过一个激活信号来激话。注意，传感器和告警器方框的名字下面没有下划线，即表示它们是真正的类。用类而不是在序列图中画许多对象的方式来表示用到的所有对象。传感器和告警设备之间的通信是异步通信。当所有的设备都被正确地测试和激活后，发送一个应答信号给系统处理器。

序列图描述了一次成功的激活的每一个细节。可以描述更一般的情况，在图中加入对错误的描述。注意：可以将时间加在图的左边。在本例中，从激活到系统直正被激活之间的时间间隔不超过 5 秒。

8.3.3 协作图

协作图描述了对象上下文(一组对象及对象间的关系)和对象间的交互(说明对象间如何协作来完成某一任务)。协作图适于描述实时系统中的对象间的交互，因为它能够说明参与交互的活动对象和静态对象的结构。在同一个协作图中，可以显示多条线程和并行发送消息。消息序列用序号来标志。

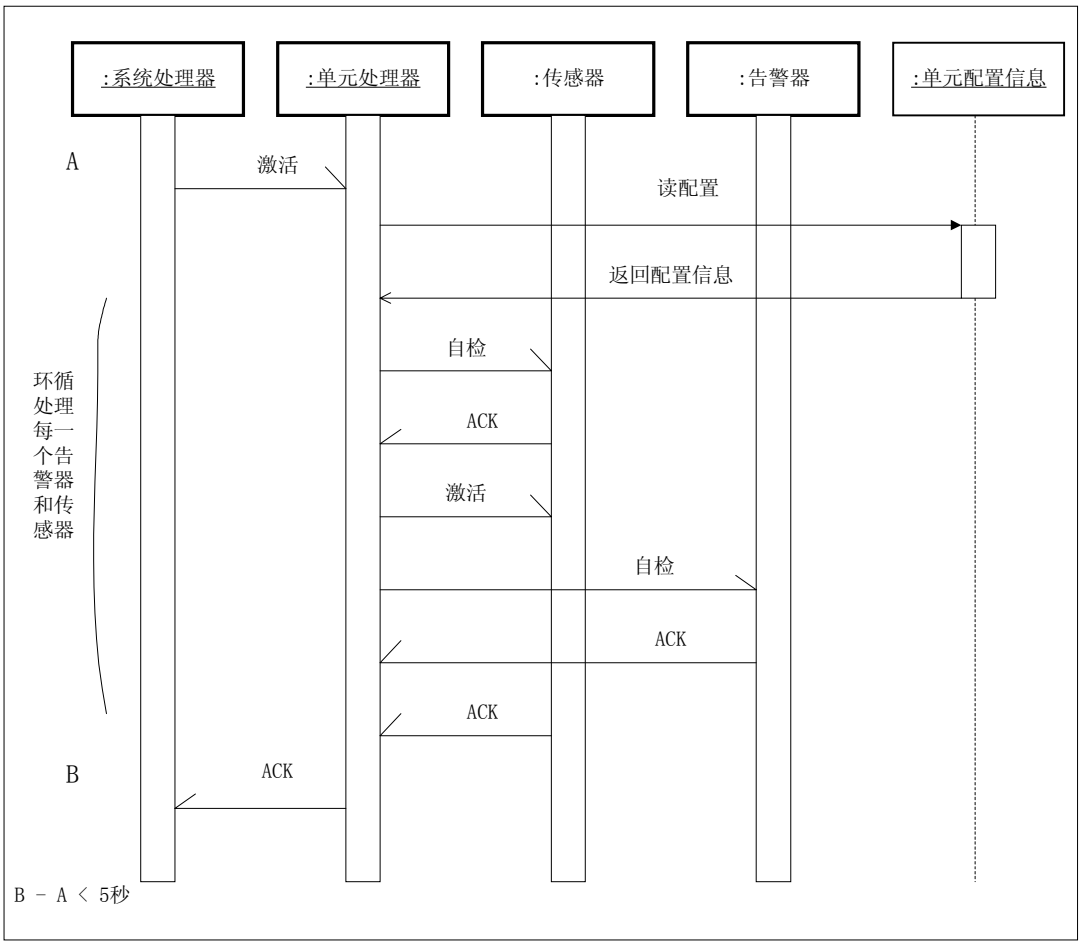

图 8-14 序列图中的告警激活序列。除了单元配置信息是以同步方式发送的外，其它消息均以异步方式发送

活动对象用粗线方框来表示，方框内常常有对象的内部结构。协作图中的消息类型同序列图中的完全一样。在协作图中，用的是包含消息序号的标签而不是序列图中的纵向时间序列。并行发送的消息用序列号表达式中的字符来表示，如协作图中的序列号 2.1a 和 2.1b 表示两条并行发送的消息。消息标签中的前缀说明也隐式包含同步消息，即前缀表示在消息被发送之前其它的消息必须被处理，这意味着可以用前缀来同步活动对象，使得工作是按一定顺序进行的，即使是在几个活动对象并发执行的情况下。消息标签也可以有一个守卫条件，即只有在守卫条件成真时才可以发送该消息。守卫条件中也可以包含同步条件，指定在使用该条件时某一资源必须是可用的。

可以在协作图中加入时间注释，虽然不如在序列图中那么明显。通常情况下，时间注

释是作为图中元素的注释或约束的形式出现。

图 8-15 中的协作图显示传感器探测到一些情况时的交互情况。当传感器探测到一些情况后，它发送一条告警信号给单元处理器。接着，单元处理器并行发送两个异步触发信号(本例中为电话和声音信号)给所有的告警器，发送一个异步告警信号给系统处理器。在系统处理器的内部，告警信号被同步处理：超级线程首先调用内部声音告警，然后将事件写入日志文件。协作图很好地描述了相关对象的结构和它们之间的交互。

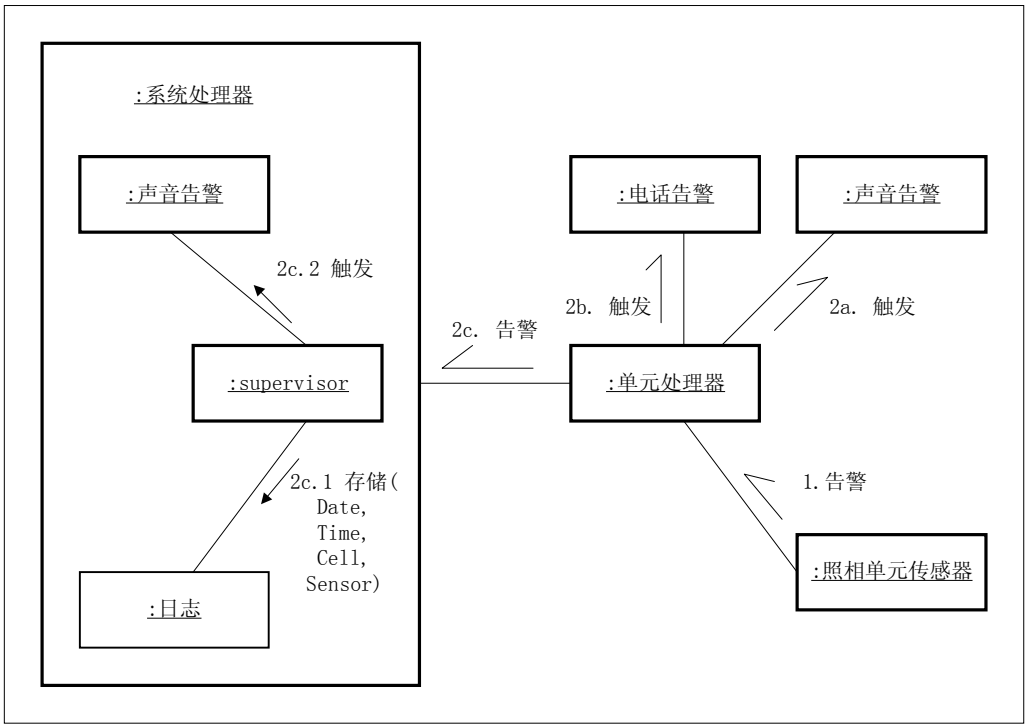

图 8-15　传感器探测到有东西在移动，产生一个告警

8.3.4　活动图

活动图主要用来描述串行流控制，它一般与类的操作相连，因而它与实时性的关系不如其它的动态图密切，虽然在描述实时系统中它也是有用的。

描述实时系统时，活动图的一个可能用处是：指定一个活动类的运行操作，因为运行操作显示类的对象的并发行为。图 8-16 显示单元处理器的运行操作。该操作等待信号，然后根据收到的信号，执行相应的动作。处理完成后，它给系统处理器发送一个信号，请求从操作系统发来一个超时信号，收到超时信号后返回到开始，继续等待新的信号。图 8-16 中，处理激活、关闭、超时信号的活动被放进一个超级活动以免活动图变得太复杂。

8.3.5　组件和展开图

组件图和展开图描述系统的物理构架，即描述类是如何在不同的代码组件中实现，如何将最终的可执行组件分配到节点上执行。

这两种图与实时性有关的方面就是将活动类分配给组件，实时性在分配的组件中实现

和执行(组件可以是源代码组件，也可以是可执行代码组件)。组件可以带有<<Process>>或<<Thread>>版类，说明活动类是作为进程还是线程来实现。

在房屋告警系统中，显示的物理构架是：一个带用户面板的主单元与许多传感器和告警器相连。所有组件均属于主单元，主单元包含系统的所有软件，所有线程，如图 8-17 所示。

8.4 如何适应实时系统

虽然 UML 有许多机制支持对实时系统的建模，但是使用 UML 扩展的机制来增强 UML 对实时系统的描述能力仍然是很必要的。这些扩展的能力使得利用 UML 可以描述某一特殊的方法，组织或应用领域。这些扩展的机制在第 7 章中已详细讨论。

可以用标准版类<<ActiveClass>>来定义一个类(或类的对象)是活动的。版类用粗线条来表示。还有另外两个标准版类，<<Process>>和<<Thread>>，用来描述组件是作为进程还是线程来实现。更多的版类来描述逻辑模型到实现环境的映射，如<<Semaphore>>, <<Mailbox>>, <<Exception>>, <<Interrupt>>, 等待。还可以利用版类来解释模型，如版类<<HardwareWrapper>>或<<Scheduler>>的作用就是说明某一个类或组件的角色。版类还可以定义不同类型的消息。

性质(标签值)的作用是定义类或操作的并发性(标签值为 Sequential, Guarded, Synchronized)。用户定义的标签值一般用来定义活动类的优先级标签或其它的调度信息。

在实时系统中，经常要用到时间约束，虽然可以在任何图中需要时间约束的地方使用它，但它最常用在序列图中。

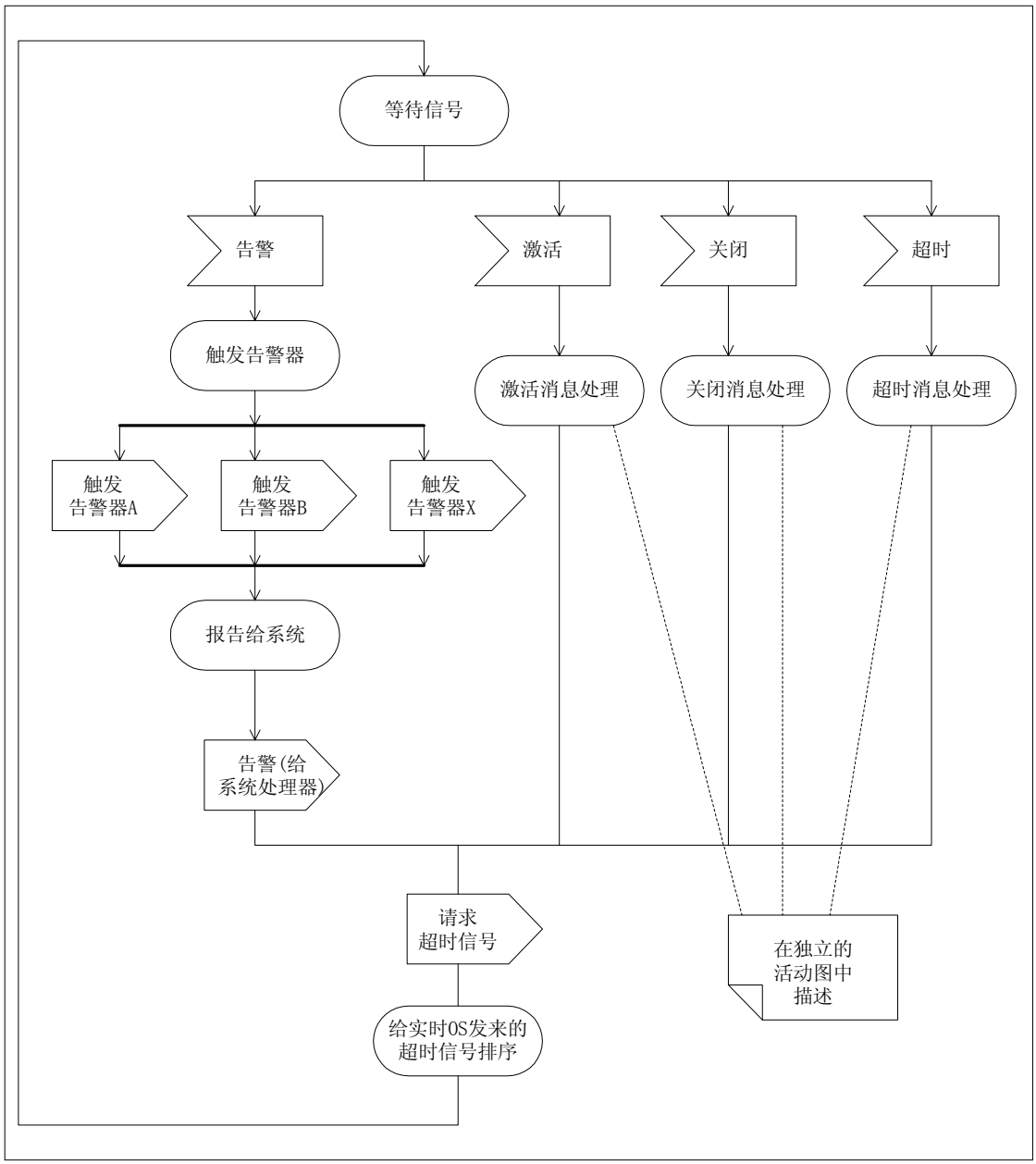

图8-16　为活动单元处理器类定义的运行操作。在活动图中，并发活动可以同这些活动的同步一起显示。处理激活、关闭、超时信号的活动被放进一个超级活动中，但也可以将这些活动用独立的活动图来描述各自的细节

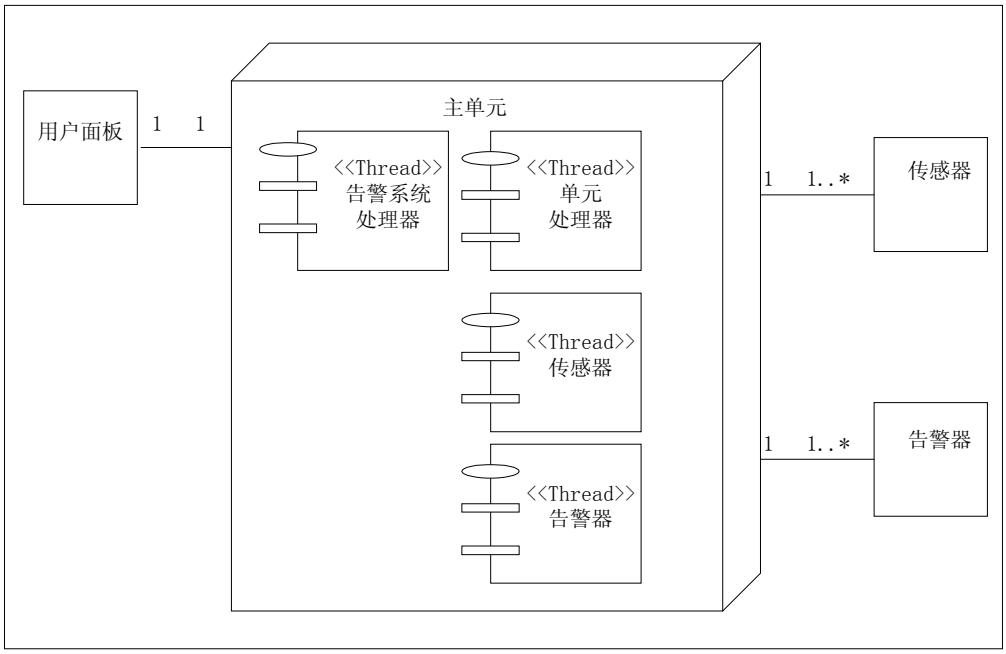

图 8-17　房屋告警系统的展开图

8.4.1　其它的与实时建模有关的事项

在实时系统建模时，并发、通信和同步是最重要的因素，但是容错、性能优化和分布式也是必须要仔细考虑的因素。

(1) 容错

容错指的是当系统中的软、硬件失败时系统仍然能正常工作。在许多实时系统中，如飞机控制系统，在任何情况下都不允许系统失败。在一个高度可靠的系统中，结果是可预测的，系统是健壮的。系统必须能稳妥有效地处理错误，在任何情况下系统都处于可操作的状态之下。有许多容错技术用于实现这一点：

- 错误和意外处理：一部分容错能力可以通过处理“正常”的错误和意外来实现。这些可处理的错误必须是可知的，额外的意外捕获也必须是已定义的。将意外处理加在程序代码中可能出现意外的地方，使得系统可以连续工作。
- 备份：备份是指系统中的软件、硬件一般有两个或多个副本，它是最常用的容错技术。在备份系统中，故障单元直接被备用单元来代替。
- 形式化证明：可以用数学方法来分析系统中的并发执行和消息交换，从而发现系统中可能存在的问题，如死锁、饿死、发并使用等。

在 UML 中，有几种机制被用来描述容错性。如果通信不是可靠的，所有的序列图中必须包含对消息丢失的说明。在这些地方，一般要求用超时来防止消息丢失。在状态图中，在最高级上定义错误状态，从而当任何不期望的错误出现时，对象进入错误状态，处理发生的错误。

(2) 性能优化

为了获取较高的系统性能，需要对基于模型的实现进行优化，在优化时应遵循以下一

些原则：

- 尽可能将活动对象实现成线程而不是进程。
- 使用同步通信(如，操作调用)而不是异步通信。
- 重用消息对象或其它的不断被用到的对象，尽量避免重新创建它们。
- 避免在不同的内存空间内复制对象。

优化的第一步也是最重要的一步是检查总体设计，而不是优化最低层次。

(3) 分布

实时系统一般是分布式系统，在 UML 中用展开图来描述。如果系统中有 OLE 或 CORBA，在不同节点间发送消息和对象是很方便的，否则，必须实现相应的分布机制。

分布机制包括：将复杂对象打包成字节流或从字节流中解包，分布对象的识别和定位。还需要有一个服务器进程或线程来完成对象上的操作的注册和解析。

8.5 小　　结

实时系统具有下列一些特点：时间是很重要的，有并发执行的线程，非确定性等。为了给实时系统建模，必须描述时间要求、并发性、异步通信和同步通信。实时系统常常是嵌入式系统。可以将实时系统分为两类：硬实时系统和软实时系统。在硬实时系统中，不允许系统失败，因而必须考虑容错性。

实时和面向对象集成在活动类和活动对象的定义中。活动类有自己的控制线程。活动类和活动对象被实现成进程或线程，具体实现需要操作系统的支持。

活动对象通过一般的操作调用(在面向对象中称为同步消息)或特殊的机制，如邮箱或消息队列(异步消息)来通信。活动对象的行为用事件来描述，异步消息信号也是一种事件。其它的事件有：守卫条件成真，经过一段时间，收到同步消息等。

同步机制主要有信号量或关键区，其作用是保护共享的资源，或避免无效率的等待。UML 支持定义活动类。活动类的行为可以用状态图中的并发子状态或复杂转移来描述。活动类和活动对象可以用在各种图中。协作图最适合描述并发行为，与实时系统建模的关系最大。