基于 UML 的可执行模型实现机制研究

付 佩[1]，谷青范[1,2]
(1.南京航空航天大学，南京 210016；2.中国航空无线电电子研究所，上海 200241)

[摘 要] 模型驱动开发（MDD）是由 OMG 提出的通用软件开发标准，旨在解决软件开发过程中遇到的需求多变以及重复开发等问题。模型驱动开发一般使用 UML 作为建模语言，然而 UML 是一种符号化语言系统，其语义采用自然语言描述，是半形式化语言，无法精确和严格描述模型行为从而实现模型的验证。为了解决这个问题，提出一种 UML 模型验证的方法，其核心是 xUML 的状态图，xUML 是 UML 的增强子集，拥有精确的语义。并在分析 Rhapsody 可执行框架 OXF 的基础上，设计并实现了一个模型可执行框架。最后通过一个模型执行的例子验证了该框架的可用性。

[关键词] 对象可执行框架；状态图；模型验证；统一建模语言(UML)
[中图分类号] TP311.52 [文献标识码]A[文章编号]1006-141X（2013）01-00010-05

Research of Executable Model Realization Mechanism Based on UML

FU Pei[1]， GU Qing-fan[1,2]
Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China;
2. China National Aeronautical Radio Electronics Research Institute, Shanghai 200241, China)

Abstract: Model-driven development is a general software development standard proposed by OMG, aiming at eliminating such problems as requirement change and duplicated development. Generally, it uses UML as the model language. However，UML is only a notation, with no formal semantics attached to its individual diagrams. Behavior definition and expression are not precise and detailed enough to execute model verification. In order to address the problem, a validation method based on Executable UML is proposed in this paper, which uses statechart as its core model. Executable UML, which could be viewed as an enhanced subset of UML with precise semantics. Furthermore, we design and implement a framework which can execute the model after analyzing the Object Execution Framework. Finally, we verify the availability of framework through a example of statechart simulation execution.

Key words: object execution framework (OXF); statechart; model verification; unified modeling language(UML)

1 引言

目前急剧增加的软件复杂性性以及需求变更引起的重复开发使软件开发周期更长，需要更多的人员协调开发，软件开发费用急剧增加。

为了解决这些问题，国际对象管理组织（OMG: Object Management Group）提出了 MDA 开发方法，它是一种开放、中立的系统开发方法和一组建模语言标准集合。模型驱动开发的原则是促使模型使用范例通过不同层次模型的抽象和转换去驱动实现一个应用程序。与传统的开发方法相比，模型驱动

开发方法有着明显的优势，它能进行高效的软件开发，缩短了开发周期。

模型驱动开发普遍使用统一建模语言（UML：Unified Modeling Language）作为模型驱动开发的语言。UML 是一种可视化建模语言，它通过用例图、类图、状态图等一系列图形符号来描述特定系统，支持不同层次的系统抽象，能够清晰而准确地描述特定系统结构、功能和行为。然而阻碍 UML 应用的最大问题是 UML 所建立的面向对象的系统模型是一种非形式化的模型，这种模型由一系列的图形符号构成，而这些符号并没有严格的形式化定义，因此无法通过模型的执行和仿真对系统的行为进行严格的验证，从而使得模型驱动开发系统正确性无法保证。

目前支持可执行模型驱动开发的工具是 IBM 公司推出的 Rhapsody 产品。Rhapsody 提供一个完整的从分析、设计直到代码实现和软件测试的开发环境，它适用于复杂实时嵌入式应用软件，能够对需求的覆盖和需求的变更情况进行分析，并采用模拟、执行和动画演示的方式对模型的正确性加以验证，最终生成可执行的完整应用程序。

本文在研究 Rhapsody 可执行模型的实现机制的基础上，设计并实现一个模型可执行框架，能够很好地支持 UML 模型验证，最后通过一个例子对设计的可执行框架进行了验证。

2　可执行模型

可执行模型是在建模的过程中能够通过模型的执行来验证系统的关键需求。现阶段许多工具软件都已能生成静态模型的代码，但是这些代码只是系统静态结构和属性的描述信息。实现模型的可执行仅仅有静态模型描述结构的代码信息远远不够，还要生成行为模型对系统动态行为的描述代码。

为生成行为模型代码，本文使用可执行（Executable）UML 作为建模语言，它具有精确的语义信息，并且与 UML 有相同的符号表示，集成了状态图的形式化语义。

本文构建的可执行模型使用类图描述系统的结构，使用状态图描述对象的生命周期各个阶段的动态行为。将系统中的概念实体抽象为类，实体对象在生命周期中的动态行为采用一个状态图来描述，它是对象交互的具体实现细节。状态图描述了对象生命周期的一系列快照，对象的每一个关键阶段定义为一个状态，通过外部激励事件引起状态跳转，使得状态离开当前状态到达新的状态。

系统静态和动态模型构建完成后，现阶段的主要任务就是生成静态模型和动态模型的代码，鉴于类图的代码生成规则比较简单，在此就不赘述。状态图的代码生成规则在后面状态解析引擎中介绍。在类图和状态图代码生成后，程序其实还是无法执行，因为缺少支持状态转换的机制，状态机执行之后无法对外部激励事件作出反应，所以需要设计一个支持状态机执行的框架。下文主要对可执行框架和代码生成规则进行研究。

通过上述分析可见，实现模型可执行研究的重点是状态机的执行。我们采用事件驱动框架作为状态机执行的驱动机制。用户注入事件到事件驱动框架，事件驱动框架进行事件分发和管理，在适当的时候分发事件给状态机对象，进行事件的处理，达到可执行的目的。这个过程需要状态解析引擎的配合，该解析引擎负责对状态图模型进行解析，根据状态图语义规则解析成具体实现语言代码。

状态解析引擎是状态机执行的基础，它负责把绘制的状态图解析为计算机可理解的代码。状态解析引擎对绘制的状态图模型元素按照指定的规则进行提取，然后根据状态图的语义生成事件框架可调度的代码。状态机主要元素包括：状态、事件、转换。具体的代码生成规则如下：

状态：每个状态生成一个枚举值变量代表状态。

状态转换：每个状态生成对应的转换处理函数对状态发生的转换进行处理。

事件：继承于可执行框架的事件类，使用唯一的事件 id 来标记事件。

3　状态机实现方法

状态机的实现有三种方式：嵌套 switch 方式，状态转换表方式以及面向对象方式。

3.1　嵌套 switch 方式特点：

简单便于实现。对于状态采用定义信号和状态枚举值，并且消耗内存比较少。但是代码复用性较差。一个复杂的状态机的代码会变大很庞大。必须

对于每个转换编写进入和退出动作，这使得代码难以维护。

3.2 状态转换表方式特点：

它使用直接映射的状态表来表示状态机。状态和信号的枚举值作为状态转换表的下标，因此它们必须是连续的。状态转换表的初始化比较复杂必须要使得状态和信号的枚举值匹配。而且增加一个新状态需要更新整个状态转换表，所花费的时间较长。

3.3 面向对象方式特点：

面向对象状态机表示依赖继承，多态性等语义以及面向对象的语言，从而使得转换更加高效。实现了事件分发的晚绑定机制。并且不需要状态枚举值和事件枚举值。通过状态指针进行事件处理。

我们的状态机实现采用的是面向对象和嵌套方式的结合，它很好地保留了嵌套方式简单方便实现的特点，并且不会产生状态爆炸现象。下面我们主要对于状态机框架的具体实现细节进行讨论。

4 可执行框架实现

4.1 可执行框架

可执行框架由三部分组成：事件队列，事件分发器以及事件消费者。

事件队列：所有事件都要加入到事件队列中才能得到处理。事件有外部事件、触发事件以及延误事件等。

事件消费者：负责事件的处理。针对不同的事件生成不同的处理函数。由事件分发器发送事件到对应事件消费者，再由具体处理函数处理。

事件分发器：它驻留在一个无限事件循环中，不断读取事件队列中的事件，然后分发给指定的状态机事件处理程序进行事件处理，如图 1 所示。每个事件都有一个指定的事件，id 事件分发器根据获取的事件进行匹配，分发事件到对应的事件处理程序。事件队列和事件分发处理器封装在一个活动类中。

框架的各个方面各司其职，组成了一个结构灵活、可扩展的软件架构，能够无缝地应用于状态机的验证中。

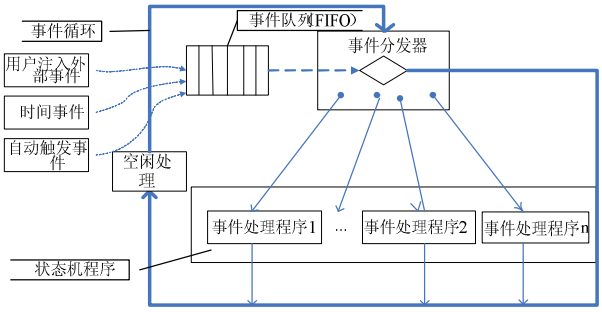

图 1 可执行框架

框架主要由活动类、响应类、事件类以及时间事件管理器类组成（如图 2 所示）。时间事件类使用操作系统提供的定时器进行计时，通过专用的时间事件管理器对时间事件进行管理；活动类（Active）是事件驱动框架的核心类，拥有一个事件队列和执行线程，由底层操作系统提供支持，负责事件的调度，它内置了一个事件分发器，发送事件到对应的响应类处理函数进行处理；响应类（Reactive）是事件的消费者，负责对触发事件、时间事件和信号事件处理以及状态机的初始化。下面就框架实现的关键技术进行讨论。

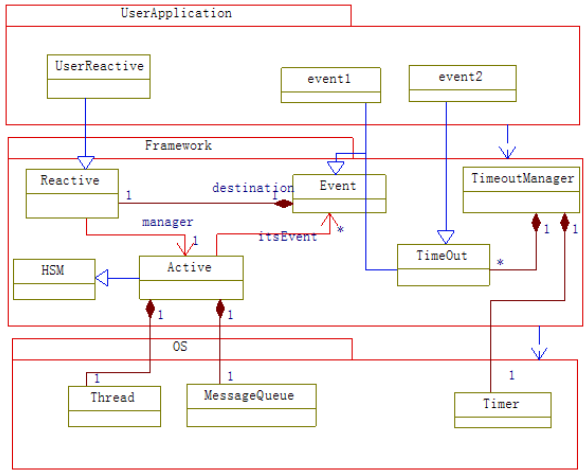

图 2 框架类结构

4.1.1 事件分发机制

由上述分析可执行框架活动类指出，事件分发是活动对象的重要功能，现在就分发机制的实现机理进行阐述。事件分发机制应该拥有事件的广播、事件订阅、事件出版等功能。具体分发处理流程如图 3 所示。

4.1.2 信号事件处理

信号事件是最常用的事件，它对应于对象之间的异步通讯。继承与框架的事件类。通过前文所述

的事件 id 进行事件判别，其具体的处理顺序图如图 4 所示。

4.1.3　超时事件处理

时间事件称为超时事件。超时是特殊的事件，继承于框架的 Timeout 类。通过时间事件管理器(TimeoutManager)对时间事件进行管理，它运行于自己的独立线程内且聚合了一个定时器对象。定时器对象每隔固定的时间片会向定时管理器发通知信号。每当定时器发出通知，时间事件管理器检查它的超时集合，把到期的超时插入活动类的事件队列中，由活动类对象进行分派时间事件，其处理顺序图如图 5 所示。

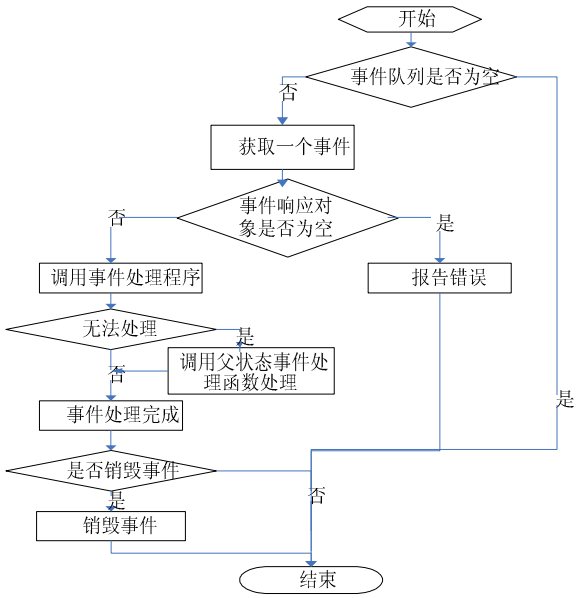

图 3　事件分发处理流程

5　实现验证

本章节通过门禁系统设计中进入控制模块状态图的模拟执行对可执行框架进行了验证。门禁系统中进入时需要进行两次检查，首先进行门禁卡合法性的检查，然后进行任务身份验证，完成后才可进入安全区域。系统的进入控制模块的状态图如图 6 所示。

通过 gen（reqReadSecurityCard）注入外部事件到状态机，事件管理器从事件队列中取出事件，把事件分发给对应的响应类对象，响应类对象调用状态事件处理函数对事件进行处理，处理完成后进入 ProcessingSecurityCardData 状态。状态机执行过程中状态变换通过红色线条显示，在动画展示过程中能够清楚的看到状态的变迁过程，设计人员通过观察系统接收外部事件后是否按预想的过程执行，达到验证系统功能和逻辑的正确性目的。如果偏离了预想状态可以对状态图进行适当的调整。再次执行状态机进行检验。

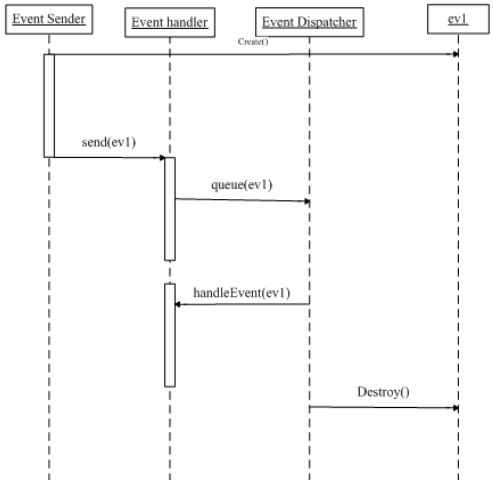

图 4　信号事件处理顺序图

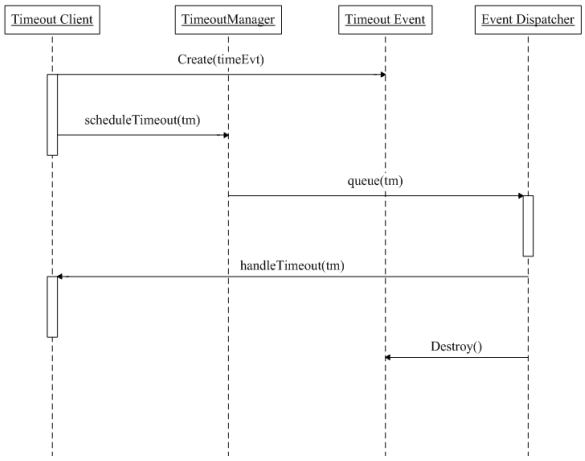

图 5　时间事件处理顺序图

6　结束语

文章深入分析了模型可执行机制，提出采用 UML 的增强子集 xUML 进行可执行建模，并使用状态图作为可执行建模的核心模型，在此基础上设计并实现了一个可执行框架，最后通过门禁系统状态图执行的例子对框架的可用性进行了验证。

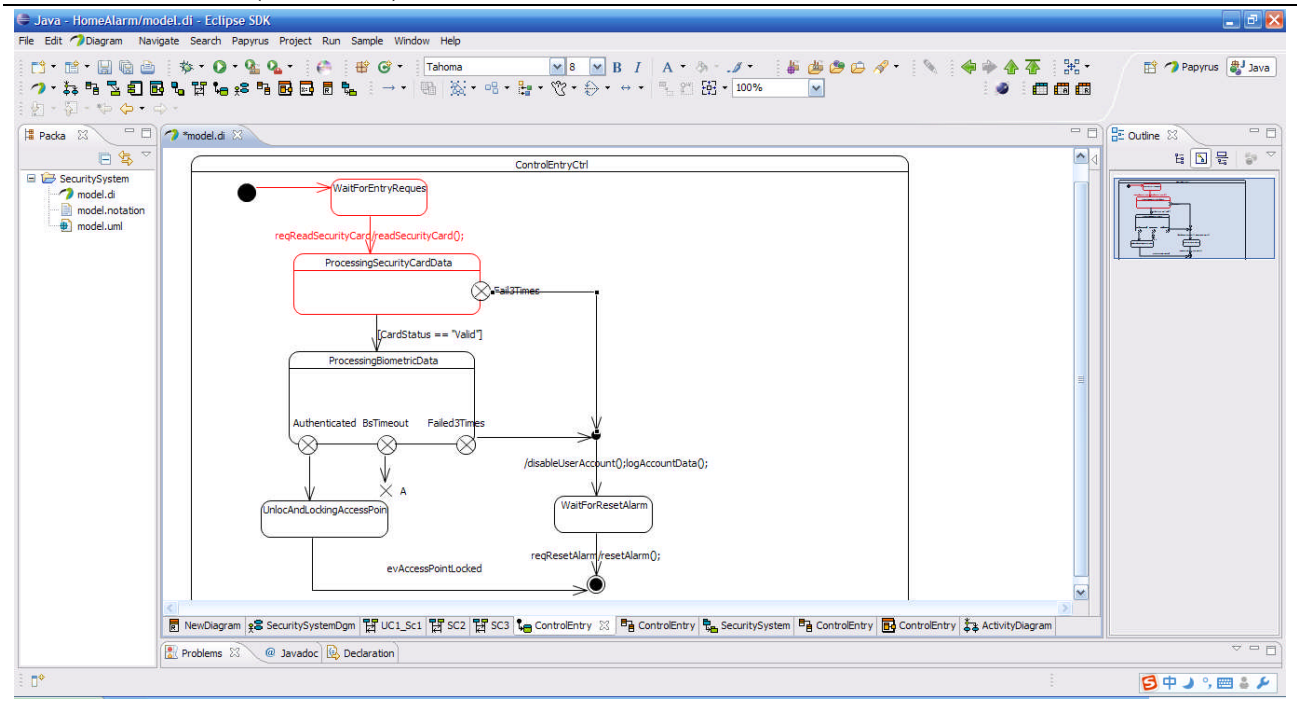

图 6 门禁系统状态图

参 考 文 献

[1] I-Logix Inc. Code Generation Guide[C]. USA: I-Logix Inc., 2003.

[2] IBM Corporation. C++ Framework Execution Reference Manual[C]. USA: IBM,2008.

[3] Douglass, Bruce Powel.UML Statecharts[C]. USA: i-Logix, 2004:1-23.

[4] Niemann, Scott. Executable Systems Design with UML 2.0[C]. USA: i-Logix Inc., 2004:1-12.

[5] D. Harel and M. Politi. Modeling Reactive Systems with Statecharts: the STATEMATE approach[R]. USA:McGraw-Hill, 1998.

［收稿日期］2012-11-10

［修回日期］2012-11-21

［作者简介］付 佩（1987－），男，硕士研究生。研究方向：软件工程、模型驱动开发。

谷青范（1974－），男，副教授。研究方向：分布式计算、软件工程、嵌入式系统开发。