基于 UML 的面向方面建模方法*)

刘瑞成　张立臣
(广东工业大学计算机学院　广州 510090)

摘　要　面向方面编程(AOP)技术通过横切关注来实现软件系统。面向方面建模可通过扩展 UML 来实现,在元模型层次上设计面向方面软件开发(AOSD)模型框架。从结构建模、行为建模、方面织入以及代码产生几个方面实现面向方面的建模方法。利用 UML 类图实现方面的结构模型,方面与核心组件及方面之间的静态模型关系。协作图表达方面与核心组件及方面之间的动态行为,并通过状态图细化方面和核心组件的动态行为,实现状态图的方面与核心组件及方面之间的织入关系,最终实现方面代码的自动生成。基于 UML 的方面建模填补 AOP 与 OOP 技术之间的空白,提高软件设计的模块化,增强代码的重用性、维护性。

关键词　面向方面,UML,元模型,框架

Aspect-Oriented Modeling Method Based on UML

LIU Rui-Cheng　ZHANG Li-Chen
(Faculty of Computer Science, Guangdong University of Technology, Guangzhou 510090)

Abstract　Aspect-oriented programming technology implements the software system using crosscutting. Aspect-orientation can be modeled by extending UML, and designed the AOSD model profile in the meta-model. The profile models aspects from the structural model, behavioral model, aspects weaving and code generation. UML class diagrams realize the aspect structures, and the static relationships between aspects and core components or other aspects. The collaboration diagrams express the dynamic behaviors. The statecharts refine the behaviors of aspects and core components, and realize the weaving of aspects, so as to enable automatic code generation of aspects. Modeling aspects by UML will fill the gap between AOP and OOP. It improves the software modularization, and makes the code more reusable and maintainable.

Keywords　Aspect-orientation, UML, Meta-model, Profile

1　引言

面向方面编程(AOP)是一种基于关注分离的新技术,系统不同的关注能够被分离出来并进行单独的设计,可以解决面向对象编程不能简单解决的复杂问题。在以往的过程化程序设计和面向对象编程技术中有些编程问题并不能很好地实现,某些程序设计代码分散在系统各个模块中,从而导致系统难以开发和维护[6],面向方面编程技术就能很好地解决这个问题,并提高了模块的重用性。面向方面软件设计方法把系统建模分成两部分:核心组件(基本元素)和方面。

面向方面建模技术允许系统开发者在系统设计时,从核心功能性需求中分离出不同的关注,例如实时性、安全性、错误和异常处理、日志、同步控制、调度、性能优化、通信管理、资源共享、分布式管理等[8]。同时通过支持方面的组合和绑定来实现系统的集成。关注分离改进系统的设计,开发者只需要实现单独的方面而不必过多地考虑其它方面和系统的核心组件。

与面向对象编程技术一样,AOP 也需要对面向方面系统进行建模,但现在还没有统一的标准的建模方法。因此需要探索面向方面的建模方法。本文试图通过对统一建模语言 UML 的扩展来实现面向方面的建模。统一建模语言 UML 具有以下的特点[11]:

· UML 是面向对象的工业化标准建模语言,能够在整个软件生产过程中很好地对软件系统进行建模。

· UML 是一种能够广泛运用于各个应用领域的建模语言,具有丰富的模型分析和设计技术,利用不同的视图来构造系统的结构和行为模型,从而实现系统的建模。

· UML 是一种可扩展的建模语言,通过不同的扩展来满足不同领域的建模需要。

特别是 UML 提供的扩展机制[5,14]允许对特定领域进行扩展满足它们的建模需要,例如实时系统的 UML 扩展[19],CORBA 的 UML 扩展[20]等。因此,可以通过扩展 UML 来对面向方面系统建模。

现在一些学者已经开展了对 UML 进行扩展来进行面向方面建模的研究。有些是对特定方面语言 AspectJ 的 UML 扩展[2,9,10,21,22],有些通过扩展 UML 符号来表达面向方面的概念,把方面表达为类元并实现构造型表达横切关注、绑定、连接点等概念[8,12],或者从结构方面[11,23]和行为方面来建模方面[15~17]等等。这些方法都没有建立一个建模方面的框架,不能满足面向方面软件开发(AOSD)各个阶段的需求。因此,有必要构建一个具有较完整体系的方面建模框架,表达 AOSD 的语义,并实现从多方面多角度来描述 AOSD。这种框架与 UML 一样具有一系列的扩展机制不同的面向方面(AO)系统建模,并与 UML 语义相一致,既能表达 AOSD 又不会改变 UML 元模型。文[4,7]提出了一种用于面向方面建模的 UML 框架,但并不完善。本文以这种框架为基础,进一步完善和改进用于面向方面建模的 UML 框架。基于 UML 元模型,提出从基本框架到面向方面系统的结构建模

*)本文受国家自然科学基金(No. 60474072、No. 60174050)、广东省自然科学基金(No. 04009465、No. 010059)、广东省高校自然科学研究项目(No. Z03024)基金资助。刘瑞成　硕士研究生,主要研究方向:实时系统、软件设计方法与实时软件设计方法。张立臣　博士,教授,硕士生导师,主要研究方向:并行处理、分布式处理、网络计算、实时系统等。

和行为建模以及方面的织入机制等整个建模体系。

本文第2节对面向方面作简要介绍;第3节描述了UML元模型及扩展机制;第4节实现面向方面建模框架,从结构模型、行为模型以及方面织入来探讨面向方面的系统建模;最后对本文进行了总结并给出进一步的研究方向。

2 面向方面简介

AOP技术中,方面是表达对系统关注抽象出来的一般特性。方面有功能性方面和非功能性方面。功能性方面有同步控制、通信管理、分布式管理、资源共享等,非功能性方面有实时性、安全性、记录日志、异常处理、性能优化等。在引入面向方面技术之前,方面代码贯穿于系统多个功能性部件之间,与其它代码搅混在一起,不利于系统跟踪和维护。

AspectJ[1]是面向方面Java编程的实现,在技术上比较成熟。因此,这里基于AspectJ简单介绍面向方面的一些概念,详情请参见文[2]。

• 连接点(join point):程序中激活advice被执行的触发点,例如对象接受方法调用点。

• 切点(pointcut):一系列连接点利用与或非操作("&&"、"||"和"!")组成切点。

• 通知(advice):关注的实现,指定函数在连接点执行的机制。通知分为"before"、"after"、"around"、"after throwing"和"after returning"。

• 方面(aspect):实现横切关注的模块化单元,由pointcut、advice和introduction等来定义,与面向对象编程的类相似,可以重用和继承。

• 引入(introduction):提供修改基类的特性,可以插入类成员(字段、方法、接口和构造器等)和关系(范化、实现等)到基类。

其中Advice、joinpoint、pointcut关系如图1所示。

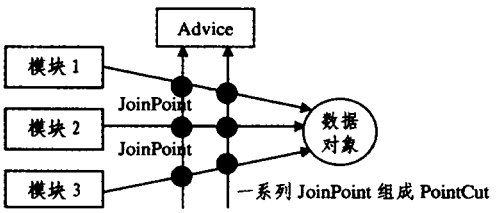

图1 AOP概念关系

AOP实现过程如图2所示。整个实现分为三个阶段,即方面分解、方面实现和方面织入。方面分解把横切关注点从系统需求中抽取出来形成方面。方面实现把各个分离出来的方面独立实现。方面织入把独立实现的方面集成到系统核心组件构成最终需求的系统。

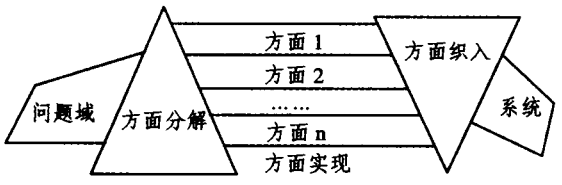

图2 AOP实现过程

3 统一建模语言UML的扩展机制

3.1 UML元模型体系结构

UML由图和元模型组成,元模型定义UML对象模型的语义,图形表达对象的结构和行为。UML元模型的概念框架是一个基于四层的体系结构[3,4],如图3所示。其中元元模型(meta-meta model)层构成了元建模体系结构的基础结构,主要责任是定义描述元模型的语言。元模型层组成UML基本元素,包括面向对象和面向组件的概念,主要责任是定义描述模型的语言,一般在这一层扩展UML来定义特定领域的UML建模框架(例如实时UML等),同样AOSD的UML框架也在元模型层定义。模型层组成UML的模型,主要责任是定义描述信息论域的语言。用户模型层是UML模型的实例,主要责任是描述一个特定的信息论域。

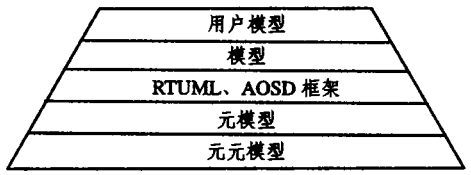

图3 UML体系结构

3.2 UML扩展机制

UML为标准的面向对象建模语言,特定领域建模提供了一系列的扩展机制,包括构造型、标记值和约束,它们提供增加新构造块、创建新特性和详述新语义的机制[5]:

• 构造型(stereotype) 可以为UML增加新事物和元模型,扩展UML的词汇,创建或者派生出问题域所特殊需要的新的构造块(例如≪aspect≫表示扩展的方面类)。新构造块具有与UML其它模型元素一样的行为和特性。

• 标记值(tagged value) 扩展UML元素的特性,允许在元素的说明中增加新的信息,为UML事物增加新特性。

• 约束(constraint) 扩展了UML元素的语义,允许增加新的规则或修改已存在的规则。一般利用OCL或自然语言来表达。

这3个扩展机制允许你修改并完善UML以满足你的需要。下面从UML元模型和框架扩展机制(Profile Extension Mechanism)实现对UML的扩展,来实现对面向方面的建模。

4 基于UML的面向方面的建模

4.1 面向方面系统建模的UML框架

UML表达方面的传统方法是通过UML图形把方面建模为类元,并构造类元来表达横切关注、组合或绑定操作[8~10,12]。这种方法不能满足所有AOSD建模的需要,可以通过构建AOSD框架来满足AOSD建模的需要,表达AOSD的语义,并从多方面多角度来描述AOSD系统。一个好的建模框架必须具备可扩展性来满足特定系统的需求,因此,需要建立一系列的扩展机制(构造型、标记值和约束)来支持AOSD建模,而且这种扩展机制必须与UML语义相一致,在不改变UML元模型的基础上能够表达AOSD。

AOSD框架的包结构与UML定义的包一样,提供层次的命名空间来定义组件,如图4所示。AOSD框架与UML元模型定义的核心包元素的关系如图5所示。UML元模型中AOSD框架扩展元素是通过定义构造型实现的。扩展机制主要也是基于构造型的概念,提供一种方法来区分元素,并且这些元素的行为与元模型实例一样。

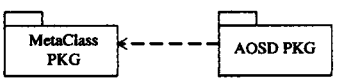

图4 AOSD框架包图

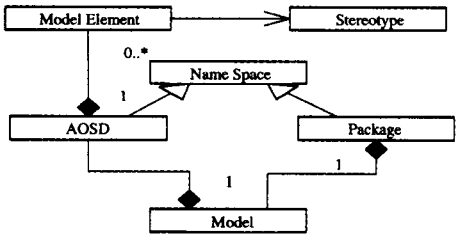

图 5　AOSD 框架与 UML 核心元素关系图

AOP[2] 提供了一系列机制把问题分解为功能性组件和方面组件，方面表达了横切核心功能性组件的关注。另外 AOP 也提供了模块化方面的方法，并使用织入机制把方面合并到核心功能性组件。AOSD 框架提供的构造型应当允许方面和核心类的模块化和组合。核心类被定义为 UML 核心模型元素，方面和相应的关联扩展这些模型元素。

AOSD 框架应该能够表达方面、核心功能性类、方面与核心类之间的关联以及方面之间和核心类之间的关联，并描述这些元素的结构和行为。

4.2　UML 表达面向方面概念

利用 UML 建模方面，是通过引入构造型《aspect》来表达方面的，《aspect》是基于类的构造型，确保《aspect》具有类实例相同的行为方式，《aspect》构造型用于建模横切关注单元。在 AOSD 框架中，类、类元与构造型《aspect》的关系如图 6 所示。

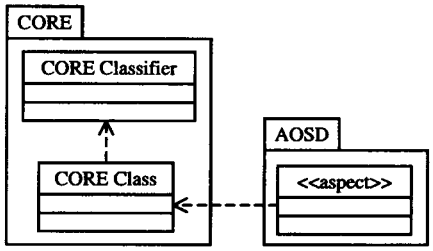

图 6　类、类元与《aspect》的关系

另外，我们还需要扩展 UML 构造型元素和标记值来表达面向方面设计涉及的概念，例如 pointcut、Advice、crosscut 等等。文[4,7～10]给出的表达面向方面概念都不完整，我们给出了较完整的方面概念 UML 表达方法，扩展的构造型如表 1 所示，扩展的标记值如表 2 所示。

表 1　扩展构造型规范

构造型	基本 UML 元模型	描述
《aspect》	Class	与核心类相对应，用于表达横切关注单元，必须与类或其它方面关联。方面通过关联与类通信，实现影响或增加核心类的功能行为。
《crosscut》	Association	用于表达方面横切核心功能性组件或其它方面的关系。
《advice》	Operation	表达方面类元特性，可以为 before、after、around、after returning 和 after throwing。
《pointcut》	Operation	表达方面与核心类或其它方面之间的连接点，一系列 joinpoint 组成一个 pointcut。
《joinpoint》	Operation	表达方面与核心类或其它方面之间的连接点。
《preactivation》	Operation	如果方面被标记为{synchronous}，则这个方面必须定义 preactivation 操作。
《postactivation》	Operation	如果方面被标记为{synchronous}，则这个方面必须定义 postactivation 操作。
《control》	Association	表达方面控制核心功能性组件的控制关系，也可表达类之间的控制关系，例同步控制。
《dominate》	Association	表达两个方面之间的关系。

表 2　扩展标记值规范

标记值	基本 UML 元模型	描述
{synchronous}	Association	表示方面影响核心类的功能，例同步方面、分布式方面。
{asynchronous}	Association	表示方面不影响核心类的功能，例日志方面、记录方面。
{abstract}	Class	表示方面(或横切点)为一个抽象方面或(抽象横切点)，与抽象类相似。
{active aspect}	Class	标记方面能够影响核心类行为。
{passive aspect}	Class	标记方面不会影响核心类行为。

4.3　结构模型

在文[11]中提出了一种 AOSD 的结构模型框架，根据需要我们简化了此模型，设计出一个更简单的框架如图 7 所示，由核心类、方面、横切接口、横切特性、核心特性以及横切关注构成。另外还定义了一个用于动态行为的横切关系。

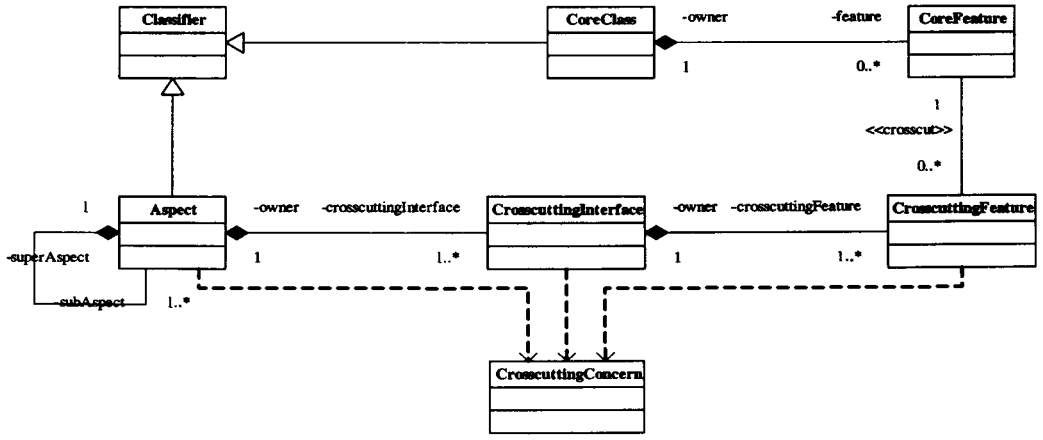

图 7　结构模型

• 核心类　描述系统核心部件，与切面元素相对应，表示系统核心功能性组件。一个核心类可以通过 Crosscut 关系

与多个横切元素相关联。

• **横切关注** 由各个关注方面、横切接口及横切特性组成，一个系统可以有多个横切关注，横切关注通过方面、横切接口及横切特性与核心类相关联。一个横切关注可以与系统中的多个类相关联。

• **方面** 用于实现横切关注，方面由一系列局部特性和横切接口组成。横切接口建模连接点，而横切特性定义这些连接点。在 UML 图中，方面表示为一个类元，描述通过织入机制(横切关系)结合到核心类的操作和属性(横切特性)。不同方面之间也具有一系列的相互作用。一个方面可以与系统中的多个类相关联。一个方面提供的服务可以分为同步和异步两种，用标记值{synchronous}和{asynchronous}表示。同步方面一般控制核心类行为，例如同步控制、调度、安全控制等功能性方面。异步方面一般为核心类提供额外的服务，不影响核心类的原有功能，例如日志记录、异常处理等方面。同时，根据需要一个方面可以把多个子方面组合为一个较大的方面[12]，以至于简化模型，易于表达方面系统。同样为了更好的模块化方面结构，也可以把多个方面的相同部分抽取形成一个新的子方面[13]。

• **横切接口** 从方面的观点来建模表达一系列横切对象的原型。横切接口建模接口层上静态的连接点，横切接口操作建模这些连接点上的横切操作。每个横切接口通过方面的结构和行为描述核心类所需求的最小特性集。一个横切接口指定如下特性[11]：1)方面引入到基类的属性和操作；2)方面期望使用的类方法的特征标记；3)精化类方法使用的方面操作。

横切接口把一个对象与方面相关的部分从对象中分离出来。如果不同的类实例都提供了同一个方面的横切接口操作，那么可把它们扩展为同一个方面。

• **横切特性** 是描述方面结合到一个或多个类的特性的模型元素。每个横切特性都具有定义扩展类型(增加、精化、重载)和横切类型(before、after、around)的属性，与 AspectJ 中 advice 定义的几种类型相对应。精化或重载基本操作特性具有关联的相互作用模式(refine-before、refine-after、refine-around、override)，相互作用模式由行为建模构成。

• **横切关系** 描述方面与核心类之间的关联关系。通过横切关联将方面的结构和行为绑定到核心类中，即方面织入到核心组件中，如图 8 所示。

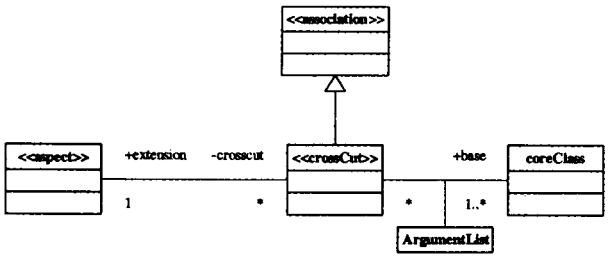

图 8 横切关系

在元模型层，横切关系是从方面到多个核心模块的关联，实际上是横切特性到核心特性的直接关联，并通过实参来表示方面的结构和行为要绑定到核心类的连接点。

有限缓冲区的例子通过引入同步方面和调度方面之后，类图主要由核心功能性类 BoundedBuffer、同步方面 SynchronizationAspect 和调度方面 SchedulingAspect 构成，如图 9 所示。为了防止死锁的出现，同步方面必须在调度方面之前进行处理，形成两个方面之间的时序关系。

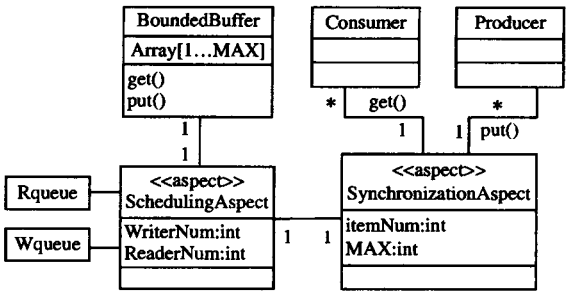

图 9 有限缓冲区方面建模的类图

4.4 行为模型

UML 元模型行为包如图 10 所示[14]，主要由 4 个子包组成：协作、用例、状态机和公共行为包。UML 提供了一系列建模系统静态和动态行为的视图，其中用例图、协作图、顺序图表示系统对象之间的动态行为，把系统动态方面看作是对系统变化部分的表示，而状态图是表示系统对象内部的行为关系。其中顺序图和协作图以及状态图和活动图在语义上是等价的，只是强调系统建模的不同方面。下面我们利用协作图和状态图来描述 AOSD 的行为建模。

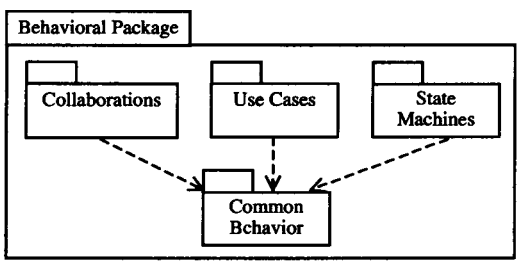

图 10 UML 行为包

• **协作图** 是强调收发消息的对象的结构组织的交互图。协作图显示了一组对象、这组对象间的链以及这组对象收发的消息，定义了一系列实例与链所扮演的角色。协作图表现了类元角色和关联角色，类元角色是协作需求的系列特性。核心类的类元角色实现系统需求的核心特性，而方面的类元角色提供核心类所需求的服务。关联角色是协作中关联两端所扮演的特定角色。关联角色通过组合规则实现，能够与构造型《Preactivation》和《Postactivation》相关联。

并发系统中，一个类的许多方法可能可以并发执行，就会与同步约束相关联，开始时功能性组件将会请求建立方面实例来获取核心类方法相关联的同步约束，方面实例实现横切接口。有限缓冲区例子的协作图如图 11 所示。

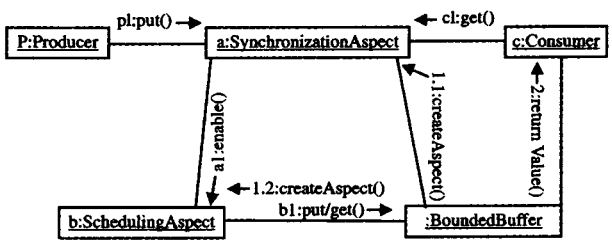

图 11 有限缓冲区协作图

• **状态图** 显示了一个由状态、转换、事件和活动组成的

状态机，强调一个对象按事件次序发生的行为。状态图中，当条件满足时事件被触发从一个状态迁移到另一个状态，迁移的结果就是动作的发生。状态图模型通过抽象迁移条件的硬编码来建模 AOSD 系统。利用状态图能够描述方面的横切行为，并实现横切关注隐式的织入到系统核心功能模块。状态图细化了模型，并能通过 CASE 工具支持实现自动产生代码。

状态图描述方面和对象行为的优点[4,15]：1）提供丰富的语义表达横切行为；2）假定系统是一个有限的状态系列，降低系统的复杂性；3）状态图模型具备完整的行为规范，保留了设计和实现之间迁移，使自动产生代码成为可能。

状态图在面向对象建模时，能够很好地建模对象内部行为，但是并不直接支持面向方面对象的行为建模，因此，必须在不改变状态图基本原理的情况下，对状态图实行扩展，增加表达方面的机制来建模方面行为，从而实现状态图支持面向方面建模的目的。

状态图中，状态的迁移是在事件、方法调用或计时器时间的终结触发实现的。状态图的方面建模应该考虑方面与方法、迁移的关联，而不是状态。为了清楚地表示每个对象的状态图以及方面的织入过程，可以利用层次状态图表示。每个区域之间的交互是通过共享变量、其它区域的状态变化或者消息传递机制（广播、传播事件）等来实现的[15]。

状态图应该能够允许核心和方面状态图并发、独立的开发，只有在特定应用需要时才进行绑定织入。

例如在有限缓冲区例子中，假设有一个核心对象和两个方面，两个方面分别处理同步和调度问题。那么状态图如图 12 所示，分为三个区域分别表示缓冲区、同步方面和调度方面的状态图。

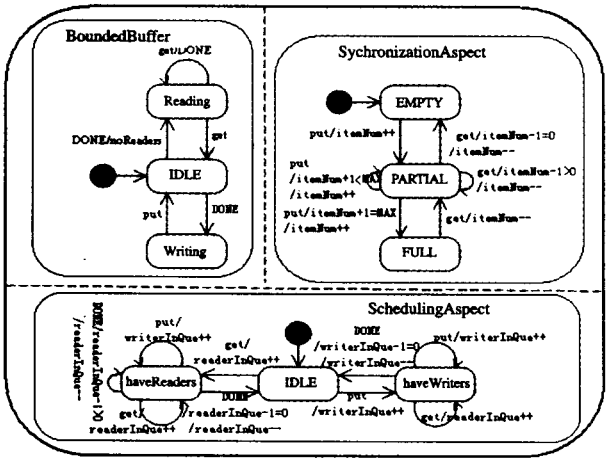

图 12　并发缓冲区与同步、调度方面状态图

4.5　方面织入

文[15]使用的织入方法是利用在状态图中的消息传递机制，通过硬编码的方式实现方面的隐式织入。但这种方法是开发人员直接实现事件传播，需要开发人员对事件传播的硬编码，指定事件传播的顺序。核心组件和方面的状态图是紧耦合关系，缺乏透明性。核心组件开发人员与方面开发人员需要知道对方的状态迁移和事件，不利于方面的重用。

文[16]扩展了文[15]方法，建立状态图框架，允许状态图到代码的转换，多个状态图对象可以连接建立区域以及允许横切状态图被织入到其它的状态图。通过利用形式化方法声明每个状态图的事件的处理方法，而不是对事件传播的硬编码，来实现方面的隐式织入。从更高层次进行抽象，更容易开发具有横切关注的复杂系统，同时也提高了方面的可重用性和灵活性。

以上面有限缓冲区例子来说明织入过程，有限缓冲区的状态图如图 12 所示。缓冲区、同步方面与调度方面的状态都是由同样的事件（put、get）控制。方面的织入是通过定义状态被触发迁移的规则来实现的。有限缓冲区例子中，迁移规则是通过定义缓冲区、同步方面和调度方面三者所处不同的状态下，什么情况才允许各自的一个状态迁移到下一个状态。为了简单明了起见，这里我们假设缓冲区处于 IDLE 状态，并产生 put 操作请求。

定义可操作规则如下：

1. 当同步方面处于 EMPTY 状态，且调度方面处于 IDLE 状态时，则同步方面迁移到 PARTIAL 状态，变量 itemNum 加 1，调度方面允许状态从 IDLE 迁移到 haveWriters，并进行 put 操作处理，缓冲区状态迁移到 Writing 状态，并允许方面状态图先于核心状态图处理事件。

2. 当同步方面处于 PARTIAL 状态，满足 itemNum＋1＜MAX，且调度方面处于 IDLE 状态时，则同步方面迁移到 PARTIAL 状态，变量 itemNum 加 1，调度方面允许迁移到 haveWriters 状态，并进行 put 操作处理，缓冲区状态迁移到 Writing 状态。并允许方面状态图先于核心状态图处理事件。

3. 当同步方面处于 PARTIAL 状态，满足 itemNum＋1＝MAX，且调度方面处于 IDLE 状态时，则同步方面迁移到 FULL 状态，变量 itemNum 加 1，调度方面允许迁移到 haveWriters 状态，并进行 put 操作处理，缓冲区状态迁移到 Writing 状态。并允许方面状态图先于核心状态图处理事件。

满足上述规则的情况就可以进行状态迁移和事件处理，否则进行错误处理。有限缓冲区例子织入方面的状态图如图 13 所示。

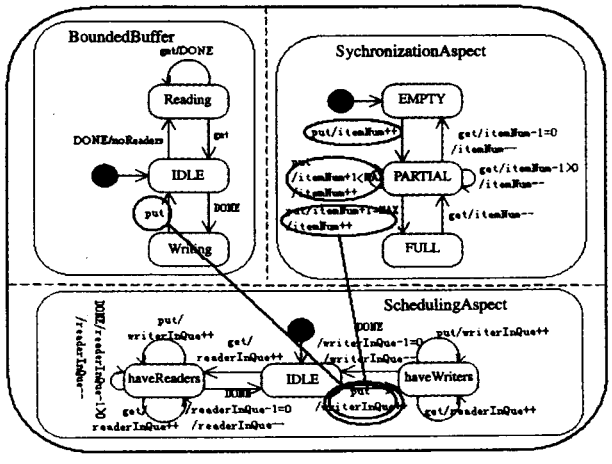

图 13　织入方面状态图

为了能进一步实现代码的自动产生，还可以提供简单清晰的接口来指定织入关系和事件的重解释（reinterpretation），文[16]提出的方法并不能完全实现方面与方面之间的织入关系，也不支持一个连接点同时织入多个方面，因此我们扩展了文[16]的方法，解决了上述的问题。另外，标记为{synchronous}的方面和标记为{asynchronous}的方面实现不同处理方式。上面的织入过程可以通过使用基本状态图对象和方面状态对象的引用来指定：

AspectID＝core. crosscutBy(SchedulingAspect)；

AspectID = SchedulingAspect. crosscutBy (SynchronizationAspect);

上面就指定了核心组件和方面以及方面之间的织入关系,其中 AspectID 是为了区分织入的不同方面。在织入过程中方法将会被调用来映射核心和方面状态图中的事件,而且这些方法保留了事件重解释(reinterpretation)所定义的规则。具体方面织入时只需要指定方面的具体信息,上述定义的规则等价于下面的表示:

1) reinterpretEvent (SchedulingAspect," IDLE"," put"," EMPTY","put/itemNum++",AspectID,Statechart. PREHANDLE); reinterpretEvent(core,"IDLE","put","IDLE","put/writerNum++", AspectID,Statechart. PREHANDLE);

2) reinterpretEvent(SchedulingAspect," IDLE"," put"," PARTIAL","put/itemNum+1<MAX/itemNum++",AspectID,Statechart. PREHANDLE); reinterpretEvent (core," IDLE"," put"," IDLE","put/writerNum++",AspectID,Statechart. PREHANDLE);

3) reinterpretEvent (SchedulingAspect," IDLE"," put"," PARTIAL","put/itemNum+1=MAX/itemNum++",AspectID,Statechart. PREHANDLE); reinterpretEvent (core," IDLE"," put"," IDLE","put/writerNum++",AspectID,Statechart. PREHANDLE);

这种织入方式中,每个状态图之间不必指定其它的状态图的具体细节,方面状态图可以被应用于任何需要的情况下。由于没有严格限制状态图之间的横切关系,一个方面状态图可以横切核心状态图或者其它方面状态图。另外还建立了特定事件迁移之间的可确定性关系,确保方面状态图具有改变事件或事件数据的能力,并在核心状态图之前或之后进行事件迁移。这与 AspectJ 定义的 before 和 after 通知相似,文[9,10]详细介绍 AspectJ 的织入过程如图 14 所示,但 AspectJ 的方面织入是由编译器实现的。核心状态图与方面状态图之间的动作交互是通过事件或事件数据实现的。

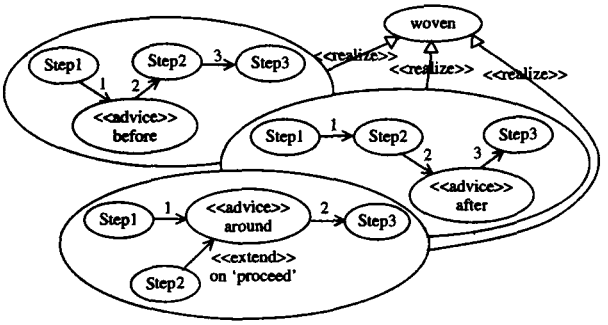

图 14 AspectJ 中 Advice 的织入过程

如果是利用形式化方法来定义状态图迁移规则的,就可以实现迁移正确性的验证,同时也适合各种面向方面系统的建模。文[17]实现了利用 CASE 工具实现代码自动产生,文[18]提出了一种从方面模型到 AspectJ 代码的转换方法,实现方面的 AspectJ 代码自动产生。我们通过形式化说明从更高层次实现方面的织入,目的就是为了实现方面建模方法的更一般化,并能更好与 CASE 工具结合,最终实现代码自动产生。

结论 面向方面技术作为一种能与面向对象技术相结合的新兴技术,通过对横切关注建模来表达系统。面向方面的横向设计与面向对象的纵向软件设计的结合,形成了一种二维正交的软件设计方法。横切关注是复杂的,方面建模能够简化模型的复杂性。

目前,面向方面编程技术并不成熟,还处于研究阶段,没有形成统一的标准和建模方法,通过对面向对象编程技术的分析,发现面向方面编程应该扩展到整个软件开发周期,从设计到代码产生都充分利用 AOP 技术,这需要一种统一的建模框架来建模方面系统。UML 作为当前较流行和成熟的面向对象建模语言,可通过扩展 UML 建模面向方面,既提高了 UML 的建模能力,又实现了 AOP 与 OOP 技术的结合,提高了软件设计能力。促使设计的软件更具有模块化,提高代码的可重用性、可维护性。

目前已有一些通过扩展 UML 来进行面向方面的建模的研究,但都不够完善。本文在已有的基础上进行扩充,在元模型层次上实现一个较完整的面向方面的建模框架。从结构建模、行为建模、方面织入以及代码产生几个部分阐述了面向方面的建模方法。结构建模利用 UML 类图实现面向方面系统的结构模型,方面与核心组件及方面之间的静态模型关系。利用 UML 协作图表达方面与核心组件及方面之间的动态行为,通过状态图细化方面和核心组件的动态行为,并实现状态图的方面与核心组件及方面之间的织入关系,最终实现方面代码的自动生成。

面向方面建模技术的研究才刚刚开始,还有许多问题需要进一步的探讨和研究,例如:

1)进一步实现 UML 与面向方面的结合,扩展 UML 符号来描述面向方面的概念。2)利用形式化方面来规范框架及织入方面的验证;3)面向方面的语义问题;4)进行 CASE 工具的结合,实现方面代码的自动产生;5)把面向方面建模方法应用到一些大的分布式、异构的系统开发中。

面向方面与面向对象技术的结合是今后软件分析与设计的发展方向,最终实现两者的无缝结合,并进一步结合形式化方法,充分利用形式化方法的表达和验证能力从而在更抽象层次上实现方面建模及正确性验证。

参 考 文 献

1 http://eclipse.org/aspectj/

2 Kiczales G, Hilsdale E, Hugunin J, et al. An Overview of AspectJ. In: Proc. of 15th ECOOP, 2001

3 OMG Document ad/97-08-03. UML Syntax and Semantics guide. http://www.rational.com/uml

4 Aldawud O, Elrad T Bader A. UML profile for aspect-oriented software development. In: position paper for the Third international workshop on Aspect oriented modeling, 2003

5 Booch G, 等编. 邵维忠, 等译. UML 用户指南. 北京:机械工业出版社, 2001

6 Kiczales G, et al. Aspect-Oriented Programming. In: Proc. of the 11th European Conf. on Object-Oriented Programming, June 1997

7 Aldawud O, Elrad T, Bader A. A UML Profile for Aspect Oriented Modeling. Workshop on AOP, OOPSLA 2001

8 Zakaria A A, Hosny H, Zeid A. A UML Extension for Modeling Aspect-Oriented Systems. In: Intl. Workshop on Aspect-Oriented Modeling with UML, Germany, Sept. 2002

9 Stein D. An Aspect-Oriented Design Model Based on AspectJ and UML: [Master Thesis for the Master Degree in Management Information Systems]. The Department of Business Arts, Economics, and Management Information Systems University of Essen, Germany, 2002

10 Stein D, Hanenberg S, Unland R. A UML-based Aspect-Oriented Design Notation for AspectJ. In: Proc. of the 1st Intl. Conf. on Aspect-Oriented Software Development, April 2002

(下转第 213 页)

表 2 嵌入式 CPU 特性的权重值

嵌入式 CPU 名称	ARM920T	ARM7EJ-S	StrongARM SA-1110	Geode™ GX1-333	TMS320C6713
价格	1200.00	850.00	600.00	600.00	680.00
工作频率	250	133	206	330	300
体积	3.5mm²	3.0mm²	3.5 mm²	3.0mm²	3.5mm²
容量	—	—	—	—	—
工作电压	1.62V	1.08V	1.75V	2.2V	1.4V
汉化	0	0	0	0	0
价格 (权重)	1.527	1.081	0.763	0.763	0.865
工作频率(权重)	−1.025	−0.546	−0.845	−1.354	−1.231
体积 (权重)	1.061	0.909	1.061	0.909	1.061
容量 (权重)	0	0	0	0	0
工作电压(权重)	1.006	0.671	1.087	1.366	0.870
汉化 (权重)	0	0	0	0	0

则由式(4),可得到 CPU 的权重矩阵:

$$H^1=\begin{bmatrix} 1.527 & 1.081 & 0.763 & 0.763 & 0.865 \\ -1.025 & -0.546 & -0.845 & -1.354 & -1.231 \\ 1.061 & 0.909 & 1.061 & 0.909 & 1.061 \\ 0 & 0 & 0 & 0 & 0 \\ 1.006 & 0.671 & 1.087 & 1.366 & 0.870 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{11}$$

同理,由式(4),(5)可分别得到组成元素 FLASH 存储器、操作系统和嵌入式数据库应用软件的权重矩阵 H^2,S^1,S^2 如下:

$$H^2=\begin{bmatrix} 1.011 & 0.647 & 0.539 & 0.916 & 1.887 \\ 0 & 0 & 0 & 0 & 0 \\ 2.577 & 0.638 & 0.542 & 0.814 & 0.429 \\ -1.265 & -0.316 & -0.316 & -0.632 & -2.470 \\ 1.1 & 0.9 & 1 & 0.9 & 1.1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{12}$$

$$S^1=\begin{bmatrix} 1.786 & 0.089 & 0.074 & 2.976 & 0.074 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 2.239 & 0.784 & 0.672 & 1.209 & 0.096 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1.667 & 1.667 & 1.667 & 0 \end{bmatrix} \tag{13}$$

$$S^2=\begin{bmatrix} 4.808 & 0 & 0 & 0 & 0.192 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.886 & 0.237 & 1.499 & 1.425 & 0.954 \\ 0 & 0 & 0 & 0 & 0 \\ 1.25 & 1.25 & 1.25 & 1.25 & 0 \end{bmatrix} \tag{14}$$

由式(10)得

$F(H^1, H^2, S^1, S^2) = \min\limits_{1\le j\le 5}\{16.961, 8.011, 8.993, 12.209, 3.827\} = 3.827$ 说明选择第五列硬、软件组成元素作为系统的协同设计的方案,可以达到优化的效果。

小结 随着嵌入式系统复杂性的不断提高,当前热门的硬/软件协同设计方法已显现出优势。本文采用系统组件的权重值为参考,组成元素划分为依据的设计理念,构建了移动嵌入式系统设计的 EHSC 方法,试图寻找移动嵌入式系统设计中硬/软件协同设计的最佳搭配方案。并依照此模型,完成一种移动嵌入式系统"电子书包"[6]阅读器的设计和实现。由于不同移动嵌入式系统对成本的要求以及各种限制条件,在设计复杂移动嵌入式系统时,可以根据计算的复杂性和具体的限制条件,将应用系统按照不同领域分类,以便用科学和系统的方法而不是靠经验来选择目标结构。因此在这个研究领域还有许多研究工作要做。

参 考 文 献

1 Chou P, Ortega R B, Borriello G. Interface Co-Synthesis Techniques for Embedded Systems. In: Proc. of the IEEE/ACM International Conference on Computer-Aided Design. San Jose, CA, Nov. 1995

2 Athanas P, Silverman H F. Processor Reconfiguration Through Instruction-Set Metamorphosis. Computer, 1993, 26(3)

3 Kumarjames S, et al. THE CODESIGN OF EMBEDDED SYSTEMS. Kluwer Academic Publishers, 1996

4 Thomas F, Nayak M M, Udupa S, Kishore J K, Agrawal V K. A hardware/software codesign for improved data acquisition in a processor based embedded system, Microprocessors and Microsystems, 2000

5 郭晓东,刘积仁. 一种基于遗传算法的硬件/软件划分方法. 计算机辅助设计与图形学学报,2000(4)

6 郑世珏,张江陵. 基于嵌入式系统模式的电子课本. 计算机工程,30(4):193～195

(上接第 209 页)

11 Chavez C, Lucena C. A Metamodel for Aspect-Oriented Modeling. Workshop on Aspect-Oriented Modeling with the UML. In: The First Conf. on Aspect-Oriented Software Development (AOSD 2002), the Netherlands, April 2002

12 Kandé, Kienzle, Strohmeier. From AOP to UML: Towards na Aspect-Oriented Architetural Modeling Approach. Agosto, 2002

13 Katara M, Katz S. Architectural views of aspects. In: Proc. of the 2nd Intl. Conf. on Aspect-oriented software development, Boston, Massachusetts, March 2003

14 Rumbaugh J, 等编. 姚淑珍, 唐发根, 等译. UML 参考手册. 北京: 机械工业出版社, 2001

15 Aldawud O, Bader A, Ellrad T. Weaving with Statecharts. Aspect-Oriented Modeling with UML workshop at the 1st International Conference on Aspect-Oriented Software Development, April 2002

16 Mahoney M, Bader A, Elrad T, et al. Using Aspects to Abstract and Modularize Statecharts. In: The 5th Aspect-Oriented Modeling Workshop in Conjunction with UML 2004 Lisbon, Portugal, Oct. 2004

17 Aldawud O, Bader A, Constantinides C, et al. Modeling Intra-Object Aspectual Behavior. In: Proc. of the 1st ICSE Workshop on Describing Software Architecture with UML, Toronto, Canada, May 15, 2001

18 Groher I, Schulze S. Generating Aspect Code from UML Models. The 4th AOSD Modeling With UML Workshop, 2003

19 Selic B, Rumbaugh J. Using UML for Modeling Complex Real-Time Systems. ObjecTime Limited/Rational Software whitepaper, 1998

20 OMG Document ad/02-04-01. UML Profile for CORBA Specification. http://www.omg.org

21 Stein D, Hanenberg St, Unland R. Designing Aspect-Oriented Crosscutting in UML. In: AOSD-UML Workshop at AOSD'02, Enschede, The Netherlands, Apr. 2002

22 Han Y, Kniesel G, Cremers A B. A Meta Model and Modeling Notation for AspectJ. http://www.cs.iit.edu

23 Harrison W, Tarr P, Ossher H. A position on considerations in UML design of aspects. In: Position paper in Workshop on Aspect-Oriented Modelling with UML in conjunction with AOSD 2002, Enschede, The Netherlands, Apr. 2002