面向方面的体系结构描述语言 AC2-ADL

文 静 应 时 张琳琳 倪友聪
(武汉大学软件工程国家重点实验室 武汉 430072)

摘 要 体系结构描述语言(ADL)是基于体系结构的软件开发的基础,传统的 ADL 由于缺乏对混杂与分散在软件体系结构多个单元中的各种设计决策的描述能力,导致软件体系结构设计方案难以理解、难以演化和难以重用。通过设计面向方面的软件体系结构描述语言 AC2-ADL,使用方面组件明确地描述系统的横切关注点;并引入方面连接件以及抽象出软件体系结构语境中的注入点,呈现结构之间复杂的交互,以解决不同关注点的分散和交织等问题,试图为设计和描述面向方面的软件系统的软件体系结构提供一种有效的解决方案。研究结合电子商务领域的网上拍卖系统,讨论了该语言的主要应用过程,具有一定的参考作用。

关键词 面向方面的软件体系结构,软件体系结构描述语言,方面组件,方面连接件,软件体系结构层注入点
中图法分类号 TP311.5 **文献标识码** A

Aspect-oriented Architecture Description Language AC2-ADL

WEN Jing YING Shi ZHANG Lin-lin NI You-cong
(State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)

Abstract Architectural Description Language(ADL) is a foundation of the software development based on software architectures, the traditional ADLs lack the ability to describe the crosscutting concerns and crosscutting interactions in the software architecture, leading to the difficulties in comprehension, evolution and reusability of software architectural design decisions. We proposed a new aspect-oriented ADL AC2-ADL, aiming to offer an effective systematic solution for the representation of aspect-oriented software system, AC2-ADL provides aspectual components to describe the crosscutting concerns. In addition, by introducing Aspectual Connector and abstracting the joinpoint of the architecture, it described the complicated interactions among the software architecture elements. Summarily, the whole description process of AC2-ADL was discussed systematically through a case study in e-business domain.

Keywords AO architecture, AO architecture description language, Aspect component, Aspect connector, Architectural joinpoint

软件体系结构描述语言(Architecture Description Language)已经被认为是一个重要的工具,用于在软件开发的早期,系统地推理软件体系结构的总体设计方案。它在高层抽象上描述了构成软件系统的元素、以及元素之间的交互关系、指导元素组合的模式和有关的约束要求。在过去的几十年中已有大量的研究机构在软件体系结构描述语言的研究上做出了不少杰出的贡献,目前主流的 ADL 有 Aesop, MetaH, C2, Rapide, SADL, Unicon 和 Wright 等,这些 ADLs 强调了体系结构不同的侧面,对体系结构的研究和应用起到重要的作用[1]。在对软件体系结构的深入研究和广泛应用中,人们发现通过常规方法设计出的软件体系结构方案中始终存在着一些横切(crosscutting)的行为和横切的特征。这些横切的行为和特征横切了组成软件体系结构的组件和连接件。这种横切结构不但造成组件和连接件之间的紧耦合,而且还使得组件和连接件中的内容变得混杂,它们所代表的概念变得复杂,它们的边界也变得模糊起来。目前大部分的 ADLs 由于缺乏对这些横切行为和横切的特征描述,从而导致软件体系结构设计方案难以理解、难以演化和难以重用。

近年来,随着面向 Aspect 的程序设计(Aspect-Oriented Programming)[2]技术的成熟及进一步地深入发展和广泛地应用,随着在软件生命周期早期阶段研究和开发方面技术,即通常被人们称为早期方面(early aspect)技术[3],逐渐引起人们的注意,将 AOP 中的方面及其相关概念和技术,扩展到软件体系结构设计阶段就成为一个非常有前景并值得高度关注的研究方向。在软件体系结构设计阶段中,在软件体系结构抽象层次上,研究和应用面向方面的 ADL 具有重要的意义。首先,利用面向方面的设计方法,设计出的软件体系结构可以解决软件体系结构设计方案中各种设计决策、行为和特征混

到稿日期:2008-01-16 返修日期:2009-03-03 本文受国家高技术研究发展计划 863 项目(2006AA01Z168),国家自然科学基金项目(60773006)资助。

文 静(1982—),女,博士生,主要研究方向为体系结构、面向方面软件开发;应 时(1965—),男,博士,教授,博士生导师,主要研究方向为面向对象软件工程方法、基于组件的软件工程方法、软件体系结构和模式、软件的可重用性与互操作性等;张琳琳(1974—),女,博士生,讲师,主要研究方向为体系结构、面向方面软件开发。

杂与分散在软件体系结构层多个组件和连接件中的问题，使这些横切的内容可以单独地被模块化，使软件体系结构关注点得到直接和明确的体现和跟踪，可以提高软件体系结构的可理解性和可维护性，有利于软件体系结构的演化和重用。其次，利用面向方面的设计方法设计出的软件体系结构，可以为后续阶段使用 AOP 实现技术提供获取和标识代码层方面的线索，有利于 AOP 编码技术的高效应用和 AOP 编码阶段与软件体系结构设计阶段更顺利地衔接。

本文通过设计一种新的面向方面的软件体系结构描述语言 AC2-ADL，明确地描述系统的横切关注点，弥补传统的 ADL 在横切关注点描述方面的不足；对体系结构层的横切关系进行建模，明确地描述结构之间复杂的交互；并定义了体系结构层的注入点，既为面向方面软件体系结构描述语言的编织机制和协作机制提供赖以工作的信息基础，也为开发人员理解集成了各种必要的软件体系结构方面、组件和连接件而得到的最终设计结果的信息基础。通过结合网上拍卖的实例展示了使用该语言的主要步骤和结果。最后讨论了研究中的主要问题和进一步的工作。

1 相关研究工作

迄今为止，已经有大量的工作关注于面向方面的软件开发过程中的方方面面，比如体系结构模型，面向方面的组件，体系结构描述语言，以及与复杂系统相关的方面的动态配置和织入技术等等。这些工作成果为我们的理论提供了一个良好的理论与实践的平台，使得研究能够朝着一个正确的方向前进。由于本篇文章的重点是关于面向方面的体系结构建模和描述，因此对于相关工作的研究也定位于该领域。

目前，主流的设计方法大部分是基于 UML 的建模方法如 PCS 框架[4]，面向方面的产生式方法如 AOGA[5,9]，TranSAT 框架[6]和 DAOP-ADL[10]等，它们的主要目的是提供恰当的抽象概念和相应的可视化标记符号来捕获体系结构中的横切关注点，并使这些关注点单元化模块化。然而由于这些模块把与横切相关的信息都封装在方面组件中，比如横切接口、切入点、组合规则，以及绑定元素等，使得方面组件的设计模型过于臃肿，并且不利于方面组件的重用。在 AC2-ADL 的模型中，通过方面连接器把切入点、通知、横切类型等概念说明从方面中独立出来，使得方面组件只用关注于提供自身的横切功能，而不用关心横切规则。因此这些方面组件能够根据不同的方面连接器和基本组件组合起来，以描述不同关注点的需求，从而提高基本组件和方面组件的重用性。

在 Thaís Batista[11]等人提出的 AspectualACME 中，认为传统的组件已经足以描述体系结构中的各种计算单元，包括对象、元对象，层(Layers)甚至是方面(Aspects)，因而没有在 AspectualACME 中定义新的元素来表示方面；但为了弥补传统的连接器缺乏对描述体系结构中关注点之间的横切交互关系(Crosscutting Interaction) 的不足，因此，他们提出了一种新的模型—方面连接器(AspectualConnector)来建模和描述体系结构中的横切交互关系。这和我们在连接器方面的设计很类似，而在 Epoch 的连接器中提供了 introduce 关键字来表述横切接口对目标组件的结构上的影响，这在 AspectualACME 中是没有的，它只提供了对目标组件行为上的影响的表述，这在很大的程度上削弱了使用面向方面技术进行设计的能力。此外，由于 AspectualACME 没有提供专门的模块来描述横切关注点，这导致了横切关注点和非横切关注点在体系结构层表达的不明确，从而导致实现层缺乏可追踪的线索。

此外，一些特殊的方法也值得关注。如 Jennifer Pérez[8]等人提出的 PRISMA 是一种基于反射的面向方面和组件的体系结构模型。在 PRISMA 模型中，元级可以包括多个不同的方面如：功能性方面、协调方面、分布方面、质量方面、上下文可知性方面(Context-Awareness Aspect)、演化方面等。这些方面可以根据不同的需求组合起来，并根据相应的基级中的元素(组件、连接器以及系统等)被具体化。而方面的执行则是通过组件和连接器所包含的织入的规格说明来定义。模型 PRISMA 使得方面的集成和每一个方面的实际代码的产生可以相互独立。然而，在越来越复杂的软件系统中，只在元级设计方面是不够的，一些方面也需要在基级的设计元素中出现，它们不仅可以封装那些分散和混在不同模块中的元素，还能提高设计的灵活性。

在这些研究工作的基础上，给出了一种面向方面的体系结构设计方法，并定义了面向方面的描述语言 AC2-ADL 作为软件体系结构设计工具，用以描述系统面向方面的软件体系结构。

2 AC2-ADL

本段首先介绍了面向方面的软件体系结构设计方法的基本思想，并在此基础之上，给出面向方面的体系结构描述语言 AC2-ADL 的概念框架以及主要的语法元素。

2.1 基本思想

体系结构中的关注点是一块用户所感兴趣的领域，也可是某个问题的解决方案的一部分，甚至是该体系结构需要完成的一个通用性的功能或是某个质量属性[12]。比较典型的例子是体系结构层的一些非功能属性：如安全性、容错性、持久性和同步性等。这些关注点的行为往往趋向于和其他的关注点混杂在一起，而导致它们难以推理和实现；或是分散在系统的不同模块中，为了完成全局的目标而破坏了模块的封装。因而被称之为横切关注点。

常规的软件体系结构设计方法几乎都是首先以业务过程和用例脚本为主要关注点，设计出软件体系结构的基本结构。然后，在软件体系结构中以直接横切构件和连接件的形式，将满足非功能需求的设计方案加入到基本结构中去。由于没有简单规范地考虑如何满足非功能需求的设计步骤，使得设计过程变得复杂起来，此外，由于非规范地加入满足非功能需求的设计方案，模糊了组件和连接件的角色与边界，使得设计结果也变得难以理解。如何使用适当的体系结构元素描述和建模这些不同性质的关注点，以及如何提供有效的组合方式将这些体系结构元素组装成最终的系统软件体系结构是研究体系结构描述语言的重点。

我们的方法基于 AOSD 以及 EA 的思想，在软件的体系结构层引入一种新的组件类型——方面组件为横切关注点的行为进行建模，同时保留传统的组件概念建模系统的功能性关注点，并使用一种新的连接件类型——方面连接件以及显式的表达软件体系结构语境中的注入点来呈现结构之间复杂的交互，以解决不同关注点的分散和交织等问题。最后，通过体系结构的配置/拓扑显示的描述体系结构层中，各元素实例

的组合关系。

2.2 概念框架以及基本语法

为了刻画面向方面的体系结构，AC2-ADL 除了传统的 ADL 中所包含的概念元素以外，还引入方面组件(Aspectual Component AC)和方面连接件(Aspectual Connector AC)作为体系结构层的第一类元素。然而，一个好的 ADL 不仅要有较好的抽象表达能力，还应能够在体系结构设计阶段对有关性质进行分析和推理。因此，AC2-ADL 定义了相应的语法元素类型，并结合一种序逻辑语言 XYZ/E[13]作为各种组件的行为进行逐步求精，以及相关性质的分析的基础。下面分别以 BNF 的形式[1] 给出了 AC2-ADL 的部分类型系统，以及其中主要元素的相关语法。

类型系统：类型系统的定义有利于软件体系结构正确性的验证，以及软件体系结构中概念元素的替换，如同类型的组件实例可以代替体系结构配置中该类型的实例。图 1 展示了 AC2-ADL 的类型系统由两部分组成：基本类型和抽象类型，其中基本类型又包括简单类型和复杂类型，如简单类型 Any，表示该类型的变量可以是任意一种简单类型；而复杂类型 location[Type]则表示对值的存和取，通过该类型，一个值可以存放在多个不同位置中，系统通过这些位置的别名，可以取出该值。而抽象类型涵盖了 AC2-ADL 的基本概念框架，如组件，连接件，方面组件，方面连接件，体系结构配置等主要元素。这里的组件和连接件与传统的 ADL 在概念上没有很大的区别，因此，本文重点介绍该 ADL 中两种特殊的元素：方面组件和方面连接件，以及从传统的体系结构配置扩展而来的面向方面的体系结构配置规范。

方面组件：与传统组件不同，方面组件作为一种新的元素，在属性和功能上有了较大的改变。为了体现自身的横切特征，方面组件定义了一种特殊的接口类型，称之为横切接口(crosscutting Interfaces)[7](如图 1 所示)，横切接口定义了方面组件能够提供的计算委托及其用途上的约束。通过不同的横切接口，一个方面组件可以对不同的组件进行行为和结构上的影响。

Abstract syntax of types

Type::= baseType | abstractType
baseType::= simpleType | complexType
simpleType::= **Any** | **Natural** | **Integer** | **Real** | **Boolean**| **String**
complexType::= **tuple** [Type, …, Type] | **location** [Type] | sequence[**Type**] | **set**[Type] | **bag** [Type]
abstractType::= component | aspectComponent | aspectConnector| connector|architecture |interface |role
interface::= providedInterfaces | requiredInterfaces | crosscuttingInterfaces
role::= baseRoles | crosscuttingRoles

图 1 AC2-ADL 的部分类型系统

此外，根据系统的复杂性，方面组件可能会包含两个甚至两个以上不同类型的横切接口来提供不同的横切功能作用于不同的实体。例如在一个使用面向方面设计模型方法对观察者模式进行设计的例子中[7]，观察(Observation)方面组件为观察者模式中的两个参与者——主题(Subject)和观察者(Observer)提供了两个横切接口。这两个横切接口一个横切了主题对象，另一个则横切了观察者对象，并分别为这两种不同类型的对象引入了不同的属性或功能。把这种方面称为异构方面(Heterogeneous Aspect)。通过利用复合方面组件可以解决在目前的面向方面的设计中出现的异构方面的描述问题。一个复合方面组件可由多个不同的功能单一的方面组件组合而成，因此可以含有相应的子体系结构(sub-Architecture)，通过映射声明(mapping Declaration)，外部的横切接口可以委托子体系结构中元素提供的相应横切服务。根据这些思想，图 2 定义了方面组件的基本语法。

aspectComponent::=
aspectComponent component_name **is** {
  parameters {parameters name **is** simpleType;}
  methods {methods}
  crosscuttingInterface {crosscuttingInterfaces}
  [**subArchtecture** subArchtecture_name **is**{ architeture|**null**;}]
  [**mappingDeclaration** {mapping_expression}| **null**;]
  [**internalProcess is** {{conditionalelement_expression;}| **null**;}]
  [**constrains is** {{constrains_expression;}| **null**;}]
}

图 2 方面组件的基本语法

如图 2 所示，方面组件的语法规范由横切接口、数据类型定义和方法、方面组件内部行为的描述以及子体系结构(如果有)等元素组成。其中，组件内部行为以及方法中的行为都由时序逻辑语言 XYZ/E 中的条件元表达式(conditional element expression)描述。这里需要对时序逻辑语言 XYZ/E 做一个简要的介绍，XYZ/E 中的条件元有两种形式：

$$LB=y\wedge R\Rightarrow \$Ov=e\wedge \$OLB=z \quad (1)$$

$$LB=y\wedge R\Rightarrow @(Q\wedge \$OLB=z) \quad (2)$$

形式(1)所示条件元直接定义了程序相邻状态之间的转换关系；形式(2)所示条件元表示程序抽象规范，其中符号@可以是下一时刻算子$O或最终时刻算子◇；两种形式的条件元都是时序逻辑公式，其语义即为此时序逻辑式的语义模型。此外，XYZ/E 也提供输入命令和输出命令：

$LB=y\wedge R\Rightarrow ChNm?\ x\wedge \$OLB=z$;

$LB=y\wedge R\Rightarrow ChNm!\ e\wedge \$OLB=z$,

其中，“*ChNm*”是通道名，x 是变量，*e* 是表达式。“*ChNm*? *x*”表示通过通道 *ChNm* 将数据送入变量 *x*；“*ChNm*! *e*”表示将 *e* 的值经过通道 *ChNm* 送出。在 AC2-ADL 中，接口和接口中的方法都可以作为 XYZ/E 中的通道。有关 XYZ/E 具体的语法可参看文献[13]。

接口是各种组件与外部世界的一组交互点。与面向对象方法中类的说明相同，ADL 中的组件接口说明组件提供那些服务(消息、操作、变量等)。在 AC2-ADL 的类型系统中，接口有 3 种类型(如图 1 所示)：提供接口(providedInterfaces)、请求接口(requiredInterfaces)和横切接口(crosscutting Interfaces)。每种接口可以包含一组操作(Operation)，操作对应组件中定义的方法，接口通过这些操作和外部环境交互，如提供或定制的所需要的服务。因此，操作中的方法需要支持消息交互的场景，并注明其消息的传输的方向。图 3 描述了 AC2-ADL 中 3 种接口类型的定义。

[1] BNF 中“::=”表示定义；“|”表示“or”；“[…]”表示可选择的，而“{…}”表示可重复的，黑体符号表示终结符。

```
crosscuttingInterfaces::=
  crosscuttingInterface_name is {
    [ add_Operation method | {method;}|null ] |
    [ replace_Operation { method | {method;} | null ] |
    [ introduce_Operation { [parameters_name | methods| provided-
Interfaces | requiredInterfaces]} | null]
  }
requiredinterfaces::=
  requiredInterface_name is {
    [Operation method | {method;} | null ] }
providedInterfaces::=
  providedInterface_name is {
    [Operation method | {method;}|null ] }
method::=method_name is
[direction(parameters_name) |{ parameters_name;} | null]
{{conditionalelement_expression;}| null;}
direction::= in | out | in|out
```

图 3　AC2-ADL 中接口及方法的语法定义

值得注意的是，横切接口中的操作与传统的提供接口和请求接口不同，它被分为 3 种不同的类型，分别被表示为 add_Operation，replace_Operation 和 introduce_Operation。其语义可解释为：通过 add_Operation 操作，横切接口可以把该操作中的方法添加到目标组件中，以增强目标组件的行为；通过 replace_Operation 操作，横切接口可以使用该操作中的方法代替目标构建中的方法，从而改变目标构建的行为，这两种操作对应着方面组件的动态横切特性，即一个方面组件可以影响组件的行为；而 introduce_Operation 操作体现了方面组件的静态横切的特性，通过该操作方面组件可以为目标构建引入相关的属性，方法，甚至接口，从而影响目标构建的结构。总之，这些操作体现了面向方面技术所特有的特征，使得软件架构师能以一种本质上更优雅的方式，插入跨越整个系统的公共行为。

方面连接件：在复杂的软件系统中，为了满足多种的属性，不同的关注点中某些元素可能是相同的，也就是这些关注点会在这些元素上产生重叠（overlapping）[14]。这种关注点的重叠就体现为不同的方面组件如何对同一个注入点的横切交互问题。比如，假设某系统不仅需要为某个关键的数据提供安全性操作，而且还要防止数据溢出（overflow），当安全性关注点在某个敏感点对一些数据进行加密和解密的操作时，防止数据溢出的关注点必须事先检查所需要的操作结果是否超出了一个确定的范围而产生了数据溢出。这是一个典型的关注点重叠在一起的问题，因为加密的操作必须要假设加密后的数据应该在数据溢出关注点所规定的范围内，而所提供的解密数据也不能超过这个范围。在体系结构层，这种现象将涉及到基本组件与方面组件之间，方面组件与方面组件之间复杂的交互协议。

针对这种因关注点重叠而造成的方面与方面的交织（intertwined）问题，避免降低方面组件的可重用性，AC2-ADL 利用方面连接件来明确的表达面向方面的体系结构中基本组件与方面组件之间的交互和协作关系。与组件相似，方面连接件也由一组接口组成，这些接口通常被称之为角色。在 AC2-ADL 中，方面连接件包括两种角色类型（如图 1 所示）：基本角色（baseRole）和横切角色（crosscuttingRole），它们定义了该连接件所表示交互的参与者，如基本角色对应一系列被横切的对象，而横切角色的扮演者往往是方面组件的横切接口。

此外，为了保证体系结构中的组件连接以及它们之间的通信正确，并能够推导出所期待的服务作为它连接的对象，每个角色可以包括 1 个或多个行为（behavior），它们组成了该角色的扮演者需要参与的活动。在实际的连接中，角色中的每个行为可由接口中的操作所对应的方法来扮演。通过连接协议（crosscutting protocol），连接件可以定义角色中行为之间的交互规范，在 AC2-ADL 中分别有 4 种横切交互协议类型，分别为：before，after，around 以及 introduce。图 4 提供了方面链接件的主要语法。

```
aspectConnector::=
  aspectConnector connector_name is {
    baseRole baseRoles
    crosscuttingRole crosscuttingRoles
    crosscuttingProtocols crosscuttingProtocol_expressions |
        {crosscuttingProtocol_expressions;}| null;
    [constrains is {constrains_expressions;}| null;]
}
baseRoles::=
   {Role_name;}|{Role_name behaviors behavior_names;}
crosscuttingRole_name::=
   {Role_name;}|{Role_name behaviors behavior_names;}
crosscuttingProtocol_expressions::=
    crosscuttingRole_name. behavior_name
    crosscuttingProtocol_types
    baseRole_name. behavior_name
    crosscuttingProtocol_types::=
before | after |around| introduce
```

图 4　方面链接件的主要语法

面向方面的体系结构：新的组件以及连接件类型的出现使得系统的软件体系结构变得复杂起来，如何提供便利的方式来描述它们的配置是 AC2-ADL 中另一项需要解决的问题。这其中包括如何定义体系结构层的注入点，以及如规范体系结构的配置等主要问题。针对这两个问题，给出了相应的解决方案。

在面向方面的实现层中，注入点（joinpoint）是程序执行中的一个精确执行点，例如类中的一个方法调用。然而，在体系结构层，对注入点的描述将会比在实现层中的描述更加抽象，并且所囊括的概念也更加广泛。它可以是接口中的操作，也可以是组件的接口，甚至，是体系结构中的某个连接件或组件，这也意味着方面不仅可以影响组件还能影响组件间的交互。因此，AC2-ADL 明确的定义了四类注入点：method_JP，interfaces_JP，component_JP 和 connector_JP，此外，为了体现横切接口的批量注入机制，提供了一组切点指示器（pointcut designer 简称 pcd），如图 5 所示，一个 pcd 可以指定单个注入点，也可以表示成一组注入点的集合。通过引入通配符“*”以及“and、not”等操作符，架构师能方便并准确地定义相应的切点指示器。如一个 pcd 可以表示为“Component_instance_name. *. *”形式，其语义可解释为：某组件实例中的所有接口中的所有操作都是注入点。

```
pcd::= method_JP | interfaces_JP
    | component_JP | connector_JP
```

```
    | (and pcd pcd) | (or pcd pcd) | (not pcd)
method _JP::=component_instance_name.
    interfaces_name. [method_name| *]
interfaces_JP::=component_instance_name.
            [interface_name| *]
component_JP::= component_instance_name| *
connector_JP::= connector_ instance_name| *
```

图 5　切点指示器

体系结构配置描述了运行时刻各种组件和连接件实例之间的拓扑结构，并提供信息来确定组件是否正确连接、接口是否匹配、连接器构成的通信是否正确，并说明实现要求行为的组合语义。如图 6 所示，AC2-ADL 通过在配置(Configuration)中指明是系统中的哪些元素扮演(plays)了连接件中的哪些角色，以及用哪些元素中的哪些操作与哪些角色中的哪些行为绑定关系(attachment)来完整地描述系统的拓扑结构。

```
Configuration::=
  configuration is {
    ConfigurationBody| { ConfigurationBody;}|null }
      ConfigurationBody::=
pcd plays aspectConnector_Instance_name. baseRole_name |
  aspectComponent_Instance_name. crosscutting_Interfaces_name
  plays aspectConnector_Instance_name. crosscuttingRole_name
  [attachment
    attachment_Expressions| { attachment_Expressions;}| null]
attachment_Expressions::=
    method_name attaches behavior_name
```

图 6　体系结构配置的语法规范

最后，图 7 给出了一个完整的面向方面的体系结构描述规范，包括其定义的各种组件，以及这些组件的实例和实例之间的配置等元素。

```
    Architecture::=
  architecture architecture_name is{
      component_list
      aspectComponent_list
      component_instances component_instance_list
      aspectComponent_instances aspectComponent_instance_list
    [connector_list]
    [aspectConnector_list]
    [connector_instances connector_instance_list]
    [aspectConnector_instances aspectConnector_instance_list]
    [architectural_Configuration Configuration]
  }
component_list::={component;}
component_instance_list::=
    {instance_name is component_name
        [with (parameter_instantiation)];}
aspectComponent_list::={aspectComponent;}
aspectComponent_instance_list::=
    {aspectComponentInstance_name is aspectComponent_name
      [with (parameter_instantiation)];}
    connector_list::={connector;}
aspectConnector_list::= {aspectConnector_name;}
...
```

图 7　体系结构语法规范

综上所述，AC2-ADL 实际上是一种通用的面向方面的软件体系结构描述语言，因此，将其引入到电子商务领域系统的分析和描述也是行之有效的。下面，针对该领域给出相应的描述方案。

3　案例应用

案例研究的是一个基于英式风格的网上拍卖系统[15]：系统为每个买家、卖家设立虚拟账户，提供银行帐号资金和虚拟账户钱币之间的划拨。游客可以注册成为买家或卖家。卖家拍卖自己的物品，买家参与物品的竞拍。针对每一轮的拍卖，系统对拍卖过程中买家的叫价进行仲裁，一旦拍卖结束，系统会计算最高竞标价，然后将扣除了最高价中拍卖系统委托服务费之后的部分存入卖家的信用卡中，并从买家的虚拟账户上扣除成交价与佣金的总和。

本段首先呈现出网上拍卖系统的面向方面的软件体系结构，然后利用 AC2-ADL 分别描述系统中各种体系结构元素。

3.1　网上拍卖系统的软件体系结构

图 8 展示了面向方面的网上拍卖系统的软件体系结构。其中，实线矩形表示传统的组件，而虚线矩形表示方面组件。该系统的核心功能组件分别由提供网上拍卖系统的用户图形界面接口组件 AuctionGUI，封装了拍卖业务逻辑功能的组件 AuctionBusiness 和账户管理组件 Account Management，以及数据库组件 Database 和银行提供的代理组件 BankDelegate 所组成。

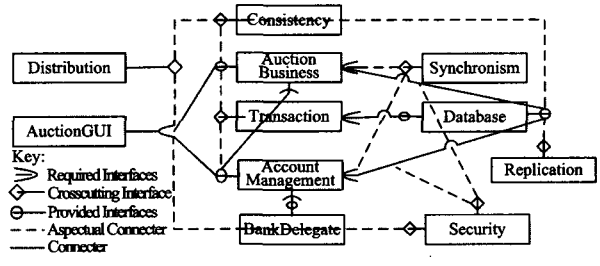


图 8　网上拍卖系统的软件体系结构

另一方面，系统的主要横切关注点由一系列方面组件描述：

- 分布性方面组件(Distribution)：通过使用 RMI 实现户图形界面接口组件(AuctionGUI)对系统服务远程访问，以及业务逻辑功能的组件 AuctionBusiness 等对数据库组件的远程访问；
- 一致性(Consistency)方面组件：用于检查拍卖系统从客户端接收的数据是否一致；
- 事务(Transaction)方面组件：使用数据库组件所提供的事务操作(如回滚事务操作，开始事务操作，提交事务操作等)来支持持久性关注点的实现；
- 同步(Synchronism)方面组件：用于实现对象状态的同步，如标价，用户账户等以确保数据的一致性和持久性；
- 备份(Replication)方面组件：为了保证可用性以及协调网上拍卖系统中各种资源和性能之间的冲突，拷贝方面需要对存储数据进行相应的备份；
- 安全(Security)方面组件：在互联网上，买家和卖家是相互透明的，容易使得不法份子有机可乘。因此，既要验证用户的合法性，也要确保在 Internet 上数据的安全性；

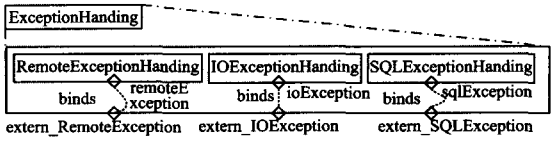
·异常处理(ExceptionHanding)方面组件:用于处理各种异常情况,如通信问题,数据存取问题,一致性问题等。这里由于该方面组件横切的对象众多,为了简化系统的体系架构图,文章省去了该方面组件在图 8 中的表示。

下面针对前面第 2 部分中所提到的面向方面体系结构设计方法中经常出现的异构方面和关注点重叠等问题,通过对网上拍卖系统中两种典型案例的描述,所提供的相应简单可行的设计方法来说明 AC2-ADL 在体系结构描述中的应用。

3.2 描述异构方面组件

这里一个典型的异构方面的例子是异常处理方面组件。在网上拍卖系统中,有很多异常情况需要不同的解决方案,例如通讯问题,数据存取问题,无效数据问题等。每类解决方案可以由一个方面组件来封装,根据设计需要,它们又可以复合成一个更复杂的异常处理方面组件。

如图 9 所示,复合方面组件 ExceptionHandling 的内部子体系结构中,包括多个不同类型的异常处理方面组件,每一个子方面只提供了一种相应的横切接口以提供有关的异常处理功能,这些内部的横切接口通过映射被绑定到复合方面组件的外部接口上,以接收系统发出的异常处理事件信息,从而完成相应的异常处理任务。图 9 展示了该方面的主要描述文本,并利用 XYZ/E 中的选择语句表达其内部的逻辑行为。




```
aspectComponent ExceptionHandling is{
  crosscuttingInterface extern_RemoteException is{}
  crosscuttingInterface extern_IOException is{}
  …
  subArchtecture exceptionHandling_SubStructure is{
  aspectComponent remoteExceptionHandling is{
    …
    Methods print_ErrorLog1 is out (string errorInformation1)
    CrosscuttingInterface remoteException is{
      add_Operation print_ErrorLog1}
      …}
    aspectComponent IOExceptionHandling is{
    …
    CrosscuttingInterface IOException is{…}
    …}
    …
    aspectComponent_instances
      RemoteExceptionHandling_Instance
        is remoteExceptionHandling
      IOExceptionHandling_Instance is IOExceptionHanding
      …}
  mappingDeclaration
    remoteExceptionHandlling_Instance. remoteException
                    binds extra_RemoteExcepton;
    IOExceptionHandling_Instance. IOException
                    binds extra_IOException;
    …
  internalProcess is{
    LB=start⇒!! [
      extra_RemoteException? RemoteExceptionEvent∧SOLB=l11,
      LB=L11⇒print_ErrorLog1∧SOLB=l12
      LB=L12⇒remoteException! errorInformation1∧SOLB=Exit|
      extra_IOException? IOExceptionEvent∧SOLB=l21
      LB=L21⇒print_ErrorLog2∧SOLB=l22
      LB=L22⇒IOException! errorInformation2∧SOLB=Exit|
      …}
    }
```

图 9 描述异构方面组件 ExceptionHandling

在 XYZ/E 中,选择语句的形式化描述如下所示:

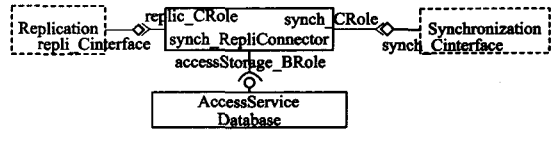
$LB= y \wedge R \Rightarrow !! [C_1 | > D_1, \cdots, C_m | > D_m]$

该表达式的语义被解释为:当前置条件条件 R 满足时,如果来自外部环境的事件 Ci 发生,那么相应的进程 Di 被调用。其中,符号"!!"表示行为的重复。这里,在方面组件 ExceptionHandling 中所描述的内部行为可解释为:通过外部的横切接口(如 extern_RemoteException)接收到一系列系统发出的异常事件消息(如 RemoteExceptionEvent),然后触发内部方面组件(如 RemoteExceptionEvent)提供的相应的异常处理功能(如 print_ErrorLog1 方法),最后通过内部的横切接口(如 remote Exception)将相应的出错信息(如 errorInformation1)发送出去。

3.3 描述相互缠绕的方面组件

关注点重叠的现象也经常出现在网上拍卖系统中,导致同一组注入点上多个方面组件相互缠绕。如图 10 所示,方面组件 Synchronization 和 Replication 在存取数据的活动发生时将产生相互缠绕的行为。具体地说,与前面 3.2 节中描述连接件时所举的例子类似,一方面,Synchronization 方面组件需要考虑多个进程访问同一个存储区的问题,如多个用户同时竞标,另一方面,Replication 方面组件应该检查存储空间的溢出问题。本段通过对方面连接件的描述以解决这类问题。

如图 10 所示,该方案通过定义方面连接件 synch_RepliConnector 呈现方面组件 Synchronization,Replication 和传统组件 Database 之间的交互。该方面连接件主要包含一个基本角色 accessStorage_BRole 和两个横切角色 synch_CRole 和 replic_CRole 以及它们相应的行为。横切协议描述了角色中各行为的交互过程,这里所描述的协议的语义可解释为:当数据存储的行为 dataStoring 或 dataRetrieving 发生时,首先应该对其相应的操作进行加锁(locking),然后进行存储区的检查(storageChecking)以判断溢出问题,当数据存储的行为完成之后再由扮演该 synch_CRole 角色的横切接口中的相关方法释放该锁(unlocking)。




```
aspectConnector synch_RepliConnector is{
  baseRole
  accessStorage_BRole behaviors dataStoring,dataRetrieving;
 crosscuttingRole
synch_CRole behaviors locking,unlocking;
   replic_CRole behaviors storageChecking,storageReplicating
 crosscuttingProtocols
  replic_CRole. storageChecking befor eaccessStorage. * ;
```

```
  synch_CRole. locking before replic_CRole. storageChecking;
  synch_CRole. unlocking after accessStorage. * ;
 }
architecture OAS is{
    ...
  configuration is{
     ...
    batabase. AccessServices plays
    synch_repliConnector. accessStorage_BRole
        attachment
            getEntity attaches dataRetrieving;
            setEntity attaches dataStoring;
    synchronization. synch_CIInterface plays
    synch_RepliConnector. synch_CRole;
        attachmcent…
    replication. replic_CInterface plays
    synch_RepliConnector_instance, replic_CRole;
        attachment…}
}
```

图 10　描述相互交织的方面组件

最后，通过体系结构配置可将这些元素的实例组合起来，从图 10 的最后一段可以看网上拍卖系统配置中所描述的与该方面连接件相关的一部分：方面连接件 synch_RepliConnector 中横切角色 synch_CRole 和 replic_Crole 分别被方面组件 synchronization 和 replication 实例中的横切接口扮演，其中接口中的方法也被附接到相应的行为上（如 getEntity attaches dataRetrieving）。

按照上面的方式，一个完整网上拍卖系统的软件体系结构可以逐步地设计出来，作为一个系统的设计蓝图，它将指导软件后续阶段的开发。

结束语　本文系统地阐述了一套面向方面的体系结构描述语言 AC2-ADL 以及相关描述方法。该语言使得架构师能在关注点分离的原理的基础上使用方面组件描述不同的关注点，并通过方面连接件，把体系结构中的元素组合起来，为体系结构的总体设计和细化提供了一套分层设计的方法，方便了不同的涉众对体系结构设计的理解。方面连接件的灵活使用有效的实现了方面组件和基本组件的重用。方面组件和方面连接件的设计为设计层的方面与实现层的方面的映射提供有力依据，例如，当一个方面组件与某个方面连接件组合时，它们就能确定实现层中的一个与之相关的方面实体，其中方面连接件对应着实现层方面中的织入机制（如 advice 类型等），而方面组件则对应着实现层方面需要完成的横切功能（如 advice 函数体等）。此外，针对目前面向方面的软件体系结构设计中出现的异构方面和关注点重叠问题，本文通过对网上拍卖系统中两种典型案例的描述，提供了相应的简单可行的设计方法。

然而，我们的研究工作仍存在着许多不足，下一步工作的重点包括，进一步完善 AC2-ADL，解决将时序逻辑集成到该体系结构描述语言中时所出现的各种语法上，以及语义上的问题。并在此基础之上，研究在软件体系结构抽象层次上，方面、组件和连接件等软件体系结构元素组合起来的编织机制，以及研究编织在一起的软件体系结构方面、组件和连接器之间的相互协作机制。

参 考 文 献

[1] Medvidovic N , Taylor RN . A classification and comparison framework for software architecture description language[J]. IEEE Transactions on Software Engineering, 2000, 26(1): 70-93

[2] Kiczales G , Lamping J , Mendhekar A , et al. Aspect - Oriented Programming[C]// Proc. of the 11th European Conference on Object-oriented Programming (ECOOP 1997). Lecture Notes in Computer Science. Springer-Verlag, 1997: 220-242

[3] Baniassad E, Clements P, Araujo J, et al. Discovering Early Aspects[J]. IEEE Software, 2006, 23(1): 61-70

[4] Kande M. A concern-oriented approach to software architecture [D]. Lausanne, Switzerland: Swiss Federal Institute of Technology (EPFL), 2003

[5] Kulesza U, Garcia A, Lucena C. Towards a Method for the Development of Aspect-Oriented Generative Approaches [C] // Workshop on Early Aspects, OOPSLA'04. Vancouver, Canada, November 2004

[6] Barais O, et al. TranSAT: A Framework for the Specifcation of Software Architecture Evolution[C]// Workshop on Coordination and Adaptation Techniques for Software Entities, ECOOP 2004. Oslo, Norway, 2004

[7] Chavez C, Garcia A, Kulesza U, et al. Taming Heterogeneous Aspects with Crosscutting Interfaces[J]. Journal of the Brazilian Computer Society, June 2006

[8] Pérez J, Ramos I, Jaén J, et al. PRISMA: Towards Quality, Aspect Oriented and Dynamic Software Architectures[C]// Proceedings of the Third International Conference on Quality Software. November, 2003: 59

[9] Garcia A , Kulesza U , Lucena C. Aspectizing Multi - Agent Systems: From Architecture to Implementation[C]// Engineering for Multi-Agent Systems III, Research Issues and Practical Applications Lecture Notes in Computer Science 3390. Springer, 2005

[10] Pinto M, Fuentes L, Troya J M. A Dynamic Component and Aspect Platform[J]. The Computer Journal, 2005, 48(4): 401-420

[11] Garcia A, Flach C, Batista T, et al. On the Modular Representation of Architectural Aspects[C]// Proc. of the 3rd. European Workshop on Software Architecture (EWSA). Nantes, France, September 2006

[12] Cuestal C E, del Pilar Romay M, de la Fuente P, et al. Architectural Aspects of Architectural Aspects[C]//EWSA 2005: European workshop on software architecture. No2, Pisa, ITALIE (13/06/2005) 20051973, vol. 3527, 2005: 247-262

[13] 唐稚松，等. 时序逻辑程序设计与软件工程[M]. 北京：科学出版社，2002

[14] Katara M , Katz S. Architectural Views of Aspects [C] // Proceedings of the 2nd international conference on Aspect-oriented software development. Boston, Massachusetts, March 2003

[15] Ezhichlevan P, Morgan G. A dependable distributed auction system: Architecture and an implementation framework[C]// Proceedings of 5th International Symposium on Autonomous Decentralized Systems. Dallas (TX), March 2001