基于时序逻辑的面向方面体系结构描述语言

倪友聪[1,2] 应 时[1,3] 张琳琳[1,4] 文 静[1] 叶 鹏[1,5]
(武汉大学软件工程国家重点实验室 武汉 430072)[1] (安徽建筑工业学院数理系 合肥 230018)[2]
(武汉大学计算机学院 武汉 430072)[3] (新疆大学信息科学与工程学院 乌鲁木齐 830046)[4]
(武汉科技学院计算机科学学院 武汉 430074)[5]

摘 要 运用传统体系结构描述语言描述的软件体系结构(SA)方案始终存在着一些横切行为和特征,它们混杂和散列在不同的SA设计单元中,使得SA难以理解、难以演化和难以重用。针对这一问题,基于时序逻辑语言XYZ/E,在统一的时序逻辑框架下设计出一种面向方面体系结构描述语言AC2-ADL。系统地阐述了AC2-ADL的概念框架并用XYZ/E进行语义解释,最后结合案例介绍了如何用AC2-ADL对SA进行描述。

关键词 软件体系结构,软件体系描述语言,时序逻辑,面向方面体系结构描述语言
中图法分类号 TP311.52 **文献标识码** A

Aspect-oriented Architecture Description Language Based on Temporal Logic

NI You-cong[1,2] YING Shi[1,3] ZHANG Lin-lin[1,4] WEN Jing[1] YE Peng[1,5]
(State Key Lab of Software Engineering, Wuhan University, Wuhan 430072, China)[1]
(Department of Mathematics and Physics, Anhui Institute of Architecture & Industry, Hefei 230018, China)[2]
(School of Computer, Wuhan University, Wuhan 430072, China)[3]
(School of Information Science and Engineering, Xinjiang University, Urumqi 830046, China)[4]
(Wuhan University of Science and Engineering, Wuhan 430074, China)[5]

Abstract There always exist some crosscutting behaviors and features in software architecture design based on traditional architectural description language, which tangle and scatter in different design elements of software architecture, leading to the difficulties in comprehension, evolution and reusability of software architectural design decisions. Aiming to these problems, under a unified temporal logic framework, an Aspect-Oriented Architectural Description Language (AC2-ADL) was proposed based on the temporal logic language XYZ/E. The framework and syntax of AC2-ADL were presented and its semantics was explained using XYZ/E. Furthermore, how AC2-ADL can be used for specifying aspect-oriented software architectures was illustrated through case study.

Keywords Software architecture (SA), Software architecture description language (ADL), Temporal logic, Aspect-oriented software architecture description language (AO ADL)

软件体系结构描述语言ADL在较高抽象层次上描述了构成软件系统的元素、元素之间的交互关系、指导元素组合的模式以及相关约束要求,对于软件系统的理解、分析、验证和演化有着十分重要的意义。过去十几年中,已提出了许多的ADL,较为著名的有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了重要的作用[1]。在对软件体系结构的深入研究和广泛应用中,人们发现运用这些传统ADLs所描述的SA方案始终存在着一些横切(crosscutting)行为和特征,它们混杂和散列在组成SA的组件和连接件中,不但造成了组件和连接件之间的紧密耦合,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,其边界也变得模糊起来,进而导致了SA设计方案难以理解、难以演化和难以重用。

为了解决上述问题,基于时序逻辑语言XYZ/E,在统一的时序逻辑框架下,设计出一种面向方面体系结构描述语言AC2-ADL。AC2-ADL不仅能对混杂和散列在SA设计元素中的横切行为和特征进行明确表示,还能对它们在不同位置、按照不同的约束、以不同的时机对组件和连接件的复杂横切影响予以准确描述,使得设计出的SA方案易于理解、易于演化和易于重用。本文系统地阐述了AC2-ADL的概念框架并用XYZ/E进行语义解释,并通过案例介绍如何运用AC2-ADL对面向方面SA进行描述。本文第1节对时序逻辑语言XYZ/E做一简单介绍;第2节给出了AC2-ADL的详细描

到稿日期:2009-02-25 返修日期:2009-05-08 本文受国家自然科学基金资助项目(60773006),高等学校博士学科点专项科研基金资助项目(20060486045)资助。

倪友聪(1976—),男,博生生,讲师,主要研究方向为软件体系结构、面向方面软件开发等,E-mail:nyc@mail.whu.edu.cn;应 时(1965—),男,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。

述;第3节结合案例说明了AC2-ADL的使用方法;第4节将AC2-ADL与其它ADL,特别是面向方面ADL进行相关比较,说明AC2-ADL的优点;最后给出结论和未来工作方向。

1 时序逻辑语言XYZ/E简介

XYZ/E[2]是基于线性时序逻辑系统的一种时序逻辑语言,它结合了静态和动态语义的特点,因而适用于对程序的逐步求精、抽象描述和验证等众多软件工程领域。它最重要的特色是能够用直言式逻辑公式表示出状态转换机制。下面对XYZ/E的基本成份做简要介绍。

在XYZ/E中有一种最基本形式的命令,被称为条件元(ConditionalElement),它有两种形式,如式(1),式(2)所示:

$$LB=y\wedge R\Rightarrow\Diamond(Q\wedge LB=z) \quad (1)$$

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

式(1)用于表示程序的抽象规范,其中符号◇为最终时刻算子;式(2)用于表示程序相邻状态之间的转换关系,其中符号$O为下一时刻算子。式(1)和式(2)中的R,Q为一阶逻辑公式,分别称为条件元的条件部分和动作部分,符号⇒为蕴含词"→"在条件元这一语言层次的特殊表示,y,z分别表示条件元的定义标号和转出标号,它们所在的等式分别称为定义等式和转移等式。

在XYZ/E中单元(Unit)是一个条件元序列(ConditionalElementList),它具有式(3)形式:

$$\Box[A1;\cdots;An]\ \text{WHERE}\ B1\wedge\cdots\wedge Bm \quad (3)$$

式(3)中每一$Ai(i=1,\cdots,n)$为一条件元,规定$A1$中的定义标号为该单元的入口标号,An中的转出标号为该单元的出口标号;$Bj(j=1,\cdots,m)$表示一个以该单元的括号内公式为其作用域的约束条件或是某些特别谓词的定义,符号";"和保留字"WHERE"等同于逻辑联结词合取。单元也是一个时序逻辑公式,其语义对应于它在线性时序模型下的语义。如果一个单元中的所有$Ai(i=1,\cdots,n)$都是式(1)的形式,并且不包含WHERE部分,那么它就构成一个可执行的程序段;如果都是式(2)的形式,那么它就是一个程序的抽象规范;一个单元可以既有式(1)形式的条件元,又有式(2)形式的条件元,从而可以表达不同程度的抽象性。

交互单元之间可以通过通道命令来进行数据交换,以实现单元间通信,通信命令可以分为输入命令和输出命令,其表示形式分别为式(4),式(5):

$$LB=y\wedge R\Rightarrow \$O(ChNm?\ x\wedge LB=z) \quad (4)$$

$$LB=y\wedge R\Rightarrow \$O(ChNm!\ e\wedge LB=z) \quad (5)$$

式(4)中"ChNm? x"与式(5)中"ChNm! e",分别表示由通道ChNm接收信息送入变量x中,由通道ChNm送出表达式e的值。两个单元能够进行通信的必要条件是一个单元在通道上发送特定类型的数据,而另一单元在相同的通道上接收相同类型的数据。

2 AC2-ADL

基于时序逻辑语言XYZ/E,AC2-ADL在线性时序逻辑框架下,统一表示了计算、交互、横切行为及其横切影响。与传统ADL类似,AC2-ADL用组件来表示计算和数据存储,用连接件来表示组件之间的交互,通过SA配置来形成SA实例的拓扑。除此之外,AC2-ADL还引入方面组件作为一等实体来显式地标识和描述横切行为和特征,引入方面连接件作为一等实体来表示横切方以不同的时机、按照不同的约束,对一个或多个被横切方施加复杂的横切影响。由于引入了新的SA元素,AC2-ADL还对传统的SA描述进行了相应扩展,使其既能表示新的结构元素,又能表示新元素与其它元素的组合关系。下面详细阐述AC2-ADL中各种设计元素。

2.1 原子组件

组件是数据存储和计算单元,从外部视角来看,它由一组端口所构成,通过输入、输出端口收发数据,组件能向外部提供服务或请求外部服务。从内部视角来看,组件由一组属性和操作所构成,用于实现其计算承诺。组件可分为原子组件和复合组件两种,原子组件是不可再分的设计元素,其语法定义如图1所示(加下划线的符号均为元语言中的符号,下同)。其中:

1)原子组件的状态是由属性区(Attributes)中的一组属性来定义;

2)端口(Port)是由通道的传输方向、数据类型以及由XYZ/E单元所表示的通道行为来共同定义;

3)内部过程区(InteranlProcess)由一组操作所组成,它们定义了原子组件内部的计算行为。其中,每个操作的内部计算过程由XYZ/E单元来定义。

端口、操作的行为语义均为相应单元在线性时序模型下的语义。

```
AtomicComponent::=
  %AtomicComponent AtomicComponentName==[
    [%Attributes[AttributeDeclPart]]
    %InternalProcess[OperationList]
    %Ports[PortList]
  ]
AttributeDeclPart::=varName:Type {;varName:Type }
OperationList::=Operation {;Operation }
Operation::= OperationName == □[ConditionalElementList][WherePart]
ConditionalElementList::= ConditionalElement {; ConditionalElement }
PortList::=Port{;Port}
Port::=portName== Direction;Type;□[ConditionalElementList]
Direction::=IN | OUT
```

图1 原子组件的语法定义

2.2 连接件

连接件是用来定义组件之间交互方式、交互规则的设计单元,从外部视角来看,它由一组角色所构成,角色规约了交互参与者应具有的行为。从内部视角来看,连接件由一组属性、操作及交互协议所构成。连接件的语法定义如图2所示。其中:

1) 连接件的状态由属性区中的一组属性来定义;

2) 内部过程区中的操作用于辅助连接件交互协议的实现;

3) 交互协议(Glue)用于定义连接件角色之间交互方式、规则和约束;

4) 角色也由通道的传输方向、数据类型及通道行为来表示。

内部过程区中的操作、交互协议、角色的行为都是用

XYZ/E 单元予以定义，它们的行为语义均为相应单元在线性时序模型下的语义。

```
Connector::=
  %Connector connectorName==[
    [%Attributes[AttributeDeclPart] ]
      [%InternalProcess[OperationList] ]
      %Roles[RoleList]
      %Glue glueName = =□[ConditionalElementList][WherePart ]
  ]
RoleList::=Role {;Role }
Role :: = roleName = = Direction; Type; □[ ConditionalElementList]
```

图 2 连接件的语法定义

2.3 复合组件

与原子组件类似，从外部视角来看复合组件也暴露了一组端口，并通过这些端口向外部提供或请求服务；但从内部视角来看，复合组件与原子组件不同之处在于：复合组件可由几个子组件复合而成，子组件又可以是原子组件或复合组件。复合组件的语法定义如图 3 所示。其中：

1) 组件列表(ComList)、组件实例列表(ComInstList)分别为复合组件所包含的子组件类型和子组件实例；

2) 子组件之间可以通过连接件进行连接，连接件类型列表(ConList)、连接件实例列表(ConInstList)分别为连接子组件所要使用的连接件类型及其实例；

3) 绑定(bind)是将子组件实例的端口通过复合组件导出。“comInstName. portName1 bind portName2”的语义解释是：用复合组件的端口名 portName2 去替换子组件实例 comInstName 规约中所有端口名 portName1 的出现，由于子组件也可能是复合的，因而这种替换操作可以逐层下去直至到达原子组件。两个端口能够进行绑定的必要条件是它们的传输方向和数据类型都相同；

4) 粘附(attach)是将组件实例的端口和连接件实例的角色进行连接，使得被连接的组件实例可按连接件实例所规约的交互协议进行交互。“comInstName. portName attach conInstName. roleName”的语义解释是：用组件实例 comInstName 的端口名 portName 去替换连接件实例 conInstName 规约中所有角色名 roleName 的出现。端口能与角色进行粘附的必要条件是它们的传输方向和数据类型都相同。

```
  CompositeComponent::=
%CompositeComponent compositeComponentName = =[
  [%Attributes[AttributeDeclPart ] ]
  [%InternalProcess[OperationList] ]
  %Ports[PortList]
  %ComList[Components]
  [%ConList[Connectors]]
  %ComInstList[ComInsts]
  [%ConInstList[ConInsts]]
  %Attachments[
    [BindList]
    [ConAttachList ]
  ]
]
Components::=Componet {;Componet }
Connectors::=Connector {;Connector }
ComInsts::= comInstName:Component {;comInstName:Component }
ConInsts::= conInstName:Connector {;conInstName:Connector }
BindList::=bindDecl {;bindDecl }
bindDecl::=comInstName. portName bind portName
ConAttachList::=attachDecl{;attachDecl}
attachDecl :: = comInstName. portName attach conInstName. roleName
```

图 3 复合组件的语法定义

2.4 方面组件

方面组件是用来封装横切行为和特征的设计单元，从外部视角来看，它由一组横切端口所构成，通过横切端口收发数据，方面组件向外部提供横切服务。从内部视角来看，方面组件由一组属性和横切操作所构成，它们用于实现方面组件的横切功能。方面组件可分为原子方面组件和复合方面组件两种，前者是不可再分的设计元素，而后者可由几个子方面组件复合而成，子方面组件又可以复合方面组件。方面组件的语法定义图 4 所示。其中：

1) 属性、横切操作、横切端口(CRPort)的定义与组件属性、操作、端口的定义十分类似，但方面组件的属性、横切操作是用于实现其横切功能；此外，与操作不同的是横切操作可以按照指定的时机和约束对组件、连接件进行横切影响；

2) 方面组件列表(AspectComList)、方面组件实例列表(AspectInstList)分别为复合方面组件所包含的子方面组件类型和子方面实例；

3) 横切绑定(CRBind) 是将子方面组件实例的横切端口通过复合方面组件导出，使得子方面组件实例能通过复合方面组件向外部提供相应的横切服务。“AspectInstName. CRPortName1 CRBind CRPortName2”的语义解释是：用复合方面组件的横切端口名 CRPortName2 去替换子方面组件实例 AspectInstName 规约中所有横切端口名 CRPortName2 的出现，由于子方面组件也可能是复合的，因而替换操作可以逐层下去直至到达原子方面组件。两个横切端口能够进行横切绑定的必要条件是它们的传输方向和数据类型都相同。

```
AspectComponent::=
  %AspectComponent aspectComponentName = =[
    [%Attributes[AttributeDeclPart]]
    %InternalProcess[CROperationList]
    %CRPorts[PortList]
    [SubStructureDecl ]
  ]
CROperationList::=CROperation {;CROperation }
CROperation::=
CROperationName = = □[ConditionalElementList][WherePart ]
CRPortList::=CRPort {;CRPort }
CRPort:: = CRPortName = = Direction; Type; □[ConditionalElementList]
SubStructureDecl::=
  %AspectComList[AspectComponet {;AspcetComponet }]
  %AspectInstList[AspectInstName:AspectComponent
    {;AspectInstName:AspectComponent }]
  %CRPortBindList[ CRBindDecl {;CRBindDecl }]
CRBindDecl::=AspectInstName. CRPortName CRBind CRPortName
```

图 4 方面组件的语法定义

我们认为横切行为之间不会直接进行交互，只有当它们按相同的时机同时横切影响同一注入点(Joinpoint)时，才会发生间接的交互，因而在 AC2-ADL 没有提供方面组件之间直接、显示的交互。

2.5 方面连接件

方面连接件用来定义方面组件与组件、连接件之间横切交互方式、横切交互规则的设计单元，从外部视角来看，它由一个 PCD 角色和一组横切角色所构成。从内部视角来看，方面连接件定义了注入点角色以及它与横切角色之间的横切交互协议。方面连接件的语法定义如图 5 所示。其中：

1) PCD 角色由 PCDRoleName 来标识，它定义了被横切对象的集合；

2) 注入点角色(由 JPRoleName 标识)和横切角色(CRRole)的行为都由 XYZ/E 单元来规约，它们分别定义了参与横切交互的被横切方和横切方应该具有的行为方式；

3) 横切交互协议(CRGLue)定义了横切角色与注入点角色在进行横切交互时，应遵循的规则和约束，它由横切交互协议类型(CRGLueType)以及用 XYZ/E 单元表示的具体横切交互过程(以下用 $Unit_{CR}$ 来代称)所构成。CRGLueType 有 BEFOREGLUE 、REPLACEGLUE 和 AFTERGLUE 3 种，横切交互的约束可以通过 $Unit_{CR}$ 中条件元的前、后置条件以及 $Unit_{CR}$ 的 WHERE 部分以一阶谓词形式进行详细规约。为了解释上述 3 种不同类型横切交互协议的语义，与文献[2]类似，定义一些元语言记号和相关操作：

① 元记号“$OLB{|S1,S2#|}$OLB”表示用 S1 单元的第一个转移等式去替换 S2 单元的最后一个转移等式；

② 元记号“LB{|S1,S2|}$OLB”表示用 S1 单元的第一个定义等式去替换 S2 单元的第一个转移等式；

③ 元记号“LB{|S1,S2#|}$OLB”表示用 S1 单元的第一个定义等式去替换 S2 单元的最后一个转移等式；

④ Merge(S1,S2)表示将 S2 单元的条件元序列追加到 S1 单元的最后一个条件元后，形成一个新的 S1 单元；

⑤ UnitAttach(unit,chName1,chName2)表示用通道名 chName1 替换单元 Unit 中所有通道名 chName2 的出现；

⑥ Unit(JPRoleName)(简记为 $Unit_{JP}$)表示注入点角色 JPRoleName 所在的操作计算单元或交互协议单元。

```
AspectConnector::=
  %AspectConnector aspectConnectorName = =[
    %PCDRole= =PCDRoleName
    %JPRole JPRoleName = = Direction;Type;□[ConditionalElementList]
    %CRRoles[CRRoleList]
    %CRGlue CRGlueName = =
        CRGlueType;□[ConditionalElementList][WherePart]
  ]
CRRoleList::=CRRole{;CRRole}
CRRole::=CRRoleName= = Direction;Type;□[ConditionalElementList]
CRGlueType::= BEFOREGLUE | REPLACEGLUE | AFTERGLUE
```

图 5 方面连接件的语法定义

BEFOREGLUE 可以用下面的 XYZ/E 循环语句来表示：

```
*(JPName∈PCDRoleName)[
  UnitAttach(UnitCR,JPName,JPRoleName);
  $OLB{|UnitJP,UnitCR#|}$OLB;
  LB{|UnitCR,UnitJP|}$OLB;
  Merge(UnitJP,UnitCR);
]
```

上语句表示对 PCDRoleName 中的每一横切对象 JPName，依次进行下述步骤：

Step1 将 $Unit_{CR}$ 的通道名 JPRoleName 用 JPName 替换；

Step2 用 $Unit_{JP}$ 的第一个转移等式去替换 $Unit_{CR}$ 的最后一个转移等式；

Step3 用 $Unit_{CR}$ 的第一个定义等式去替换 $Unit_{JP}$ 的第一个转移等式；

Step4 将 $Unit_{CR}$ 的条件元序列追加到 $Unit_{JP}$ 的最后一个条件元后，形成一个新的 $Unit_{JP}$ 单元。

通过上面 4 个步骤，使得每一个受影响单元 $Unit_{JP}$ 被替换为一新单元，该单元在计算之前首先进行横切交互协议所规约的计算，然后回到原 $Unit_{JP}$ 进行计算。

REPLACEGLUE 可以用下面的 XYZ/E 循环语句来表示：

```
*(JPName∈PCDRoleName)[
  UnitAttach(UnitCR,JPName,JPRoleName);
  LB{|UnitCR,UnitJP|}$OLB;
  Merge(UnitJP,UnitCR);
]
```

AFTERGLUE 可以用下面的 XYZ/E 循环语句来表示：

```
*(JPName∈PCDRoleName)[
  UnitAttach(UnitCR,JPName,JPRoleName);
  LB{|UnitCR,UnitJP#|}$OLB;
  Merge(UnitJP,UnitCR);
]
```

REPLACEGLUE，AFTERGLUE 与 BEFOREGLUE 是类似的，限于篇幅，此处仅给出它们的表示形式，不再予以详细解释。

2.6 体系结构描述

体系结构描述定义了体系结构的构成及其相应的连接关系，形成了体系结构拓扑。它由类型列表、实例列表、配置说明(Configuration)构成。体系结构描述的语法定义如图 6 所示。其中：

1) 类型列表包括组件类型列表(ComList)、连接件类型列表(ConList)、方面组件类型列表(AspectComList)和方面连接件类型列表(AspectConList)；

2) 实例列表包括组件实例列表(ComInstList)、连接件实例列表(ConInstList)、方面组件实例列表(AspectComInstList)和方面连接件实例列表(AspectConInstList)；

3) 配置说明(Configuration)包括绑定说明(BindList)、横切绑定说明(CRBindList)、连接件粘附说明(ConAttachList)和方面连接件粘附说明(AspectConAttachList)。

① BindList 和 ConAttachList 及 CRBindList 在复合组件和方面组件一节中已给出了它们的语义解释。

② 在 AspectConAttachList 中有横切粘附(CRAttach)和 PCD 粘附(PCDAttach)两种。CRAttach 是将方面组件实例的横切端口与方面连接件实例的横切角色进行粘附，从而使

得方面组件实例可以通过方面连接件实例实施相应的横切影响。“aspectInstName. CRPortName CRAttach ACInstName. CRRoleName”的语义解释是:用方面组件实例 apectInstName 的横切端口名 CRPortName 去替换方面连接件实例 ACInstName 规约中所有横切角色名 CRRoleName 的出现。在阐述 PCDAttach 的语义解释之前,下面先给出 SA 层注入点和切点指示器(PCD)的定义。

```
Architecture::=
  %Architecture architectureName==[……
    %ComList[Components]
    [%ConList[Connectors]]
    [%AspectComList[AspectComponents]]
    [%AspectConList[AspectConnectors]]
    %ComInstList[ComponetInsts]
    [%ConInstList[ConnectorInsts]]
    [%AspectInstList[AspectComInsts]]
    [%AspectConInstList[AspectConInsts]]
    [%Configuration[
      [[BindList]]
      [[CRBindList]]
      [[AspectConAttachList]]
      [[ConnectorAttachList]]
    ]
  ] ……
AspectConAttachList::=CRAttachList | PCDAttachList
CRAttachList::=CRAttachDecl {;CRAttachDecl }
CRAttachDecl::= apectInstName. CRPortName CRAttach ACInst-
                Name. CRRoleName
AspectConAttachList::=PCDAttachDecl{;PCDAttachDecl}
PCDAttachDecl::= PCD PCDAttach ACInstName. PCDRoleName
……
```

图 6 体系结构描述的语法定义

为了能表示横切行为在一处或多处对组件、连接件进行横切影响,我们明确定义 SA 层注入点为组件的端口、连接件的角色,并通过切点指示器(PCD)来指定一组 SA 层注入点。PCD 的定义如图 7 所示。PCD 有 ComPCD,ConPCD 和 PCD∨PCD 3 种形式。ComPCD 由组件级(componentJP)、端口级(PortJP)两种不同抽象层次以及由它们形成的复合 PCD(ComPCD∨ComPCD)所组成。ConPCD 由连接件级(ConnectorJP)、角色级(RoleJP)两种不同抽象层次以及由它们形成的复合 PCD(ConPCD∨ConPCD)所组成。ComPCD∨ConPCD 表示一种既影响组件又影响连接件的混合 PCD。

```
PCD::=ComPCD | ConPCD | ComPCD∨ConPCD
ComPCD::=ComponentJP(comExp) | PortJP(portExp)
          | ComPCD∨ComPCD
ConPCD::=ConnectorJP(conExp) | RoleJP(roleExp)
          | ConPCD∨ConPCD
comExp::= * | comInstName | comExp∨comExp
portExp::=comInstName. portName | comInstName. INPORT
          | comInstName. OUTPORT | portExp∨portExp
conExp::= * | conInstName | conExp∨conExp
roleExp::= conInstName. roleName | conInstName. INROLE
          | conInstName. OUTROLE | roleExp∨roleExp
```

图 7 PCD 的语法定义

在 PCD 定义中,符号 comInstNam,PortName 分别表示组件实例名、端口名;conInstName,roleName 分别表示连接件实例名、角色名;保留字 INPROT,OUTPROT,INROLE,OUTROLE 分别表示输入类型端口、输出类型端口、输入类型角色、输出类型角色。通过引入逻辑操作符(“∨”)、保留字及通配符“*”来方便、精确地定义一组 SA 层注入点,例如:PortJP(comInstName. OUTPORT)∨RoleJP(conInstName. OUTROLE)表示名为 comInstName 组件实例所有输出端口以及名为 conInstName 连接件实例的所有输出角色都为 SA 层注入点,在这些位置上都要受到横切影响。

对“PCD PCDAttach ACInstName. PCDRoleName”的语义解释为:用 PCD 所定义的注入点集合去替换方面连接件实例 ACInstName 规约中 PCDRoleName 的出现。

3 案例

下面以自动取款机(ATM)系统的一部分为例来简要说明 AC2-ADL 使用方法。案例中用户通过与 ATM 机交互,可以实现查询余额的功能性需求,在实现这一功能性需求的同时,系统还要实现保证 ATM 机与银行之间通信的机密性的非功能性需求。

设计组件 ATM 并让其通过连接件 Con 与组件 Bank 进行交互,以实现案例中的功能性需求;设计复合方面组件 confdAspect 来封装加密、解密横切行为,以实现案例中的非功能性需求;最后通过方面连接件 encyAC 和 decyAC 以及相应的 SA 配置说明来规约方面组件与组件、连接件的组合关系,从而建立了 SA 模型,如图 8 所示。

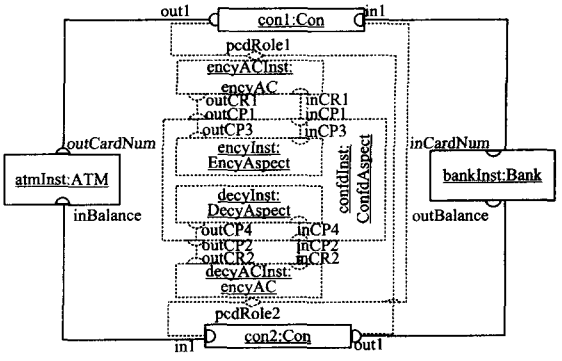

图 8 ATM 系统的部分软件体系结构模型

1)组件规约

ATM 组件的规约如图 9 所示,ATM 组件通过输出端口 outCardNum 输出查询卡号,通过输入端口 inBalance 接收查询结果,从而向外部请求余额查询服务,其具体的内部计算行为由操作 qryBal 来定义。对于 Bank 组件的定义也是类似的。

2)连接件规约

连接件 Con 的规约如图 10 所示,它有两种角色,一为输入角色 in1,其行为是在通道 in1 等待输入数据;另一为输出角色 out1,其行为是在通道 out1 输出数据。内部交互协议 glue1 规约了在 out1 通道上接收到数据 data 后,通过 in1 通道将 data 发送出去。交互协议中通道的行为与对应角色中通道行为恰是相反的,即交互协议中通过某通道输入数据,在角色规约中该通道必为输出数据,反之亦然。

```
%AtomicComponent ATM==[
```

```
%Atrributes[cardNum:STRING… ]
% InternalProcess[
  qryBal= =□[LB=START_qryBal ⇒ $ OLB=L11;
    LB=L11⇒ $ O (outCardNum! cardNum ∧ LB=L12);
    LB=L12⇒ $ O (inBalance? balance ∧ LB=L13);
    LB=L13⇒ $ OLB= EXIT ];…… ]
%Ports[
  outCardNum= =
    OUT;STRING;□[ LB=START⇒ $ OLB=L11;
    LB=L11⇒ $ O (outCardNum1! cardNum ∧ LB=EXIT) ];
  inBalance1= = IN;FLOAT;□[ LB=START⇒ $ OLB=L12;
    LB=L12⇒ $ O(inBalance1? balance ∧ LB=EXIT) ];…… ]
]
```

图 9　ATM 组件

```
%Connector Con= =[
%Atrributes[……]
% InternalProcess[……]
% Roles[
  in1= =IN;ANY;□[LB=START⇒ $ OLB=L21;
      LB=L21⇒ $ O(in1? data ∧ LB=EXIT) ];
  out1= =OUT;ANY;□[LB=START⇒ $ OLB=L22;
      LB=L22⇒ $ O(out1! data ∧ LB=EXIT) ] ]
%Glue glue1= =□[ LB=START_glue1⇒ $ OLB=L23;
      LB=L23⇒ $ O(out1? data ∧ LB=L24);
      LB=L24⇒ $ O(in1! data ∧ LB=EXIT) ]
]
```

图 10　Con 连接件

3)方面组件规约

复合方面组件 ConfdAspect 的规约如图 11 所示，它由加密子方面组件 EncyAspect 和解密子方面组件 DecyAspect 所构成。EncyAspect 通过横切端口 inCP3 接收需加密的数据，通过横切端口 outCP3 输出加密后的数据，从而向外部提供加密横切服务，其具体的内部计算行为由横切操作 encrypt 来规约，其中 encryptCom 表示抽象的加密计算，可在设计过程中对其进行逐步求精。DecyAspect 也是类似的，它在横切端口 inCP4 接收需要解密的数据，通过横切端口 outCP4 输出解密数据。ConfdAspect 的横切端口 inCP1，outCP1 分别与 EncyAspect 实例 encyInst 的横切端口 inCP3，outCP3 进行横切绑定，ConfdAspect 的另外两个横切端口 inCP2，outCP2 分别与 DecyAspect 实例 decyInst 的横切端口 inCP4，outCP4 进行横切绑定，从而使 ConfdAspect 能向外部提供加密和解密横切服务。

```
%AspectComponent EncyAspect= =[ %Atrributes[… ]
  % InternalProcess[
    encrypt = =□[LB=START_encrypt⇒ $ O LB=L41;
      LB=L41⇒ $ O(inCP3? data ∧ LB= L42);
      LB=L42⇒◇(encryptCom ∧ outCP3! enData) ∧ LB= EXIT)
]]
  % CRPorts[
    inCP3= =IN;ANY;□[LB=START⇒ $ OLB=L41;
      LB=L41⇒ $ O(inCP3? data ∧ LB=EXIT)];
  outCP3= =OUT;ANY;□[ ……] ]
]
  %AspectComponent DecyAspect= =[ %Atrributes[… ]
  % InternalProcess[decrypt= =□[LB=START_ decrypt⇒ $ O
LB=L51;
      LB=L51⇒ $ O(inCP4? data ∧ LB= L52);
      LB=L52⇒◇(decryptCom ∧ outCP4! enData) ∧ LB= EX-
IT)] ]
  % CRPorts[
    inCP4 = = IN; ANY; □[……]; outCP4 = = OUT; ANY; □[
……] ]
]
  %AspectComponent ConfdAspect= =[ %Atrributes[… ]
  % InternalProcess[……]
  % CRPorts[
    inCP1 = = IN; ANY; □[……]; outCP1 = = OUT; ANY;
□[……];
    inCP2= =IN;ANY;□[……];outCP2= =IN;ANY;□[……]
]
  %AspectComList[EncyAspect;DencyAspect]
  %AspectInstList[encyInst:EncyAspect;decyInst:DencyAspect]
  %CRPortBindList[
  encyInst. inCP3 CRAttach inCP1;
  encyInst. outCP3 CRAttach outCP1;
  decyInst. inCP4 CRAttach inCP2;
  decyInst. outCP4 CRAttach outCP2]
]
```

图 11　方面组件 ConfdAspect

4)方面连接件规约

方面连接件 encyAC 的规约如图 12 所示，其 PCD 角色由 pcdRole1 标识，两种横切角色为 inCR1 和 outCR1，它们分别定义了在相应通道上等待输入数据和输出数据。encyAC 的内部定义了注入点角色 JPRole1 和横切交互协议 crglue1。JPRole1 规约了 pcdRole1 中的每一个注入点的行为。crglue1 规约了横切交互的类型为 BEFOREGLUE，同时还定义了如下横切交互过程：由 JPRole1 通道接收数据 data 后，通过 inCR1 通道将 data 输出，然后在 outCR1 通道上等待加密后数据 enData，最后由 JPRole1 通道将 enData 送出。方面连接件 decyAC 的规约也是类似的。

```
%AspectConnector EncyAC= =[
  %PCDRole= =pcdRole1
  %JPRole JPRole1= =IN;ANY;□[ LB=START⇒ $ OLB=L71;
    LB=L71⇒ $ O(JPRole1! data ∧ LB=EXIT)]
  %CRRoles[
    inCR1= =IN;ANY;□[ LB=START⇒ $ OLB=L72;
      LB=L72⇒ $ O(inCR1? data ∧ LB=EXIT)]
    outCR1= =OUT;ANY;□[……]
  ]
  %CRGlue crglue1= = BEFOREGLUE;□[
    LB=START_crglue1⇒ $ O (JPRole1? data ∧ LB=L73);
    LB=L73⇒ $ O (inCR1! data ∧ LB=L74);
    LB=L74⇒ $ O (outCR1? enData ∧ LB=L75);
    LB=L75⇒ $ O (JPRole1! enData ∧ LB=NEXT)]
]
%AspectConnector DecyAC= =[
  %PCDRole= =pcdRole2
  %JPRole JPRole2= =OUT;ANY;□[LB=START⇒ $ OLB=
L81;
```

```
    LB=L81⇒$O(JPRole2? enData∧LB=EXIT)]
  %CRRoles[
    inCR2==IN;ANY;□[LB=START⇒$OLB=L82;
      LB=L82⇒$O(inCR2? data∧LB=EXIT)]
      outCR2==OUT;ANY;□[……]
  ]
  %CRGlue crglue2== = BEFOREGLUE ;□[
    LB=START_crglue2⇒$O(JPRole2! enData∧LB=L83);
    LB=L83⇒$O(inCR2! enData∧LB=L84);
    LB=L84⇒$O(outCR2? deData∧LB=L85);
    LB=L85⇒$O(JPRole2! deData∧LB=NEXT)]
]
```

图 12　方面连接件 EncyAC 和 DecyAC

5)SA 描述

案例中 ATM 系统的 SA 描述如图 13 所示，它包含 ATM，Bank，Con，ConfdAspect 和 encyAC，decyAC 等类型，声明了它们所对应的实例。配置说明部分给出了相应的连接关系。

① bankInst 的 inCardNum，atmInst 的 outCardNum 端口分别 attach 到连接件实例 con1 的 in1，out1 角色上，atmInst 的 inBalance，bankInst 的 outBalance 端口分别 attach 到连接件实例 con2 的 in1，out1 角色上，使得 atmInst 可向 bankInst 请求余额查询服务。

② confdInst 的横切端口 inCP1，outCP1 分别 CRAttach 到方面连接件实例 encyACInst 的横切角色 inCR1，outCR1 上，PCD：roleJP(con1. out1) ∨ roleJP(con2. out1)被 PCDAttach 到 encyACInst 的 pcdRole1 上，它表示实际注入点是 con1 的输出角色 out1 和 con2 中的输出角色 out1，从而建模了 confdInst 通过 encyACInst 对连接件 con1 和 con2 进行 before 类型横切影响，提供加密服务。解密横切影响也是类似的。

```
%Architecture ATMSystem==[ ……
  %ComList[ATM;Bank]
  %ConList[Con]
  %AspectComList[ConfdAspect]
  %AspectConList[EncyAC;DecyAC]
  %ComInstList[atmInst:ATM;bankInst:Bank]
  %ConInstList[con1:Con;con2:Con]
  %AspectInstList[confdInst:ConfdAspect]
  %AspectConInstList[encyACInst:EncyAC;decyACInst:DecyAC]
  %Configuration[ ……
    confdInst. inCP1 CRAttach encyACInst. inCR1;
    confdInst. outCP1 CRAttach encyACInst. outCR1;
    confdInst. inCP2 CRAttach decyACInst. inCR2;
    confdInst. outCP2 CRAttach decyACInst. outCR2;
    roleJP(con1. in) ∨ roleJP(con2. in) PCDAttach encyACInst. pcd-
Role1;
    roleJP(con1. out) ∨ roleJP(con2. out) PCDAttach decyACInst.
pcdRole1
  ]
]
```

图 13　ATM 系统的体系结构描述

4　相关工作

随着早期方面技术的提出，解决 SA 层混杂与散列问题已经成为一个研究热点，许多面向方面 ADL 相继提出。DAOP-ADL[3]将方面作为 SA 建模的一等实体，并通过其计算接口来规约可以拦截的消息，利用其目标事件接口来规约可以捕获的事件，通过方面计算规则来规约方面与组件组合时方面行为的执行时机和约束。Fractal ADL[4]利用方面组件来规约 SA 层横切关注点，并通过其特殊的拦截接口来影响组件，方面与组件的组合可以通过直接绑定或基于切点表达式的横切绑定来实现。在 Prisma 方法[5]中，方面作为一种新的设计元素用来定义组件、连接件的结构和行为，通过导入方面来定义组件、连接件的结构，通过规约这些方面间的组合来定义组件、连接件的行为，并针对方面之间不同的组合时机和约束，定义了 6 种不同的编织算子。Navasa 等人在文献[6]中将传统的体系结构描述语言和附加的协调模型相结合，建立了一个面向方面系统的结构化规约，提出了二级体系结构：定义基本系统的组件级、定义方面组件以及它与基本系统交互的方面级，并在方面级定义静态的公共条目(Common Item)结构、规则、控制过程和动态黑板结构等基础设施，利用这些基础设施，并基于通知协议完成方面与组件的组合。AspectualACME[7]在连接件中引入基本角色和横切角色，并通过横切交互协议来定义横切角色对基本角色的横切影响，通过 SA 配置来说明各种角色的扮演关系，连接件和配置结合在一起完成方面与组件的组合。

以上提及的这些主流面向方面 ADL 都存在着一些不足和缺陷。DAOP-ADL 和 Fractal ADL 都没有将方面的计算行为与其所参与的横切交互进行清晰地分离，造成了方面的可重用性降低；此外，二者都是基于 XML 的 ADL，缺乏严格的形式化基础，因而难以对 SA 进行分析和验证。Navasa 等人所提的方法没有较好地将体系结构描述独立于具体实现，使得体系结构设计方案的描述、分析、验证依赖于其所定义的基础设施。DAOP-ADL，Fractal ADL 及 Navasa 方法都没有将连接件作为一等建模实体，使得横切行为可影响的范围仅局限于组件，因而难以描述横切行为对复杂交互的横切影响。利用 Prisma 方法来设计组件和连接件时，SA 设计人员既要决定需要导入哪些方面，又要决定如何规约它们的编织的关系，使得设计出的 SA 方案不仅较为复杂，也较难理解和演化。另外，Prisma 方法还将模块化动作逻辑和 Pi 演算二者结合作为其形式化基础，使得对 SA 的分析、验证不仅较为困难，也较难实施。上述 ADL 和方法中仅 Fractal ADL 和 AspectualACME 提供了对量化机制的支持，但它们都没有显示、规范地定义 SA 层切点表达式，使得方面与其它 SA 设计元素的组合缺乏清晰、一致的语义。文献[2]提出了 XYZ/ADL 的整体构思，文献[8]则系统阐述了 XYZ/ADL 的概念框架，并用 XYZ/E 给出了相关语义解释，但二者都没有将 ADL 与面向方面软件开发有机结合起来，因而缺乏横切行为及其横切影响的相应表示机制。本文借鉴文献[2,8]中的许多思想，在此表示感谢。

结束语　本文基于时序逻辑语言 XYZ/E，在统一的逻辑框架下，设计出一种面向方面体系结构描述语言 AC2-ADL，它显示、精确地定义了 SA 层切点表达式，并用独立设计的方面组件来封装横切行为和特性，利用方面连接件和 SA 配置来灵活、准确地定义方面组件与组件、连接件的组合，既能提

(下转第 162 页)

询时间最短的索引结构的磁盘存取次数(PA)进行比值计算，曲线分别为访问叶结点和所有结点的 PA 次数比值。对比结果如图 6 和图 7 所示。随着数据量的增长，Rav-tree 的 PA 比值优势就越明显，效率的提高主要得益于区域评估方法使得 Rav-tree 存储开销减小，以及非启发式规则的查询算法所带来的性能提升。

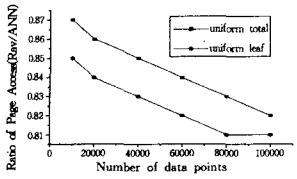

图 6 Rav-tree 与 ANN-tree 在标准数据集的磁盘存取次数比值

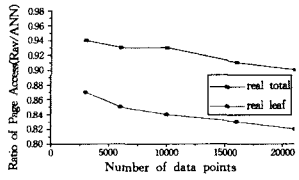

图 7 Rav-tree 与 ANN-tree 在真实数据集的磁盘存取次数比值

RANN 查询是近似查询而非精确查询，是精确性与效率的一种折中。以上实验验证了基于 Rav-tree 的典型查询相比其它索引的查询效率有明显提高。RANN 查询的查全率(Recall)直接影响查询的准确性，特别是在距离度量计算时，查全率是影响计算准确性最重要的因素。因此，实验对 RANN 查询的查全率进行了测试，结果如图 8 所示。k 为类似 RkNN 查询中预先定义的点数量(范围为 5～50)。由结果可知，即使在 k 值较小情况下，83%以上的查全率完全可以满足查询的准确性要求。

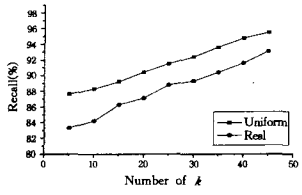

图 8 Rav-tree 在两种数据上的查全率

结束语 不同于启发式规则的索引结构，Rav-tree 利用局部近似得到 ANN 查询结果，进一步结合分域查询实现了 RANN 查询。此方法利用 VC 区域估计和六分区域通过过滤方式快速得到 ANN 查询候选集。Rav-tree 完善了现有索引结构对 RANN 查询的支持。实验结果证明了基于 Rav-tree 的 ANN 查询和 RANN 查询的有效性。

参 考 文 献

[1] Clarkson K L. Nearest Neighbor Queries in Metric Spaces[J]. Discrete & Computational Geometry, 1999, 22(1): 63-93

[2] 刘永山，郝忠孝. 空间对象的反最近邻查询[J]. 计算机科学，2005, 32(11): 115-118

[3] Achtert E, Böhm C, Kröger P. Efficient reverse k-nearest neighbor search in arbitrary metric spaces[C]// Proc. ACM SIGMOD ICMD. Chicago, Illinois, USA, June, 2006: 27-29

[4] Corral A, Manolopoulos Y, Theodoridis Y, et al. Algorithms for processing K-closest-pair queries in spatial databases[J]. Data Knowl. Eng, 2004, 49(1): 67-104

[5] 徐红波，郝忠孝. 一种基于 Z 曲线近似 k-最近对查询算法[J]. 计算机研究与发展，2008, 45(2): 310-317

[6] Lin King-Ip, Yang Congjun. The ANN-tree: An Index for efficient approximate nearest neighbor search[C]// Proc. the 7th International DASFAA. Hong Kong, China, April 2001: 174-181

[7] Sack J R, Urrutia J. Voronoi Diagrams[M]. Handbook on Computational Geometry. Ottawa: Elsevier Science, 2000: 201-290

[8] 李博涵，郝忠孝. 一种基于聚类分析的 R* 树结点重叠判定算法[J]. 计算机研究与发展，2008, 45(12): 2154-2161

(上接第 152 页)

高 SA 元素的重用性，又能灵活地规约方面组件与组件、连接件各种复杂的组合关系，从而使得所设计出的 SA 模型更加易于理解、易于重用、易于演化。我们的下一步工作方向是继续修改和完善 AC2-ADL 的语法和语义，并在此基础上设计出一种显示的编织机制，通过它将 AC2-ADL 建立的 SA 模型编织为仍用 AC2-ADL 描述的仅含组件、连接件的 SA 模型，使得 SA 设计人员更加易于分析并验证 SA 整体行为和质量特征。

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] 唐稚松. 时序逻辑程序设计与软件工程[M]. 北京：科学出版社，2002, 5: 47-49

[3] Pinto M, Fuentes L, Troya J. DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development[C]// Proceedings 2nd International Conference on Generative Programming and Component Engineering, GPCE 2003. Erfurt, Germany, September 2003

[4] Pessemier N, Seinturier L, Coupaye T. A Model for Developing Component-Based and Aspect-Oriented Systems[C]// Proceedings of the 5th International Symposium on Software Composition, SC 2006. Vienna, Austria, March 2006

[5] Perez J, Ramos I, et al. PRISMA: Towards Quality, Aspect Oriented and Dynamic Software Architectures[C]// Proceedings of the 3rd IEEE International Conference on Quality Software, QSIC 2003. Texas, USA, November 2003

[6] Navasa A, Pérez M A, Murillo J M. Aspect Modeling at Architecture Design[C]// Proceedings of the 2nd European Workshop on Software Architecture, EWSA 2005. Pisa, Italy, June 2005

[7] Batista T, Chavez C, Garcia A, et al. Aspectual Connectors: Supporting the Seamless Integration of Aspects and ADLs[C]// Proceedings of the 20th Brazilian Symposium on Software Engineering, SBES'06. Florianópolis, Brazil, October 2006

[8] 朱雪阳，唐稚松. 基于时序逻辑的软件体系结构描述语言 XYZ/ADL[J]. 软件学报，2003, 14 (4): 714-720