软件体系结构层切点指示器的形式化描述方法

倪友聪[1] 叶 鹏[2] 杜 欣[1] 肖如良[1] 张琳琳[3]
(福建师范大学软件学院 福州 350108)[1] (武汉纺织大学数学与计算机学院 武汉 430073)[2]
(新疆大学信息科学与工程学院 乌鲁木齐 830046)[3]

摘 要 软件体系结构层切点指示器是在软件体系结构层次上实现量化机制和描述方面编织的基础。一些面向方面软件体系结构的描述语言虽然引入了切点指示器(Pointcut Designator)的语法成分,但仍未给出其语义的形式化描述,因而难以精确刻画软件体系结构层方面的注入位置。针对这一问题,基于面向方面软件体系结构描述语言AC2-ADL的抽象语法树形式,设计了一种一阶逻辑语言LL4PCD(Logic Language for PCD)。在此基础上,提出AC2-ADL语言PCD的形式化描述方法。该方法能精确定义软件体系结构层PCD的语义,可为形式化分析软件体系结构层方面编织提供支持。

关键词 面向方面软件体系结构,面向方面软件体系结构描述语言,方面编织,切点指示器
中图法分类号 TP311 **文献标识码** A

Formal Description Approach for Pointcut Designator at Software Architecture Level

NI You-cong[1] YE Peng[2] DU Xin[1] XIAO Ru-liang[1] ZHANG Lin-lin[3]
(Faculty of Software, Fujian Normal University, Fuzhou 350108, China)[1]
(College of Mathematics and Computer, Wuhan Textile University, Wuhan 430073, China)[2]
(School of Information Science and Engineering, Xinjiang University, Urumqi 830046, China)[3]

Abstract Pointcut designator(PCD) at software architecture level is a foundation of realizing quantification mechanism and describing aspect weaving in aspect-oriented software architecture. Some Aspect-Oriented Architecture Description Languages (AOADLs) introduce syntax element of PCD, but formal description for semantic of PCD is not given. So it is difficult to accurately describe the injection location at software architecture level. For this problem, this paper proposed a first-order Logic Language for PCD (LL4PCD) based on abstract syntax tree form of AC2-ADL which is a kind of AOADL. Further formal description method for the PCD in AC2-ADL language was proposed on basis of LL4PCD. This method can precisely define the semantic of PCD and support the formal analysis of aspect weaving at software architecture level.

Keywords Aspect-oriented software architecture, Aspect-oriented architecture description language, Aspect weaving, Pointcut designator

1 引言

面向方面程序设计(Aspect-Oriented Programming, AOP)提供了一种模块化横切关注点的技术,有效地支持了关注点分离。经过多年不断的发展,AOP语言的理论研究,特别是在注入点(join point)、切点(pointcut)和通知(advice)等方面编织核心成分的语义定义上已取得了许多研究成果。Andrew等人运用CSP同步集精确地定义了注入点,进而提出一种基于CSP的AOP程序语义[1]。Wand等人首次给出AOP程序中处理动态注入点和递归过程的语义,并在一种微型语言上探讨了动态注入点、切点和通知的指称语义[2]。Belblidia等人通过在语义域中引入环境和配置的描述,给出了AspectJ语言中静态注入点和通知编织的语义定义[3]。

近年来,将AOP的概念、技术和方法扩展到软件体系结构层,构建面向方面软件体系结构(Aspect-Oriented Software Architecture, AOSA)已成为面向方面软件开发研究中的热点问题。目前在AOSA的描述方法上已取得了一定进展,涌现出许多面向方面软件体系结构描述语言(Aspect-Oriented Architecture Description Language, AOADL)。这些AOADL在软件体系结构层方面编织的描述内容和描述形式上不尽相同,特别是在软件体系结构层注入点的定义,在是否支持软件体系结构层量化机制上还存在一些差异。Prisma ADL语

到稿日期:2011-02-25 返修日期:2011-07-30 本文受福建省自然科学基金项目(2011J05146),福建省教育厅项目(JB11029),湖北省教育厅科学技术研究项目(B20111607),新疆维吾尔自治区高校科研计划青年教师科研培育基金(XJEDU2009S15),新疆大学博士毕业生科研启动基金项目(BS090142)资助。

倪友聪(1976-),男,博士,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:youcongni@foxmail.com;叶 鹏 男,博士,讲师,主要研究方向为面向方面软件体系结构、面向服务软件开发等。

言[4,5]引入方面作为独立的软件体系结构元素，并将注入点定义在组件和连接件上。通过设计人员手工导入方面，并规约这些方面之间的编织方式，给出组件和连接件的结构和行为定义。DAOP-ADL 语言[6]将注入点定义在组件请求接口的操作上，运用一系列计算规则来指定组件和方面的编织。Prisma ADL 语言和 DAOP-ADL 语言均不支持软件体系结构层量化机制。AO-ADL 语言[7]是一种基于 XML 的 AOADL，它将组件内部状态、组件接口中的操作以及组件之间的交互定义为注入点。AO-ADL 语言扩充了传统连接件的语义用于表示方面组件的横切影响，并在连接件中定义切点指示器(Pointcut Designator，PCD)，用于量化规约一组注入点。AspectualACME 语言[8,9]将组件的端口和连接件的角色定义为注入点，通过在软件体系结构配置中定义切点指示器，来量化地指定方面组件横切影响的一组注入点。AspectLEDA 语言[10,11]将组件接口中的操作定义为注入点。在静态的公共条目(Common Item)结构、规则和动态黑板等基础设施的支持下，通过定义插入点(类似于切点指示器)可以指定方面横切影响的一组注入点。我们定义了一种面向方面软件体系结构描述语言 AC2-ADL[12-15]，它将组件实例端口中的通道或连接件实例角色中的通道定义为软件体系结构层注入点。AC2-ADL 语言引入了方面组件用于封装混杂和散列在组件、连接件中的横切行为和特性；定义了方面连接件作为一种新的软件体系结构元素，用于描述基本角色和横切角色之间的横切交互协议；通过在编织关系配置说明中定义切点指示器，并将切点指示器与方面连接件实例的基本角色进行粘附，来量化地指定方面组件实例横切影响哪些注入点。

一些 AOADL 语言虽引入了切点指示器的语法成分，但仍未给出其语义的形式化描述，因而难以精确刻画软件体系结构层方面的注入位置。针对这一问题，基于 AC2-ADL 语言的抽象语法树形式，本文设计了一种一阶逻辑语言 LL4PCD，并给出了其语法定义和语义定义。在此基础上，提出 AC2-ADL 语言 PCD 的形式化描述方法。第 2 节首先给出 AC2-ADL 语言 PCD 的语法定义，然后基于抽象语法树，结合实例给出 PCD 的直观解释；第 3 节基于 AC2-ADL 语言抽象语法树结点的函数符号表示和一阶逻辑，详细阐述 LL4PCD 语言的语法定义和语义定义；第 4 节运用 LL4PCD 语言给出 AC2-ADL 语言 PCD 的形式化描述；最后给出结论和未来的工作。

2 AC2-ADL 语言的 PCD

AC2-ADL 语言的 PCD 用于指定一组组件实例端口中特定传输方向和数据类型的通道，图 1 给出了其语法定义。PCD 定义中符号 comInstName、portName 和 chnName 分别表示组件实例名、端口名和通道名；符号 Dir 和 DT 分别用于指定通道的传输方向和数据类型；保留字 PROPORT、REQPORT 分别表示提供类型端口和请求类型端口。AC2-ADL 语言的 PCD 可分为原子 PCD 和复合 PCD，前者是不含任何逻辑操作符的 PCD，而后者则是用逻辑操作符“or”和“not”将其它 PCD 连接起来所形成的 PCD。通过引入逻辑运算符、保留字及通配符“*”，使设计人员可以方便、灵活地指定一组软件体系结构层注入点。

```
PCD::= comInstNameportName.chnName(Dir,DT) |
       comInstNameportName*(Dir,DT) |
       comInstNamePROPORT.*(Dir,DT) |
       comInstNameREQPORT.*(Dir,DT) |
       *.PROPORT.*(Dir,DT)|
       *.REQPORT.*(Dir,DT)|
       comInstName*.*(Dir,DT) |
       *.*.*(Dir,DT) | or(PCD1,PCD2) | not(PCD1)
Dir::=IN | OUT
DT ::=INT | FLOAT | STRING | BOOL | ...
```

图 1 AC2-ADL 语言 PCD 的定义

下面以简单的例子来阐述 AC2-ADL 语言 PCD 的含义。图 2 给出了 AC2-ADL 语言描述的自动取款机 ATM 例子的基本软件体系结构(Basic Software Architecture，BSA)。它定义了 ATM 组件，并声明了其实例 atm1。例中共有两个软件体系结构层注入点，它们分别是 atm1 组件实例 getBal 端口中的 cardCh 和 balCh 通道。AC2-ADL 语言描述的 BSA 经过词法和语法分析后，会得到与之对应的一棵抽象语法树(Abstract Syntax Tree for Basic Software Architecture，AST_{BSA})。图 3 给出了图 2 所示 BSA 所对应的一棵 AST_{BSA}。图 3 中的椭圆形表示结点，旁边的数字字符表示结点的 ID 号；矩形表示结点的属性值；实有向弧表示结点之间的父子关系；虚有向弧表示结点之间的引用关系；无向边表示结点与其属性值之间的关联关系。将 BSA 变换为对应的 AST_{BSA} 后，PCD 的一种直观解释就是在这棵 AST_{BSA} 中，查找满足 PCD 定义条件的软件体系结构层注入点。

```
%BSA[
  %ComList[
    %Com ATM= =[
      %PortList[
        %REQPORT getBal(
          %CHN OUT cardCh: CARDINFO,
          %CHN IN  balCh: FLOAT)= =□[ …… ]
      ]
  ]
  % BSAConf[
    %ComInstDeclList[atm1:ATM]
  ]
]
```

图 2 ATM 例子的基本软件体系结构

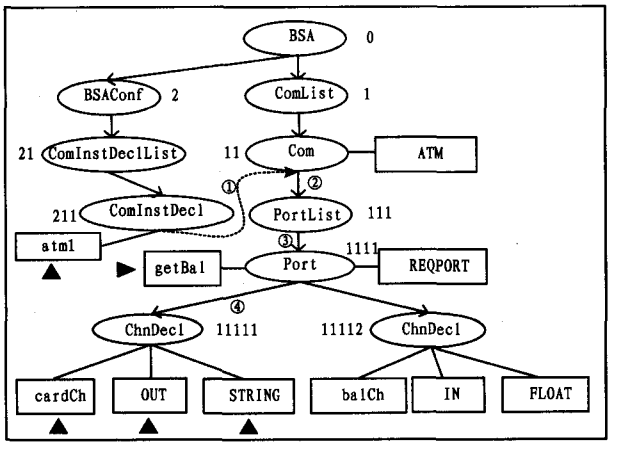

图 3 ATM 例子的 AST_{BSA}

由式(1)定义的原子 PCD1 的直观含义是：在 AST_{BSA} 中查找组件实例名、端口名分别为 atm1、getBal，并且 getBal 端口中存在名为 cardCh、传输数据类型为 STRING、传输方向为 OUT 的通道。在图 4 所示的 AST_{BSA} 中可查找到满足 PCD1 定义条件的一个注入点：atm1 组件实例的 getBal 端口中的 cardCh 通道。该注入点的查找路径由编号为①、②、③

和④的弧所构成，需要比较的结点属性用实心三角形标出。式(2)定义了包含通配符“*”的原子 PCD2，其解释是：在 AST_{BSA} 中查找名为 atml 的组件实例，并获取该实例中所有传输方向为 IN、数据类型为 FLOAT 的通道。在图 3 所示的 AST_{BSA} 中，仅有一个满足 PCD2 定义条件的注入点，即 atml 组件实例的 getBal 端口中的 balCh 通道。式(3)定义了用“or”逻辑运算符将 PCD1 和 PCD2 连接形成的 PCD3。其直观含义是：在 AST_{BSA} 中查找满足 PCD1 或 PCD2 定义条件的注入点。在图 3 所示的 AST_{BSA} 中，有两个满足 PCD3 定义条件的注入点，即 atml 组件实例的 getBal 端口中的 cardCh 和 balCh 两个通道。式(4)定义了用“not”逻辑运算符将 PCD1 连接形成的 PCD4。其解释是：从 AST_{BSA} 确定的所有注入点中，剔除满足 PCD1 定义条件的注入点。在图 3 所示的 AST_{BSA} 中，有一个满足 PCD4 定义条件的注入点，即 atml 组件实例的 getBal 端口中的 balCh 通道。

PCD1::=atml. getBal. cardCh(OUT,STRING) (1)

PCD2::=atml. *. *(IN,FLOAT) (2)

PCD3::=or(PCD1,PCD2) (3)

PCD4::=not(PCD1) (4)

3 LL4PCD 语言

为了形式化地描述 AC2-ADL 语言的 PCD、基于 AC2-ADL 语言的 AST_{BSA}，设计了一种一阶逻辑语言 LL4PCD。下面给出 LL4PCD 语言的语法定义和语义定义。

3.1 LL4PCD 语言的语法定义

LL4PCD 语言的语法由五元组：常元符号集 C、变元符号集 V、函数符号集 F、谓词符号集 P 和逻辑连接词符号集 Con 所定义。

定义 1 LL4PCD::=$\langle C,V,F,P,Con\rangle$。其中：

①常元符号集 C 由 AC2-ADL 语言描述 BSA 所使用的常量名称、变量名称以及结点标识符等构成。

②变元符号集 V 中的变元符号以“#”作为前缀名，以与 BSA 中的变量名和常量名相区分。

③函数符号集 F 包括一组预定义的函数符号。这些函数符号由 AST_{BSA} 的结点类型确定。

④谓词符号集 P 由一元谓词 exist 组成。exist 谓词用于断言 AST_{BSA} 中结点的存在性。

⑤逻辑连接词符号集 Con 由逻辑连接词符号～，∧，∨构成。

定义 1 中的函数符号可以逻辑地表示 AST_{BSA} 的结点。每个函数符号的名称由 AST_{BSA} 结点类型的名称确定，函数符号的第一个参数表示结点的 ID，第二个参数表示父结点的 ID，结点类型确定了函数符号中的其它参数。这些参数用于表示结点的属性、结点与其它结点之间的引用关系。每个参数的取值类型都是字符串。图 4 示意了 AST_{BSA} 的形式，其中的组件实例声明结点 ComInstDecl 可用式(5)表示。函数符号名由结点类型名 ComInstDecl 确定，函数符号的第一个和第二个参数分别指示 ComInstDecl 结点的 ID 和其父结点 ComInstDeclList 的结点 ID，第三个参数表示 ComInstDecl 结点的组件实例名属性，第四个参数表示 ComInstDecl 结点与 Com 组件结点之间的引用关系。

项和逻辑公式是 LL4PCD 语言中的两个基本语法对象，下面分别给出它们的定义。

定义 2 LL4PCD 语言项 term::=$c\mid \#v\mid f(t_1,t_2,\cdots,t_n)$，其中：

①任一常元 $c(c\in C)$是一个项。

②任一变元符号 $\#v(\#v\in V)$是一个项。

③若 $t_1,\cdots,t_n$ 是项，$f(f\in F)$为一个 n 元函数符号，则 $f(t_1,\cdots,t_n)$也是项。

定义 3 LL4PCD 语言的逻辑公式 Form::=exist($f(t_1,\cdots,t_n)$) | $\sim A\mid A\wedge B\mid A\vee B$

①若 $t_1,\cdots,t_n$ 为项，$f(f\in F)$是一个 n 元函数符号，则 exist($f(t_1,\cdots,t_n)$)是公式。

②若 A 是公式，则～A 是公式。

③若 A,B 是公式，则$(A\wedge B)$、$(A\vee B)$都是公式。

由规则①定义的公式称为 LL4PCD 语言的原子公式，规则②和规则③定义的公式为 LL4PCD 语言的复合公式。可用 LL4PCD 语言的闭原子公式(不含变元符号)描述 AST_{SA} 结点的存在性。图 3 所示的 AST_{SA} 中存在属性名为 ATM 的组件结点 Com，可将其描述成式(6)所示的闭原子公式。其中 11 和 1 分别为 ATM 组件结点的 ID 和其父结点 ComList 的 ID。基于 AST_{SA} 结点的存在性，利用 4 个闭原子公式的合取式可精确描述一个软件体系结构层注入点。式(7)给出了图 3 中 atml 组件实例的 getBal 端口中 cardCh 通道注入点的 LL4PCD 语言描述。

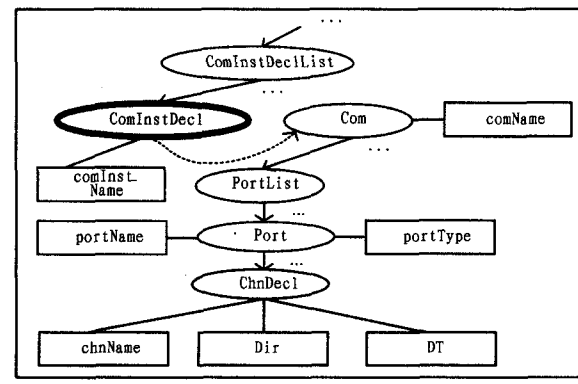

图 4 AST_{BSA} 形式的示意图

ComInstDecl(#*comInstDeclID*, #*comInstDeclListID*, #*comInstName*, #*comID*) (5)

exist(*Com*(11,1,ATM)) (6)

exist(ComInstDecl(211,21,atml,11)) ∧ exist(PortList(111, 11)) ∧ exist(Port(1111, 111, getBal, REQPROT)) ∧ exist(ChnDecl(11111, 1111, *cardCh*,*OUT*,*STRING*)) (7)

LL4PCD 语言是一种特殊的一阶逻辑语言，它的函数符号由 AST_{BSA} 的形式确定，利用闭公式可描述 AST_{BSA} 中结点的存在性和软件体系结构层注入点。

3.2 LL4PCD 语言的语义定义

要使 LL4PCD 语言的项和公式有意义，必须建立 LL4PCD 语言的语义模型。在此基础上定义 LL4PCD 语言的项和公式的语义。

3.2.1 LL4PCD 语言的语义模型

在 LLPCD 语言的 Herbrand 域 $\mathbf{H}$[16] 的基础上，构造了 LL4PCD 语言的语义模型 $M_{Factbase}$。它是一个二元组(S,σ)，其中 S 为 LL4PCD 语言的结构，由论域 $\mathbf{H}$ 和解释 I 组成；σ 为赋值。下面分别对论域 $\mathbf{H}$、解释 I 和赋值 σ 3 部分进行详细

阐述。

(1)论域H

$M_{Factbase}$模型中的论域H由3个部分组成:第一部分是由LL4PCD语言的Herbrand域所定义的非空元素集合H;第二部分是定义在H上的非空函数集合F_H;第三部分是关于H的非空命题集合P_H。

可在H上定义相应的函数以解释LL4PCD中函数符号的含义。对于LL4PCD语言的任一n元函数符号f,都给出其在H上对应的n元函数f_H的定义。

定义4 设f为LL4PCD语言的任意n元函数符号,在H中定义相应的n元函数f_H,其定义域为$H\times\cdots\times H$,值域是H,并且令$f_H(t_1,\cdots,t_n)=f(t_1,\cdots,t_n)$。

当f_H的n个自变量分别取值为H中的n个元素$t_1,\cdots,t_n$时,f_H的函数值也是H中的一个元素$f(t_1,\cdots,t_n)$,它是H中的基项,因而f_H是由字符串构成的n元序偶集到字符串集的函数。

H的非空命题集合P_H中的命题可通过引入关系予以定义,为了在H中定义与exist谓词符号相应的一元关系$exist_H$,首先给出逻辑事实库Factbase的定义。

定义5 称Factbase为AC2-ADL语言描述的BSA所对应的一个逻辑事实库,它满足:若闭原子公式$A=$exist$(f(t_1,\cdots,t_n))$且$A\in$Factbase,当且仅当在该BSA对应的AST_{SA}中存在项$f(t_1,\cdots,t_n)$所表示的结点。

从定义5可以看出:Factbase实质上由一元谓词符号exist形成的闭原子公式的集合,并且一个Factbase与一棵AST_{SA}一一对应。图4给出了图3所示的AST_{SA}所对应的逻辑事实库Factbase。基于Factbase,定义6给出了与exist谓词符号对应的一元关系$exist_H$的定义。

定义6 对于H中的任意一个元素t,称t满足$exist_H$关系,当且仅当exist$(t)\in$Factbase。

$exist_H$是定义在H上的一元关系,它对H中的元素t进行判断,如果exist$(t)\in$Factbase,则t具有$exist_H$关系。

(2)解释I

解释I将LL4PCD语言中的常元符号、函数符号和谓词符号,分别解释为H中的元素、H上的函数和关系。论域H与解释I合在一起称为LL4PCD语言的结构S,如定义7所示。

定义7 LL4PCD的结构$S::=(H,I)$,其中H为论域,I是从LL4PCD到H的映射,简称解释,记I:LL4PCD→H,它满足下列3个条件:

①对LL4PCD中每一常元符号c,$I(c)=c$;

②对LL4PCD中每一n元函数符号f,$I(f)=f_H$;

③对LL4PCD中的一元谓词符号exist,I(exist)$=exist_H$。

(3)赋值σ

赋值σ是一个定义域为变元符号集合V,值域为H的映射,记为σ:$V\to H$。赋值σ把LL4PCD中的每一个变元$\#vi$,赋以H中的一个元素$a\in H$,记为$\sigma(\#vi)=a$。把结构S和赋值σ合在一起,构成了LL4PCD语言的语义模型$M_{Factbase}$。

3.2.2 项和公式的语义

确定了LL4PCD语言的语义模型$M_{Factbase}$后,现可以解释LL4PCD语言中项和逻辑公式的语义。

(1)项的语义

LL4PCD语言中变元和常元被解释为H中的元素,函数符号被解释为H上的函数,项也随之被解释为论域H的元素。

定义8 LL4PCD语言的项term的语义用$term_{M_{Factbase}[\sigma]}$表示,其中$M_{Factbase}$是LL4PCD语言的语义模型,$\sigma$:$V\to H$为赋值。项term的语义$term_{M_{Factbase}[\sigma]}$归纳定义如下:

①$(\#v_i)_{M_{Factbase}[\sigma]}=\sigma(\#v_i)$,$\#v_i$为一变元符号。

②$(c)_{M_{Factbase}[\sigma]}=c$,$c$为一常元符号。

③$f(t_1,t_2,\cdots,t_n)_{M_{Factbase}[\sigma]}=f_H((t_1)_{M_{Factbase}[\sigma]},(t_2)_{M_{Factbase}[\sigma]},\cdots,(t_n)_{M_{Factbase}[\sigma]})$。

式①说明变元符号$\#v_i$的语义是赋值σ在$\#v_i$的取值,它是H中的一个元素。式②说明常元c的语义是H中的元素c。式③说明项$f(t_1,\cdots,t_n)$的语义仍是H中的一个元素。它可以这样得到:把f解释成n元函数f_H,然后分别求$(t_i)_{M_{Factbase}[\sigma]}$,得到$H$中的$n$个值;再求函数$f_H$在$((t_1)_{M_{Factbase}[\sigma]},(t_2)_{M_{Factbase}[\sigma]},\cdots,(t_n)_{M_{Factbase}[\sigma]})$处的值。

(2)逻辑公式的语义

对于LL4PCD语言的逻辑公式而言,由原子公式的真假值确定复合公式的真假值与其它一阶逻辑语言相同。下面仅给出$M_{Factbase}$模型下原子公式的真假值定义。

定义9 原子公式exist(t)在$M_{Factbase}$模型和赋值σ下的真假值,定义为:

如果$(\text{exist}(t))_{M_{Factbase}[\sigma]}\in$Factbase,则$(\text{exist}(t))_{M_{Factbase}[\sigma]}=T$,否则$(\text{exist}(t))_{M_{Factbase}[\sigma]}=F$。

项t在$M_{Factbase}$模型下的语义$(t)_{M_{Factbase}[\sigma]}$满足$exist_H$关系,即$(t)_{M_{Factbase}[\sigma]}$是逻辑事实库Factbase所对应的$AST_{BSA}$中特定结点的函数符号表示时,公式exist$(t)$在$M_{Factbase}$下语义为真,否则为假。

4 AC2-ADL语言PCD的形式化描述

运用上一节所定义的一阶逻辑语言LL4PCD可将AC2-ADL语言的PCD形式化地描述成含变元符号的一阶谓词公式。通过寻找相应的赋值σ(可能存在多个)使得谓词公式在模型$M_{Factbase}$下解释为真,可以精确地确定相应的注入点。

由图1定义的PCD可分成原子和复合两类,共10种不同情形的PCD。每种原子PCD都可用4个LL4PCD语言原子公式的合取式式(8)表示。例如式(1)定义的原子PCD1可被形式化描述成所示的合取式。合取式中的4个原子公式分别用于断言组件实例声明列表中包含名为atm1的组件实例声明、该实例存在端口列表、该实例的端口列表中存在名为getBal的端口、该端口中包含名为cardCh、传输方向和数据类型分别为OUT和STRING的通道。在图5的逻辑事实库Factbase下,是使得合取式$(\text{PCD1})_{LL4PCD}$为真的唯一赋值σ。

```
exist(BSA(0,-1))
exist(BSAConf(2,0))
exist(ComList(1,0))
exist(ComInstDeclList(21,2))
exist(Com(11,1,ATM))
exist(ComInstDecl(211,21,atm1,11))
exist(PortList(111,11))
exit(Port(1111,111,getBal,REQPORT)
exit(ChnDecl(11111,1111,cardCh,OUT,STRING)
exit(ChnDecl(11112,1111,balCh,IN,FLOAT)
```

图5 ATM例子的AST_{BSA}所对应的逻辑事实库

复合 PCD 包括用“or”或“not”连接子 PCD 两种不同形式。对于 or(PCD1,PCD2)形式的复合 PCD,可分别对其两个子切点指示器 PCD1 和 PCD2 进行 LL4PCD 语言描述,然后再用逻辑连接词“∨”连接形成复合公式。例如式(1)、式(2)和式(3)定义的复合 PCD3 可被形式化描述成式(10)所示的析取式。在图 5 的逻辑事实库 Factbase 下,使得式(10)的析取式为真的赋值有式(9)的 σ 和式(11)的 σ'。not(PCD1)形式的复合 PCD,可被描述成两个部分的合取式。前一部分表示所有的注入点,后一部分表示 PCD1 的 LL4PCD 描述的否定。它表达的含义是在所有注入点中剔除 PCD1 所确定的注入点。例如式(1)和式(4)定义的复合 PCD4 可被形式化描述成式(12)所示的合取式。在图 5 的逻辑事实库 Factbase 下,使得式(12)为真的赋值是式(13)的 σ''。

exist(ComInstDecl(#comInstDeclID, #comInstDeclListID, atm1, #comID)) ∧
exist(PortList(#portListID, #comID)) ∧
exist(Port(#portID, #portListID, getBal, #portType)) ∧
exist(ChnDecl(#chnDeclID, #portID, cardCh, OUT, STRING))) (8)

σ={#comInstDeclID=211, #comInstDeclListID=21, #comID=11, #portListID=111, #portID=1111, #portType=REQPORT, #chnDeclID=11111} (9)

(exist(ComInstDecl(#comInstDeclID, #comInstDeclListID, atm1, #comID)) ∧ exist(PortList(#portListID, #comID)) ∧ exist(Port(#portID, #portListID, getBal, #portType)) ∧ exist(ChnDecl(#chnDeclID, #portID, cardCh, OUT, STRING))) ∨ (exist(ComInstDecl(#comInstDeclID, #comInstDeclListID, atm1, #comID)) ∧ exist(PortList(#portListID, #comID)) ∧ exist(Port(#portID, #portListID, #portName, #portType)) ∧ exist(ChnDecl(#chnDeclID, #portID, #chName, IN, FLOAT))) (10)

σ'={#comInstDeclID=211, #comInstDeclListID=21, #comID=11, #portListID=111, #portID=1111, #portName=getBal, #portType=REQPORT, #chnDeclID=11112, #chName=balCh} (11)

(exist(ComInstDecl(#comInstDeclID, #comInstDeclListID, #comInstName, #comID)) ∧ exist(PortList(#portListID, #comID)) ∧ exist(Port(#portID, #portListID, #portName, #portType)) ∧ exist(ChnDecl(#chnDeclID, #portID, #ChnName, #Dir, #DT))) ∧ ~(exist(ComInstDecl(#comInstDeclID, #comInstDeclListID, atm1, #comID)) ∧ exist(PortList(#portListID, #comID)) ∧ exist(Port(#portID, #portListID, getBal, #portType)) ∧ exist(ChnDecl(#chnDeclID, #portID, cardCh, OUT, STRING))) (12)

σ''={#comInstDeclID=211, #comInstDeclListID=21, #comInstName=atm1, #comID=11, portListID=111, #portID=1111, #portName=getBal, #portType=REQPORT, #chnDeclID=11112, #chName=balCh, #Dir=IN, #DT=FLOAT} (13)

AC2-ADL 语言中,其它形式的 PCD 的 LL4PCD 语言描述也类似,本文不赘述。

结束语 基于 AC2-ADL 语言的基本软件体系结构的抽象语法树形式,设计了一种描述软件体系结构层切点指示器的一阶逻辑语言 LL4PCD,定义了其语法和语义。在此基础上给出了运用 LL4PCD 语言形式化描述 AC2-ADL 语言切点指示器 PCD 的方法,使得软件体系结构层 PCD 得以精确解释。下一步工作方向是基于 PCD 的形式化描述,开展软件体系结构层方面编织的形式化分析工作,为发现方面编织中的问题提供有效支持。

参 考 文 献

[1] Andrews J H. Process-algebraic foundations of aspect-oriented programming[A]// Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns, 2001[C]. Berlin, Heidelberg, and New York: Springer-Verlag, 2001: 187-209

[2] Wand M, Kiczales G, Dutchyn C. A Semantics for Advice and Dynamic Join Points in Aspect-Oriented Programming[J]. ACM Transactions on Programming Languages and Systems, 2004, 26(5): 890-910

[3] Belblidia N, Debbabi M. Formalizing AspectJ Weaving for Static Pointcuts[A]// Fourth IEEE International Conference on Software Engineering and Formal Methods, 2006[C]. Pune: IEEE, 2006: 50-59

[4] Pérez J, Ali N. Integrating aspects in software architectures: PRISMA applied to robotic tele-operated systems [J]. Information and Software Technology, 2008, 50(9/10): 969-990

[5] Pérez J, Ramos I, Jaén J, et al. PRISMA: Towards Quality, Aspect Oriented and Dynamic Software Architectures[A]// Proceedings of the Third International Conference on Quality Software (QSIC'03), 2003[C]. Dallas, Texas, USA: IEEE Computer Society, 2003: 59-66

[6] Pinto M, Fuentes L, Troya J M. DAOP-ADL: an architecture description language for dynamic component and aspect-based development[A]// Proceedings of International Conference on Generative Programming and Component Engineering (GPCE '03), 2003[C]. Erfurt, Germany: Springer, 2003: 118-137

[7] Pinto M, Fuentes L. AO-ADL: An ADL for Describing Aspect-Oriented Architectures [J]. Early Aspects: Current Challenges and Future Directions, 2007, 4765(2007): 94-114

[8] Batista T, Chavez C, Garcia A, et al. Aspectual Connectors: supporting the seamless integration of aspects and ADLs[A]// Proceedings of the ACM SIGSoft XX Brazilian Symposium on Software Engineering, 2006[C]. Florianopolis, Brazil: ACM, 2006: 1-16

[9] Garcia A, Chavez C, Batista T, et al. On the modular representation of architectural aspects[A]// Proceedings of Software Architecture, 2006[C]. Nantes, France: Springer, 2006: 82-97

[10] Navasa A, Pérez M, Murillo J. AspectLEDA: Extending an ADL with Aspectual Concepts[A]// Proceedings of Software architecture: first European conference, 2007 [C]. Madrid, Spain: Springer-Verlag, 2007: 330-334

[11] Navasa A, Pérez-Toledano M A, Murillo J M. An ADL dealing with aspects at software architecture stage[J]. Information and

Software Technology,2009,51(2):306-324
[12] 倪友聪,应时,文静,等.一种面向方面软件体系结构中的编织机制研究[J].计算机研究与发展,2010,47(4):695-706
[13] 倪友聪,应时,张琳琳,等.基于时序逻辑的面向方面体系结构描述语言[J].计算机科学,2010,37(1):146-152
[14] Ni Youcong,Ying Shi,et al. Modeling Aspect-Oriented Software Architecture[A]//Proceedings of 2009 International Conference on Industrial and Information Systems (IIS2009),2009[C]. Haikou,China:IEEE,2009:108-113
[15] Wen Jing, Ying Shi, Zhang Lin-lin, et al. AC2-ADL: Architectural Description of Aspect-oriented Systems[J]. International Journal of Software Engineering and its Applications,2009,3(1):1-10
[16] 李未.数理逻辑基本原理与形式演算[M].北京:科学出版社,2007

(上接第 95 页)

使用 OWLS-MX 来查询服务,记录下 OWLS-MX 返回的服务数量。做了 3 组不同数据的测试,详见表 2,分别根据 3 个不同的用户需求比较两种算法的返回结果。最后结果如图 6 所示。

表 2 对比实验数据

	服务集合规模	用户需求
test1	300	car,price
test2	150	novel
test3	200	scholarship,organization

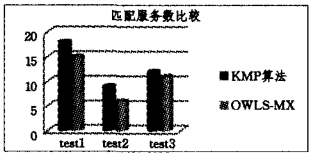

图 6 算法结果对比图

由图可以看出,改进后的 KMP 算法可以比 OWLS-MX 发现更多的潜在可匹配的服务。如在测试 2 中,用户要求输入满足概念 "novel"、返回服务中有一个输入为"publication"的服务,而 OWLS-MX 则没有返回该服务。在测试 3 中,用户检索包含"scholarship,organization"参数的接口,返回了包含"award,government"的服务,语义是强相关的,而 OWLS-MX 也没有检索到。根据以上结果可以证明,与基于本体的语义匹配相比,本文方法可以更好地发现服务之间的语义联系。

6.2.2 算法耗时分析

为了了解算法的执行效率,将特定服务与不同规模服务集进行匹配计算。图 7 为实验的耗时情况。

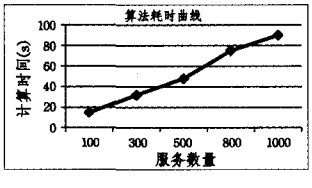

图 7 耗时曲线图

由图 7 可以看出,当服务集的大小为 300 时,计算整个服务库之间服务两两匹配需要将近 20s,而当总服务数量达到 1000 以后,计算的时间仅为 90s。计算得到的总耗时与匹配次数呈线性关系,与第 4 节中得到的计算复杂度相符。平均每次匹配操作所需时间为 100ms 左右,其中包含了服务中概念间相似性计算的时间。因此可以证明,对于管理服务数量在万级的系统(如 Seekda)来说,该方法也是完全适用的,并在原型系统[7]中得到了验证。

结束语 服务匹配是 Web 服务体系的重要支撑技术。如何提高服务匹配的准确性以及效率,是本文研究的重点。利用基于 WordNet 的语义相似性计算以及基于二分图的 KM 匹配算法,提出了的服务匹配 KMP 算法。经实验证明,该方法查全率优于传统语义方法,且算法效率满足实际应用。下一步将考虑通过数据库缓存语义相似性,进一步提高匹配系统的整体执行效率,这对大规模的服务发现是具有实际意义的。

参 考 文 献

[1] 岳昆,王晓玲,周傲英,等. Web 服务核心支撑技术:研究综述[J].软件学报,2004,15 (3):428-442
[2] Christensen E, Curbera F, Meredith G, et al. Web services description language (WSDL) 1. 1[OL]. http://www. w3. org/TR/wsdl, World Wide Web Consortium, W3C Note, March 2001
[3] Miller G A, Beckwith R, Fellbaum C D, et al. WordNet: An online lexical database [J]. International Journal of Lexicography. 1990(3):235-244
[4] Schickel-Zuber V, Faltings B. OSS: A Semantic Similarity Function Based on Hierarchical Ontologies[C]// Proceedings of IJCAI. 2007:551-556
[5] 邓水光,尹建伟,李莹,等.基于二分图匹配的语义 Web 服务发现方法[J].计算机学报,2008,31(8):1364-1375
[6] Zeng Cheng, Guo Xiao, Ou Wei-jie, et al. Cloud Computing Service Composition and Search Based on Semantic[C]// Proceedings of the 1st International Conference on Cloud Computing (CloudCom '09). 2009:290-300
[7] Seekda[OL]. http://www. seekda. com/
[8] Klusch M, Fries B, Sycara K. Automated Semantic Web Service Discovery with OWLS-MX[C]// Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 06). 2006:915-922
[9] Wu J, Wu Z H. Similarity-based Web service matchmaking[C]// International Conference on Services Computing. Orlando, FL, USA: IEEE Computer Society, 2005 (1):287-294
[10] Zhuang Z, Mitra P, Jaiswal A. Corpus-based Web Services Matchmaking[C]// AAAI. Conference. 2005:46-52
[11] Chen Lei, Yang Geng, Wang Dong-rui. WordNet-powered Web Services Discovery Using Kernel-based Similarity Matching Mechanism[C]// The Fifth IEEE International Symposium on Service Oriented System Engineering. 2010:64-68
[12] Plebani P, Pernici B. URBE: Web Service Retrieval Based on Similarity Evaluation [J]. IEEE Transactions on Knowledge and Data Engineering, 2009, 16:1926-1942
[13] Liu Fang-fang, Shi Yu-liang, Yu Jie, et al. Measuring similarity of Web services based on WSDL[C]// IEEE 8th International Conference on Web Services. 2010:155-162