ABC-S²C：一种面向贯穿特性的构件化软件关注点分离技术

梅 宏 曹东刚

（北京大学信息科学技术学院软件研究所 北京 100871）

摘 要 描述了一种在基于构件的软件开发中系统化支持面向贯穿特性的关注点分离技术. 基于构件的软件开发强调通过组装可复用构件支持软件复用，在目标应用有多个关注点、存在贯穿特性的情况下，如何在构件组装时模块化封装各关注点的实现逻辑并将其组织成有机整体是一个重要的问题. 文章以基于构件、面向体系结构的软件开发方法 ABC 为基础，提出通过构件运行支撑平台的支持，在运行时刻动态组织各贯穿特性的方法 ABC-S²C. 其机制是首先引入面向 Aspect 的软件开发中的概念 Advice，用 Advice 对贯穿特性进行建模和模块化封装；其次是将连接子结构化和实体化，通过连接子将各 Aspect 和构件代码关联在一起，由连接子在运行时刻截获对构件的服务请求，按照配置动态调用各贯穿特性的处理逻辑 Advice. 这样的一套以连接子为核心的动态机制支持对黑盒构件的复用，在构件化软件生命周期主要阶段提供了对“贯穿特性”的系统化的模块化支持技术及机制.

关键词 贯穿特性；关注点分离；构件；中间件；Aspect

中图法分类号 TP311

ABC-S²C: Enabling Separation of Crosscutting Concerns in Component-Based Software Development

MEI Hong CAO Dong-Gang

(*Institute of Software, School of Electronics Engineering & Computer Science, Peking University, Beijing* 100871)

Abstract This paper describes an approach to systematically supporting separation of crosscutting concerns in component-based software development. Component-based software development focuses on reusing black-box COTS components. So how to effectively support the modularization of those “crosscutting concerns” without component source code is quite challenging. This paper proposes an approach named ABC-S²C that addresses this issue at runtime with the support from middleware platform. In this approach, a new first-class entity called advice that is the concept of aspect-orientation is introduced to model the crosscutting concerns related to component interactions. The advices and components are dynamically weaved together into another entity named connectors. The connectors and advices are explicit entities supported by the underlying middleware. At runtime, the connector intercepts the service requests and then invokes those configured advices. Since it is the connector but not component that provides the weaving points, the approach effectively addresses the issue of “crosscutting concerns” while facilitates black-box COTS component reuse at the same time.

Keywords crosscutting concern; separation of concern; component; middleware; aspect

收稿日期：2005-03-24；**修改稿收到日期**：2005-08-30. **本课题得到国家“九七三”重点基础研究发展规划项目基金**（2002CB312003）、**国家自然科学基金**（60125206，60233010，90412011）**资助**.**梅 宏**，男，1963 年生，博士，教授，主要研究领域为软件工程、软件复用、软件构件技术、分布计算技术等. E-mail：meih@pku.edu.cn. **曹东刚**，男，1975 年生，博士，讲师，主要研究领域为软件工程、Internet 技术等.

1 引　言

“关注点分离(separation of concerns)”是软件工程中的一个基本原则.“关注点分离”原则要求软件设计人员能够以模块化的方式实现各关注点.然而,由于在目前的程序设计技术框架下,程序语言大多存在“主导分解方式的霸权”(the tyranny of dominant decomposition)限制[1],导致总有一些关注点(如安全、事务、通信等)的实现代码难以模块化,和其它关注点的实现代码彼此贯穿,使得最终的程序呈现“代码交织(code tangling)”和“代码散布(code scattering)”现象.这样的关注点即被称为所谓的“贯穿特性(crosscutting concerns)”.在目前的程序设计技术下,具有贯穿特性的目标系统难于实现、难于理解、难于演化.

为了解决“贯穿特性”问题,20 世纪 90 年代出现了一种称为“面向 Aspect 的程序设计(Aspect Oriented Programming,AOP)”[2,3]技术.AOP 构建在已有的技术基础(如面向对象设计技术)之上,同时将传统方法学中分散处理的贯穿特性集中实现为系统的一阶元素——Aspect,用 Aspect 机制求解非业务关注点,用面向对象技术求解业务关注点,Aspect 和对象的开发是独立进行的.最后利用编排工具将 Aspect 在预定义的插入点(jointpoint)和对象的代码编排重组在一起,形成最终的可运行系统.图 1 给出了一个 AOP 系统运用 Aspect 机制开发软件的过程的简单抽象.

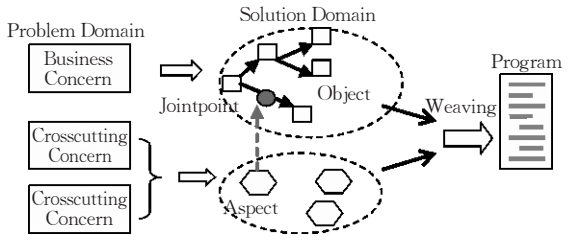


图 1　AOP 机制

AOP 技术允许开发者使用模块化方式分别描述系统的不同关注点及其关系,是一种有效的“关注点分离”技术.然而,目前的 AOP 技术大多是通过编译/预编译源代码或目标代码的方式解决问题,它要求必须掌握一个软件系统所有模块的源代码.对构件化软件中的贯穿特性而言,这种处理方式并不适合,其原因有二:

(1)首先,构件技术强调通过组装可复用构件(尤其是黑盒的 COTS 构件)来生产软件系统,而可复用构件常常是由第三方以无源代码的形式提供给复用者的.无源代码意味着不能直接采用 AOP 的编译插入 Aspect 的方式.

(2)其次,构件化软件的各个构件常常分布于网络的不同自治节点上.提供服务的构件和请求服务的客户方相对独立,如果贯穿特性需要贯穿客户方和服务方双方的代码,则如何在各个分布的客户端进行处理是一个很大的挑战.

图 2 给出了一个构件化软件中的贯穿特性问题示例.

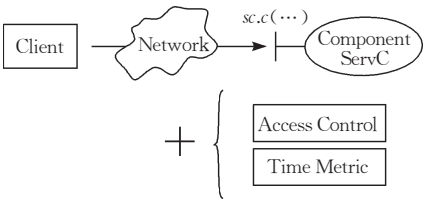


图 2　构件化软件贯穿特性问题示例

在该例中,多个应用客户 Client 通过服务器端构件 ServC 的公共服务接口请求业务服务(如调用 $sc.s(\cdots)$).ServC 是复用的第三方黑盒构件.在只考虑业务关注点的情况下,该软件系统很简单.然而,如果增加下述两个关注点,问题将变得复杂:

(1)访问控制.对访问者的身份进行控制,只有合法的客户才能获得 ServC 的服务;

(2)时间测量.为了某种实时监控或性能优化的目的,度量服务请求的网络传输时间以及在服务器端的执行时间.

“访问控制”要求在客户程序中准备用户身份凭证等信息,在服务器端进行认证授权.而“时间测量”需要在客户程序代码中计算本次调用总共花费的时间 t_{total},在服务器代码中记录请求在服务器端的服务执行时间 t_s,并把 t_s 传送到客户端,计算出传输时间 $t_n=t_{total}-t_s$.由于 ServC 是黑盒构件,且应用客户 Client 和 ServC 不在本地,在需要保持原有模块代码稳定的前提下,编译方式的面向 Aspect 程序设计技术难以解决上述两个关注点的问题.

事实上,“构件化”本身即是一种有效的关注点分离技术——用“构件”来实现系统的各个关注点.但当前的构件模型几乎都是从功能/服务的角度对系统进行描述,大多采用的是“构件接口+构件实现”的模式,对非功能属性(非功能属性是典型的贯穿特性)的支持相对不足.例如在 EJB 构件模型中,业务构件不必处理通信、安全、持久性、事务等传统“贯穿特性”,这些关注点通过声明的方式,交给构件

容器或应用服务器(如 J2EE,CORBA,.NET 等)完成.这一方面使得系统的高层体系结构设计严重依赖于底层中间件平台的具体技术细节;另一方面由于中间件平台提供的服务类型和服务能力基本固定、难以扩展,难以满足如图 2 所示的新的关注点需求,导致高抽象层次的构件复用组装活动变得相当困难.

针对中间件机制和面向 Aspect 的程序设计技术在对贯穿特性的支持方面的优缺点,本文开展构件化软件开发中面向贯穿特性的"关注点分离"技术研究,探讨如何系统化提供对"贯穿特性"的模块化支持技术及机制,从而在体系结构建模、编程、运行时刻等软件生命周期主要阶段,为开发人员提供良好的"关注点分离"技术支撑.基本思想是:

(1)在中间件的支持下,提供一种灵活的运行时刻动态编制(weaving)各关注点代码的机制,从而避免对构件源程序的依赖,支持黑盒复用.

(2)扩展连接子(connector)的语义,使之结构化,提供一种从软件工程角度看待中间件基础设施的视点;引入面向 Aspect 的概念——Advice,用 Advice 对贯穿特性的行为进行描述;通过连接子将 Advice 与构件代码关联在一起,从而使得贯穿特性和构件的开发完全独立进行.

(3)在中间件运行平台中显式支持连接子和 Advice.连接子和 Advice 作为真正的实体存在于运行时刻中,从而和设计时刻的连接子及 Advice 保持良好的对应关系,有效缩短概念鸿沟.

本文的基础是北京大学软件研究所提出的基于体系结构、面向构件的软件开发方法 ABC(Architecture Based Component Composition)[4,5].由于在本文的研究中,连接子关联着代表业务功能的构件和代表贯穿特性的 Aspect,处于一个中心的地位,对连接子的支持贯穿于从体系结构设计到运行时刻等软件生命周期主要阶段,因此,本文的关注点分离框架的中心内容就是名为"ABC-S^{2}C"(Seamless Support of Connectors)的连接子模型.

本文第 2 节描述了 ABC-S^{2}C 技术框架,包括 ABC-S^{2}C 连接子模型、运行时刻的连接子支持机制以及 ABC-S^{2}C 应用开发过程;第 3 节通过一个例子详细演示了 ABC-S^{2}C 如何支持具有多个贯穿特性的构件软件的开发;第 4 节比较了相关研究工作;第 5 节总结全文.

2 ABC-S^{2}C 关注点分离框架

ABC 方法的一个主要目标是缩短体系结构设计到实现的距离,提高基于构件的软件开发的效率.它以体系结构模型作为系统蓝图指导系统的开发全过程,利用中间件技术作为构件组装的实现框架和运行支撑.ABC 的基本概念包括体系结构(architecture)、构件(component)、接口(interface)、连接子(connector)等.ABC 理论框架中的构件,是指软件系统中具有相对独立功能的、可以明确辨识的有机构成成分,是遵循某种构件模型的软件实体.构件的业务行为全部通过接口表达——提供服务的"provides"接口和请求服务的"requires"接口.

由于 ABC-S^{2}C 要解决构件化软件中的贯穿特性问题,除了引入上述 ABC 理论框架中的基本概念,并作了适当调整(例如对连接子的定义)之外,还借鉴面向 Aspect 技术的研究成果,增加了一个名为 Advice 的新的一阶建模元素,该元素是面向 Aspect 的软件开发中的概念,用于对贯穿特性的行为精确建模.Advice 不仅能够描述非业务关注点的行为,也可以描述部分业务构件的功能性行为,此时 Advice 即提供了一种业务功能动态扩展机制,使得开发人员可以对构件功能接口的语义进行重定义或者扩展.

ABC-S^{2}C 主要实体定义如下.

定义 1. 体系结构(architecture).

体系结构 $S_a=\langle I,T,A,C,B\rangle$,其中 $I=\{i\mid i$ 是一个接口$\}$,$T=\{t\mid t$ 是一个功能构件$\}$,$A=\{a\mid a$ 是一个 Advice$\}$,$C=\{c\mid c$ 是一个连接子$\}$,$B=\{b\mid b$ 是构件和连接子的一项绑定$\}$.

定义 2. 接口(interface).

接口 $i=\langle n,M\rangle$,其中 n 是接口的名称,$M=\{m\mid m$ 是一个业务方法$\}$.接口从服务的角度描述了功能构件的业务行为,是 ABC-S^{2}C 连接子模型的核心概念.

定义 3. 构件(component).

构件 $t=\langle n,P,R,o\rangle$,其中 n 是构件实例的唯一标识符,$P=\{p\mid p\in I;p$ 是一个对外提供(provides)的接口 interface$\}$,$R=\{\langle x,r\rangle\mid \exists m\in M,x=m.n,r\in m.P,r$ 是一个所依赖(requires)的接口$\}$,o 是接口 P 的实现体.每个构件必须至少有一个提供接口,可以没有依赖接口.这意味着 ABC-S^{2}C 是从复用的角度考虑构件,即构件至少应当对外提供服务(服务器端构件),而那些客户端构件则不在重点考虑之列.

定义 4. Advice.

Advice $a=\langle m,a_c,a_s\rangle$,其中 $\exists i\in I,m\in i.M,a_c$ 是对交互 m 进行约束的源端动作,a_s 是对交互 m 进

行约束的宿端动作. m 是构件接口中的一个业务方法. Advice 用于描述和构件交互相关的贯穿特性行为，是 ABC-S²C 引入的 Aspect 概念. 关于 Advice 的详细细节将在下一节中介绍.

定义 5.　连接子(connector).

连接子 $c=\langle I,A_x\rangle$，其中 $i\in I$，$A_x=\{a_x\mid a_x\in A, a_x.m\in i.M\}$. 连接子封装了构件间的业务交互(即接口 i)，同时还封装了对业务交互 i 的各种约束(即 Advice 集合 A_x). 因此，在构件间业务接口 i 稳定的情况下，ABC-S²C 连接子的行为主要由各 Advice 决定.

定义 6.　绑定(binding).

绑定 $b=\langle t_c,c,t_s\rangle$，其中 t_c 是源端构件，t_s 是宿端构件，$c\in C$，$c.i\in t_s.P$，$\exists r\in t_c.R$，$c.i=r.i$，$r.n=t_s.n$. 在两个构件交互之前，它们之间必须通过对连接子的配置进行绑定，源端构件 t_c 提供的接口必须和宿端构件 t_s 依赖的接口以及连接子 c 中定义的接口相匹配.

2.1　ABC-S²C 连接子模型

ABC-S²C 连接子模型由三个平面组成：业务平面(business plane)、贯穿平面(crosscutting plane)以及系统平面(infrastructure plane)，如图 3 所示.

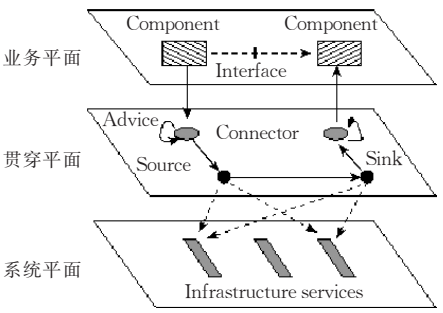


图 3　ABC-S²C 连接子模型

在这三个平面中，业务平面关注应用的业务逻辑，所有的业务构件都位于该平面. 贯穿平面则是贯穿特性和业务关注点所“贯穿”之处，即各关注点最终所综合在一起的胶合点. 当构件交互的时候，连接子将分别用源端及宿端的 Advice，在源端和宿端对构件交互行为进行约束. 系统平面则为 Advice 和构件提供公共基础设施服务. 例如，关于安全的 Advice 可能需要基础设施平面提供的安全服务支持，如 LDAP 服务等.

从图 3 中可以看出，一个逻辑的 ABC-S²C 连接子在物理上由两部分组成：源端的“ConnectorSource”和宿端的“ConnectorSink”. 所谓的源端(source side)和宿端(sink side)，是相对构件交互而言：在一次交互中，发起交互的一方称为源端，接受交互的一方即为宿端. 以传统的客户机/服务器模式的交互为例，客户机即是源端构件，服务器即是宿端构件. ABC-S²C 连接子内部结构如图 4 所示.

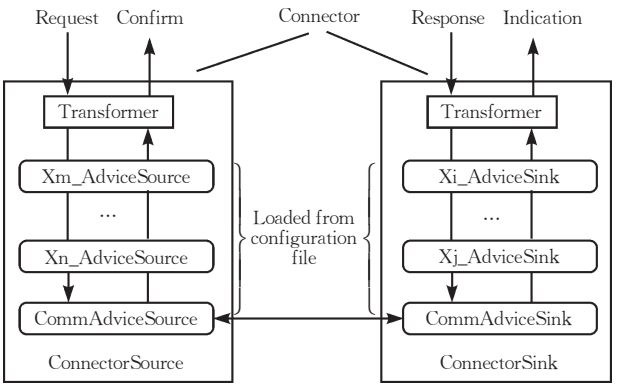


图 4　ABC-S²C 连接子结构图

图 4 中构件间的交互采用了消息传递方式，交互过程分解为源端客户发出请求(request)，宿端服务器构件接收到请求(indication)，宿端服务器构件返回响应(response)，源端客户得到处理结果(confirm). ABC-S²C 连接子中的 Transformer 模块将交互的业务信息编码成一种独立的通用消息格式，使得之后对消息的处理和构件业务类型及具体消息类型无关. 在经过 Transformer 模块处理之后，源端连接子 ConnectorSource 将依次调用本方配置的各 AdviceSource，最后通过 CommAdviceSource 将消息发送到宿端连接子. 宿端的 ConnectorSink 处理过程与源端连接子类似.

在 ABC-S²C 连接子模型中，业务领域的解决方案(各种业务构件)和贯穿特性解决方案(各个 Advice)在连接子中被集中到一起，连接子起到了胶合代码的作用. 因此，贯穿特性是由连接子和 Advice 一起以模块化的方式解决的，连接子提供了类似于“jointpoint”的机制. 这样就可以在不必改动构件的源代码的情况下，通过修改相应的 Advice，改变构件的交互约束，从而改变构件的交互行为.

2.2　ABC-S²C 贯穿特性建模

连接子回答了关于贯穿特性在哪里和构件代码“交织”的问题，即“where”的问题，更核心的何时交织以及贯穿特性怎样描述的问题，即“when and what”，由 ABC-S²C 的 Advice 机制解决.

在 ABC-S²C 的观点看来，大多数的贯穿特性都是实际上对构件业务交互的行为约束，这种约束可以是直接的，也可以是间接的. 例如，日志记录的是交互情况，或者由于交互引起的状态改变；安全和

事务也是针对交互的;负载均衡则要解决由于交互引起的负载分布问题.上述贯穿特性的行为也称为交互行为,这些贯穿特性的行为通常都可以精确描述.

ABC-S^2C 引入了一个新的一阶实体 Advice,用于描述约束构件交互行为的各种复杂的贯穿特性.通常,一个贯穿特性由两部分动作/行为组成:源端动作和宿端动作.前文给出的 Advice 定义,只是从逻辑上描述其含义,而在物理上,由于交互通常是双边行为(多方参与的交互可以分解为若干双边交互),因此 Advice 分为源端(AdivceSource)和宿端(AdviceSink)两部分,二者协同完成一个完整的交互.

以典型的贯穿特性的例子——安全授权(authorization)为例,其问题描述如下:只有已经授权过的用户才能够调用构件 AccountMgr 提供的接口 Account中的方法.

上述的授权关注点可以精确描述和建模:源端的授权 Advice 首先为授权准备相关的主体(principal)信息以及主体凭证(credential)信息,然后将主体及凭证信息传递到宿端的 Advice;宿端授权 Advie 将判断调用者的身份信息,根据本地的安全控制策略(如访问控制列表)决定调用者是否可以进行目标操作(即是否对其授权).

通常典型的可以用 Advice 描述的贯穿特性包括:

(1)通信(communication).构件间的通信可以是本地的,也可以是远程的,通信是最常见的贯穿特性;

(2)安全(security).构件执行安全检查,包括认证、鉴权、加密、审计等;

(3)事务(transaction).保证请求以事务语义执行;

(4)并发(concurrency).控制请求、活动的并发执行情况;

(5)持久性(persistence).为构件提供数据的持久存储服务;

(6)生命周期管理(life-cycle management).控制构件的生命周期;

(7)记帐/日志(profiling/logging).记录请求调用情况;

(8)适配(adaptation).当构件间接口不适配的时候进行适配;

(9)容错(fault-tolerance).对计算构件屏蔽其它计算构件的失效和恢复;

(10)迁移(migration).对计算构件屏蔽其它构件位置的迁移情况;

(11)复制(replication).用一组相同的构件对外提供一个单一的计算服务,等等.

通常上述关注点需要底层基础设施服务的支持,例如安全基础设施、事务服务器、容错基础设施等.这些基础设施会对外提供应用程序接口 API,供 Advice 调用.需要注意的是,一些贯穿特性可能只有源端动作,如源端日志;某些贯穿特性则只有宿端动作,如宿端并发控制.

2.3 运行时刻的中间件支撑机制

在中间件中显式支持 ABC-S^2C 的连接子是本文技术框架的核心.然而,目前的中间件系统只有各种网络通信的概念(如 Socket、连接、报文、stub/skeleton 等等),并没有显式的连接子的概念,因此无法满足 ABC-S^2C 的要求.

ABC-S^2C 要求底层中间件平台支持机制满足下列要求:

(1)透明(transparency).构件通过连接子进行交互,但是连接子机制对于构件而言是透明的.构件开发者和组装者完全遵照标准的构件规范进行开发,从而可以保证已有的在构件方面的投资,同时可以有效复用符合规范的 COTS 构件.

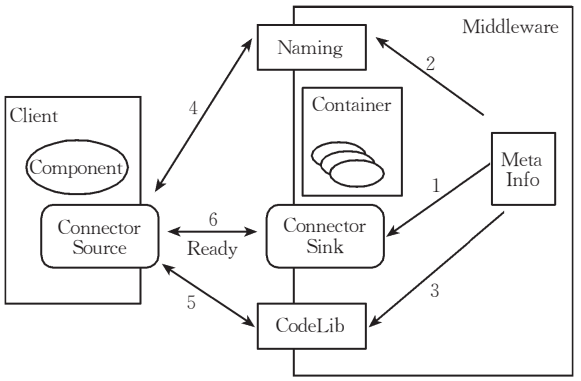
(2)简单(simplicity).连接子机制简单易用.应用开发者可以在不必了解太多中间件底层细节的前提下,即可轻松创建各种 Advices,对构件交互进行约束.应用部署者不是在代码中,而是在部署描述符中说明一个连接子有哪些 Advices.

(3)灵活(flexibility).连接子的定义可以在后开发周期中动态改变.只需在部署描述符中改变连接子的构成,一个构件就可以在下次启动后拥有一个完全不同的交互机制.在运行时刻对连接子进行动态改变也同样是可能的.

ABC-S^2C 运行时刻连接子的支撑机制如图 5 所示.在部署时刻,中间件将首先解析应用部署描述符,得到所有连接子的元信息,其后的工作流程如下:

1.中间件根据从部署描述符里得到的连接子元信息,生成宿端连接子"ConnectorSink",并将之和构件容器绑定.在 ConnectorSink 和构件容器绑定之后,构件即准备好处理客户请求,为客户服务.

2.中间件将构件注册信息发布到一个命名服务——Naming service 中.被发布的注册信息包含了如何在客户端建立源端连接子对象"ConnectorSource"的指令.ConnectorSource 对象封装了客户端的交互逻辑,驻留于客户端地址空间,扮演着一种类似于服务器端构件的代理(proxy)的角色.



1. 生成和绑定 ConnectorSink　　4. 获得 ConnectorSource 信息
2. 发布 ConnectorSource 信息　　5. 获得 ConnectorSource 代码
3. 发布 ConnectorSource 代码　　6. 构造并初始化对象

图 5　ABC-S²C 运行时刻支撑机制

3. 中间件将“ConnectorSource”相关的代码和数据发布到一个代码库(code repository)中. 客户端程序在创建“ConnectorSource”对象的时候，可能需要这些代码和数据. 典型的这类代码和数据是客户端的 Advice 代码. 代码可以是 Java 的class 文件，或者 C/C++的源文件. 在客户端需要的时候，它将根据从 Naming 中获得的指令，从代码库中下载其需要的代码或数据.

4. 客户应用在 Naming 服务中查找提供目标服务(required interface)的构件的注册信息. 在成功查找到目标信息后，它将得到关于如何在本地为和目标构件建立交互而建立 ConnectorSource 对象的指令信息.

5. 如果客户端机器在本地没有建立 ConnectorSource 对象的必需的代码和数据，它就需要根据指令，从网络上的某个代码库(code repository)中下载相关的数据和代码.

6. 在成功下载所有的代码和数据到本地之后，客户端程序即可以构造并初始化 ConnectorSource 对象. 之后，客户程序和服务器构件之间即可通过连接子——ConnectorSource 对象和 ConnectorSink 对象建立绑定. 构件之间的交互机制得以建立. 构件之间即可进行业务交互.

ABC-S²C 的连接子工作机制中要求发布服务的一方同时也要发布如何访问该服务的指令或代码. 在 ABC-S²C 中，每个构件都可以看作通过其“provides”接口对外提供服务. 因此，其访问设施(即连接子)也在部署时刻被发布到 Naming 服务中. 服务请求者(即客户应用)在 Naming 服务中查找该服务，并透明下载必需的数据和代码，以在本地构建负责交互的连接子.

ABC-S²C 对连接子实现机制的具体技术细节不作要求，如何在中间件中支持 ABC-S²C 连接子由中间件提供者自己决定. 目前，我们已经在北京大学软件研究所自主开发的 J2EE 应用服务器系统 PKUAS[6] 的基础上，实现了一个支持 ABC-S²C 的中间件原型系统，限于篇幅，将另行撰文介绍.

2.4　ABC-S²C 应用开发过程

目前大多数基于构件的应用开发过程被分为四个主要的子过程：建模/体系结构设计、构件开发/编程、应用组装和应用部署，这和 ABC 基于软件体系结构的开发过程基本一致. ABC-S²C 是 ABC 理论框架的一部分，但由于其强调对关注点分离的支持，需要显式支持连接子和 Advice 的开发，其应用开发过程如图 6 所示.

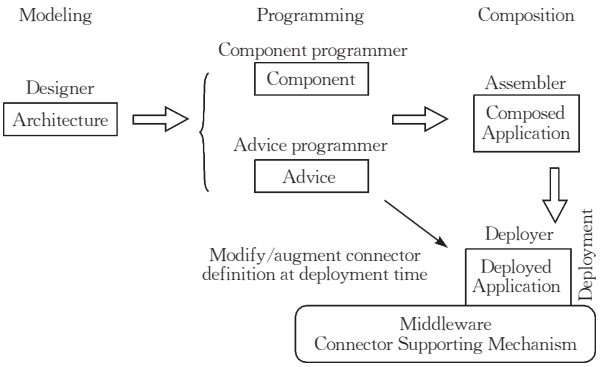


图 6　ABC-S²C 应用开发过程

ABC-S²C 开发过程描述如下：

1. 建模(modeling). 系统开发者在两个平面对应用系统进行建模：在业务平面(business plane)用构件对系统的业务功能进行建模；在贯穿平面(crosscutting plane)用“Advices”对贯穿特性进行建模，Advices 是对构件业务交互的约束，其和构件的交互点被封装在连接子“Connectors”中. 建模的结果即是系统体系结构，该结构不但精确描述了业务行为，还精确描述了贯穿特性的行为.

2. 编程(programming)：与建模阶段相对应，编程阶段同样也有两种角色：面向业务功能的构件开发者和面向贯穿特性的 Advice 开发者，这两类开发人员完全独立. 编程阶段的结果就是各个构件的实现以及各 Advice 的实现(连接子的行为由相应的 Advice 决定). 底层支持 ABC-S²C 的中间件平台要为 Advice 程序员提供应用程序接口 API，从而使各 Advice 可以访问中间件平台所提供的服务.

3. 组装(composition). 应用组装者要把各构件(包括本次开发的或者购买的 COTS 构件)以及连接子与 Advice 等组装到一起，形成最后的应用系统. 在此阶段会定义组装描述符，该描述符提供了应用的元信息，详细定义了构件和连接子的相关信息.

4. 部署(deployment)：应用部署者将把组装得到的应用部署到底层的中间件平台上. 该中间件平台需要提供显式的连接子支撑机制. 注意在部署阶段，应用部署者可能会对连接子的定义进行调整或修改，而无需修改构件的源代码. 例如部署者可以增加一些关于系统监控的特殊“Advices”，对应用以及系统的行为进行检测和管理.

ABC-S²C 应用开发过程很好地体现了“关注点分离”的思想：业务构件的开发和关注贯穿特性的 Advice 的开发是独立的、正交的. 整个系统的行为

可以在部署时刻通过动态修改连接子的定义(增加/删除/修改 Advices)而改变,而且这种改变无需改动构件的源代码,因此带来了极大的灵活性.系统的可理解性和可维护性也大大增强.

3 应用示例

本节以第 1 节引言中图 2 所示的问题为例,演示 ABC-S²C 机制如何在保持原有模块不变的前提下,模块化实现新增的安全关注点和时间度量关注点.由于 PKUAS 预定义了安全服务,在此只给出时间度量的实现机制.

时间度量 Advice 包括源端的 MetricAdviceSource 和宿端的 MetricAdviceSink,其源代码如图 7 所示.MetricAdviceSource 首先计算总花费时间 t_{total},记录到变量 total 中;然后调用 result.getPiggybackContext(key),得到服务执行时间 t_s,记录到变量 ctx 中,最后得到传输时间 t_n,记录于变量 t_{tran} 中.MetricAdviceSink 则计算服务在服务器端的执行时间 t_s,然后调用 result.setPiggybackContext(key, new Long(aft-bef)),将之捎带给对端的 MetricAdviceSource.

```
public class MetricAdviceSource
  extends EJBAdviceSource
{
  public Result advice(Invocation mi)
    throws exception
  {
    long bef=system.currentTimeMillis();

    //All Advices are arranged in a link
    //recursively call next Advice to
    //continue the process
    Result result=getNext().advice(mi);

    long aft=System.currentTimeMillis();
    //compute total time
    long total=aft-bef;
    ContextKey key=new ContextKey("Metric",101);

    //fetch the piggybacked result
    Object ctx=result.getPiggybackContext(key);

    //compute the transport time
    Long ttran=total-((Long) ctx).longValue();
    return result;
  }
}
```

```
public class MetricAdviceSink
  extends EJBAdviceSink
{
  public Result advice(Invocation mi)
    throws Exception
  {
    long bef=System.currentTimeMillis();

    //All Advices are arranged In a link
    //recursively call next Advice to
    //continue the process
    Result result=getNext().advice(mi);

    long aft=System.currentTimeMillis();
    ContextKey key=new ContextKey("Metric",101);

    //piggyback the server-side process time
    result.setPiggybackContext(key,
    new Long(aft-bef));
    return result;
  }
}
```

图 7 时间度量 Advice 源代码

图 8 给出了连接子描述符.连接子的配置和构件的配置被直接放到一起,因此它们之间是一种隐式的绑定(binding).在连接子的配置元素"connector"中,源端连接子和宿端连接子是通过"source"元素和"sink"元素分开配置的,它们各自通过"advice"元素声明 Advice 的类名.MetricAdviceSource 和 MetricAdviceSink 是可复用的模块,其代码和构件代码由服务构件开发者一起打包交中间件运行平台部署.应用的客户端不必在本地编译 MetricAdviceSource 源代码,在运行时刻,它会自动从中间件运行平台代码库下载相关可执行代码.这样的一套动态机制有效避免了前文所述的传统编译模式下的问题.

```
〈enterprise-beans〉
  〈ejb〉
    〈ejb-name〉ServC〈/ejb-name〉
    …
    〈connector〉
    〈source〉
      〈advice〉pku.as.advice.SecurityAdviceSource〈/advice〉
      〈advice〉pkuas.test.time.MetricAdviceSource〈/advice〉
    〈/source〉
    〈sink〉
      〈advice〉pku.as.advice.SecurityAdviceSink〈/advice〉
      〈advice〉pkuas.test.time.MetricAdviceSink〈/advice〉
    〈/sink〉
    〈/connector〉
  〈/ejb〉
〈/enterprise-beans〉
```

图 8 连接子描述符

4 相关研究工作

从软件体系结构的角度,系统主要可分解为构件和构件间的交互,其中构件间的交互由连接子封

装.然而,长久以来,人们对构件交互以及连接子在系统中的角色认识并不深入.

近年来南加州大学的 Medvidovic 等人对构件交互进行了深入分析,并对现有连接子进行了分类,认为软件连接子除了负责传递构件之间的数据以及控制,还应该提供一些服务,如持久性、安全、事务等[7].Medvidovic 等人对连接子的研究对后来的相关研究产生了较大影响,如 SOFA/DCUP 连接子模型.

SOFA/DCUP[8]是捷克 Charles 大学的 Dusan Balek 等提出的一种连接子模型,在实现层次支持了 Medvidovic 的连接子理论.然而,SOFA/DCUP 没有解决好和底层中间件的关系.它的自有构件模型不兼容于现有的主流工业标准,因而无法得到现有主流中间件的支持.

目前较为常见的解决贯穿特性问题的途径,是对现有构件模型进行修改或扩展,或者在程序语言级对编程语言进行扩展,这几种方式的缺点是难以兼容主流的 COTS 构件.例如 COMQUAD 构件模型[9]对 EJB 和 CORBA CCM 构件模型进行了扩展,从而可以在基于构件的系统中支持对非功能属性的规约和运行时刻支持.JAsCo 是一个针对构件应用的面向 Aspect 的程序语言[10],该语言是 Java 的扩展,用一种称为"aspect bean"的实体,用来描述与构件业务功能发生交互的那些非功能的行为;用一个称为"connector"的实体将"aspect bean"部署在特定的构件环境中.JAsCo 和其它 Aspect 语言一样,在将"aspect bean"和构件业务逻辑编排到一起的时候,需要修改构件的源代码.

其它在程序语言级为基于构件的开发提供支持的还有:DJCutter[11],扩展了 AspectJ,支持分布计算环境中的贯穿特性;Jiazzi[12],也是通过对 Java 进行扩展支持面向 Aspect 的编程;等等.

DADO 则是一种在中间件中支持可靠性、安全、日志等贯穿特性的方法[13].DADO 用一对"Adaplet"来对一个贯穿特性进行描述,在运行时刻,Adaplet 被分别附加到应用的 stub 和 skeleton 对象上.DADO 没有显式的连接子的概念,用 Adaplet 对贯穿特性进行描述也不如 Aspect 自然.并且 Adaplet 要附加到生成的 stub/skeleton 上,应用程序也必须事先知道 Adaplet 的代码,这就降低了 Adaplet 的灵活性.

开源 J2EE 应用服务器 JBoss① 以及 J2EE 应用框架 Spring② 分别提供了 Aspect 支持设施 JBoss AOP 和 Spring AOP,其中 JBoss AOP 对 Aspect 的支持粒度较细,能力较强,但是使用成本和学习成本高,相应的支持机制复杂;Spring AOP 对 Aspect 的支持粒度较粗,能力相对较弱,但是使用成本和学习成本低,支持机制相对简单,这二者本质上都是一种支持面向 Aspect 的程序设计的技术框架,关注于在实现层次解决贯穿特性问题.

5　结束语

本文对构件化软件的关注点分离问题进行了初步探讨,提出一种利用中间件平台的支持、实现动态编制各关注点代码的方法 ABC-S²C.ABC-S²C 基于已有的构件化理论研究成果 ABC 方法以及构件运行支撑平台 PKUAS,在软件生命周期主要阶段(从体系结构设计到运行时刻)提供了系统化的对贯穿特性的支持手段,支持复用已有构件模型和 COTS 构件产品.ABC-S²C 提出用连接子将贯穿特性和业务逻辑组织在一起,并在运行时刻在中间件中显式支持连接子,从而提供了一个从软件工程角度看待中间件基础设施的视点.

未来工作包括:(1)尝试在需求阶段开展关注点识别等工作,建立贯穿包括需求建模在内的整个软件生命周期的一致的关注点分离支撑机制;(2)考虑 Advice 之间的冲突检测消解问题;(3)为 PKUAS 提供对应用自定义的连接子和 Advice 进行调试的工具.

参　考　文　献

1　Ossher H., Tarr P.. Using multi-dimensional separation of concerns to (re) shape evolving software. Communications of the ACM, 2001, 44(10): 43～50

2　Kiczales G., Lamping J., Mendhekar A. *et al*.. Aspect oriented programming. In: Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), Finland, 1997, 220～243

3　Cao Dong-Gang, Mei Hong. Aspect orientation——A new approach to programming. Computer Science, 2003, 30(9): 5～10(in Chinese)
(曹东刚,梅　宏.面向 Aspect 的程序设计——一种新的编程范型.计算机科学, 2003, 30(9): 5～10)

4　Chen F., Wang Q., Mei H., Yang F.. An architecture-based approach for component oriented development. In: Proceedings of the 26th International Computer Software and Applications

① (2004)JBoss group website. [Online]. http://www.jboss.org/

② (2005) Spring framework. [Online]. http://www.springframework.org

Conference (COMPSAC'02), Los Alamitos, California, 2002, 450～455

5 Mei Hong, Chen Feng, Feng Yao-Dong, Yang Jie. ABC: An architecture based component oriented approach to software development. Journal of Software, 2003, 14(4): 721～732(in Chinese)
(梅 宏,陈 锋,冯耀东,杨 杰. ABC:基于软件体系结构、面向构件的软件开发方法. 软件学报, 2003, 14(4): 721～732)

6 Huang G., Wang Q., Cao D., Mei H.. Pkuas: A domain-oriented component operating platform. Acta Electronica Sinica, 2002, 30(12a): 39～43

7 Mehta N. R., Medvidovic N., Phadke S.. Towards a taxonomy of software connectors. In: Proceedings of the 22th International Conference on Software Engineering(ICSE'00), Limerick, Ireland, 2000, 178～187

8 Dusan Balek. Connectors in software architectures[Ph. D. dissertation]. Charles University, Czech, 2002

9 Göbel S., Pohl C., Röttger S., Zschaler S.. The COMQUAD component model: Enabling dynamic selection of implementations by weaving non-functional aspects. In: Proceedings of the 3rd International Conference on Aspect-oriented Software Developmen(AOSD'04), Lancaster, UK, 2004, 74～82

10 Suvée D., Vanderperren W., Jonckers V.. JAsCo: An aspect-oriented approach tailored for component based software development. In: Proceedings of the 2nd International Conference on Aspect-oriented Software Development(AOSD'03), Boston, Massachusetts, 2003, 21～29

11 Nishizawa M., Chiba S., Tatsubori M.. Remote pointcut: A language construct for distributed AOP. In: Proceedings of the 3rd International Conference on Aspect-oriented Software Developmen(AOSD'04), Lancaster, UK, 2004, 7～15

12 McDirmid S., Hsieh W. C.. Aspect oriented programming with Jiazzi. In: Proceedings of the 2nd International Conference on Aspect-oriented Software Development(AOSD'03), Boston, Massachusetts, 2003, 70～79

13 Wohlstadter E., Jackson S., Devanbu P.. DADO: Enhancing middleware to support crosscutting features in distributed, heterogeneous systems. In: Proceedings of the 25th International Conference on Software Engineering(ICSE'03), Portland, Oregon, 2003, 174～186

MEI Hong, born in 1963, Ph. D., professor. His current research interests include software engineering and software engineering environment, software reuse and software component technology, distributed object technology.

CAO Dong-Gang, born in 1975, Ph. D., lecturer. His research interests include software engineering, Internet technologies and component-based software development.

Background

This work is supported by the National Basic Research Program of China(973 Program) under grant No. 2002CB312003; the National Natural Science Foundation of China under grant Nos. 60233010, 60125206, 90412011. The research goal of these projects is to improve the methodology and technology of software engineering. The research scopes cover several fields of software engineering, including software reuse and software component technology, domain engineering, software architecture, component operating platform, etc.

The team has made important progress, amongst of which is the proposition of Architecture-Based Component-oriented(ABC) software development approach. The ABC approach proposes to offer an effective systematic solution for component-based reuse by taking advantage of both software architecture and component-based software development. Several tools and platforms have been developed, e. g., the architecture description language ABC/ADL, the architecture modeling tool ABCTool, the component operating platform PKUAS, etc. Some tools are already applied in industrial projects.

The work described in this paper is aimed at systematically supporting separation of crosscutting concerns in component-based software development, which is an important issue within the scope of ABC. The ABC-S^2C approach is based on the existing work of ABC, mainly the architecture description language ABC/ADL and the component operating platform PKUAS.