ABC:基于体系结构、面向构件的软件开发方法*

梅 宏+, 陈 锋, 冯耀东, 杨 杰

(北京大学 信息科学技术学院 软件研究所,北京 100871)

ABC: An Architecture Based, Component Oriented Approach to Software Development

MEI Hong+, CHEN Feng, FENG Yao-Dong, YANG Jie

(Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

+ Corresponding author: E-mail: meih@pku.edu.cn

http://www.sei.pku.edu.cn/belljointlab

Received 2003-01-02; Accepted 2003-01-28

Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. *Journal of Software*, 2003,14(4):721~732.

Abstract: The component-based software reuse and development is considered as an effective and efficient approach to improve the productivity and quality of software development, and is applied widely in building distributed systems. But, current software component technologies are concentrating mostly on component implementation models and runtime interoperability, lacking systematic approach to guide the whole development process. Recently, the research on software architecture (SA) has made significant progress, which takes components as fundamental units and provides a top-down approach to component-oriented development by describing the gross structure and features of software systems. In this paper, an SA-based component-oriented development approach is proposed, trying to offer an effective systematic solution for component-based reuse. This approach introduces the software architecture into each phase of software lifecycle, takes SA as the blueprint of system development, shortens the gap between high-level design and implementation by toolkit support, and realizes the automated system composition on runtime component underpinning platforms.

Key words: software architecture; software component; software reuse

摘 要: 基于构件的软件复用和开发被认为是提高软件开发效率和质量的有效途径,并在分布式系统中得到了广泛的应用.但是,目前的软件构件技术主要还是着眼于构件实现模型和运行时互操作,缺乏一套系统的方法以指导整个开发过程.近年来,以构件为基本单元的软件体系结构研究取得了较大的发展.它通过对软件系统整

* Supported by the National Natural Science Foundation of China under Grant No.60233010 (国家自然科学基金); the National Science Fund for Distinguished Young Scholars of China under Grant No.60125206 (国家杰出青年科学基金); the National High-Tech Research and Development Plan of China under Grant No.2001AA113060 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312003 (国家重点基础研究发展规划(973)); the Major Project of Science and Technology Research of the Ministry of Education of China under Grant No.MAJOR0214 (教育部重大项目)

第一作者简介: 梅宏(1963－),男,重庆人,博士,教授,博士生导师,主要研究领域为软件工程,软件复用,软件构件技术,分布对象技术.

体结构和特性的描述,为面向构件的软件开发提供了一个自顶向下的途径.介绍了一种以软件体系结构为指导,面向构件的软件开发方法,试图为基于构件的软件复用提供一种有效的解决方案.这种方法主要是将软件体系结构引入到软件开发的各个阶段,作为系统开发的蓝图,利用工具支持的自动转换机制缩小从高层设计到实现的距离,而后在构件平台的运行支持下实现自动的系统组装生成.

关键词: 软件体系结构;软件构件;软件复用

中图法分类号: TP311 **文献标识码**: A

基于构件的软件复用作为一种提高软件生产率和软件质量的有效途径,是近几年软件工程界研究的重点之一,被认为是继面向对象方法之后的一个新的技术热潮.一般来说,基于构件的复用包括 3 个相关的过程:构件的开发、构件的管理和基于构件组装的系统开发.文献[1]认为,“在基于构件的软件开发中,系统开发的重点从程序设计变成构件组装”.

近年来,在中间件技术的基础上,结合软件复用思想和面向对象方法,基于构件的软件开发(component based software development,简称 CBSD)技术受到了高度重视.通过标准化运行级构件的规约,依靠构件运行平台(中间件平台)提供的基础设施,CBSD 提供了一种自底向上的、使用标准软件构件构造系统的有效途径,并得到了广泛的应用.

CBSD 的兴起主要是源于下面 4 个背景[2]:在研究方面,现代软件工程思想,特别是对复用技术的强调;在产业方面,支持用构件来建造 GUI、数据库和应用部件的一些理论上质朴但实际可用的技术的成功;在策略方面,某些主流互操作技术,如 CORBA,COM 和 EJB 的开发者的推动;在软件界,对象技术的广泛使用,提供了建造和使用构件的概念基础和实用工具.

CBSD 主要是关注运行级构件和它们之间的互操作,提供了一种自底向上的、基于预先定制包装好的类属元素(构件)来构造应用系统的途径.不过,当前 CBSD 关注的重点都局限在二进制构件的规范上,例如 CORBA,EJB 和 DCOM,仅仅提供了在实现层次上支持构件交互的基础机制,缺少系统化的指导开发过程的方法学.实际上,我们没有理由仅仅局限在运行层次上来看待构件,也不应该只是对代码进行复用,构件复用应该涵盖软件生命周期的各个阶段.当前,虽然也存在一些开发过程,如统一开发过程(unified process)[3]和 CATALYSIS[4],可以用于指导 CBSD,但是,提出这些过程模型的出发点并非针对 CBSD,因此未能全面展现 CBSD 的本质.

自 20 世纪 90 年代初期开始,软件体系结构(software architecture)的研究受到了广泛的关注和重视,并被认为将会在软件开发中发挥十分重要的作用.它将大型软件系统的总体结构作为研究的对象,认为系统中的计算元素和它们之间交互的高层组织是系统设计的一个关键方面[5].其研究和实践旨在将一个系统的体系结构显式化,以在高抽象层次处理诸如全局组织和控制结构、功能到计算元素的分配、计算元素间的高层交互等设计问题[6].作为其最重要的一个贡献,SA 的研究将构件之间的交互显式地表现为连接子(connector),并将连接子视为系统中与构件同等重要的一阶实体.这样,SA 提供了一种在较高抽象层次观察、设计系统并推理系统行为和性质的方式,也提供了设计和实现可复用性更好的构件、甚至复用连接子的途径.

SA 研究的主要成果表现为体系结构描述语言(architecture description language,简称 ADL).从构件组装的角度来看,ADL 可以视为对构件描述语言(CDL)的进一步扩展.构件描述语言的基本思想是将构件看成是一个黑盒,通过描述构件接口的语法和语义,使得复用者不必过多地涉及构件代码细节,就可以在构件描述这一抽象层次之上进行构件组装.而 ADL 除了描述构件接口的语法和语义之外,还负责描述:系统中包括的构件和连接子以及它们之间的交互关系、构件的非功能类性质以及构件间协议,从而为构件组装提供了更为有力的支持.

经过 10 多年的研究,SA 在理论上已经较为成熟.Marry Shaw 在文献[7]中提出,软件体系结构的研究已经度过了开始的发展时期,开始进入完善和应用的阶段.近年来也有一些将 SA 实用化的尝试.例如,Bass 等人对体系结构的风格进行分类整理,试图给出在实际系统开发中应用软件体系结构的指导方法[8];Mary Shaw 在 Unicon 中,通过预定义构件和连接子的种类,可以利用工具在一定程度上自动生成系统(代码)[9].但是这些尝试都不是很成功,其原因首先在于大多数 SA 的研究都还集中在对体系结构的描述和高层性质验证上,对体系结构的求

精和实现的支持能力明显不足[10],例如对于如何在系统开发中选择适当的体系结构风格现在还缺乏行之有效的指导方法.就像文献[11]中所指出的,SA 的研究现在还主要是对已有的软件系统进行整理、描述,而不是如何去指导软件的开发.另一个重要原因是,从 SA 模型到实际系统实现之间存在着较大的距离.由于目前主流的设计和实现语言都是面向对象的,如何从高层抽象的 SA 模型转换到具体的底层实现一直都没有一个比较好的解决方法.对于这个问题,现在也有一些将构件、连接约束等体系结构概念引入编程语言的努力,例如 Washington 大学的 ArchJava[12]和 Utah 大学的 Jiazzi[13].

SA 也是一种基于构件的思想.它从系统的总体结构入手,将系统分解为构件和构件之间的交互关系,可以在高层抽象上指导和验证构件组装过程,提供了一种自顶向下、基于构件的复用途径.而已被业界广泛接受的 CBSD 技术,其相关技术和主要的构件规范已经相当成熟.它不仅定义了构件如何在运行时刻进行交互,而且还提供了使用对象来构造构件的手段,这就在高层的 SA 模型和详细的 OO 设计模型以及具体的 OO 语言实现之间提供了一个现实可行的桥梁.因此,我们在文献[10]中提出了 ABC(architecture based component composition)方法,即将 SA 与 CBSD 相结合,以 SA 模型作为系统蓝图指导系统的开发全过程,把分布式构件技术作为构件组装的实现框架和运行时的支撑,使用工具支持的映射规则缩小设计和实现间的距离,自动地组装、验证所需要的系统.

本文第 1 节从基本思想、建模语言和工具支持等方面介绍 ABC 方法.第 2 节用一个实例说明如何使用 ABC 方法开发软件系统.第 3 节总结全文,并对未来的工作进行展望.

1 ABC 方法概述

ABC 方法的根本思想是在构件组装的基础上,使用 SA 的理论与概念来指导软件开发,以提高系统生成的效率和可靠性.

1.1 基本思想

• **软件体系结构是软件生命周期中的重要产物,它影响到软件开发的各个阶段**.首先,SA 为系统的不同参与者(客户、开发人员、用户等)提供了交流的基础,也是系统理解和演化的基础.其次,SA 体现了系统最早的一组设计决策,对系统的整体特性、后续开发和组织进行了约定.此外,使用 SA 方法不仅可以对软件构件进行复用,还可以实现更高层次上的复用,例如对 SA 模型本身的复用.

进一步来说,SA 应该不仅仅局限在高层设计的描述中,而是要扩展到软件开发的全过程中,充分地发挥 SA 模型在软件开发中的指导作用,并在软件生命周期的各阶段间保持良好的可追踪性.因此,我们提出了基于软件体系结构的软件开发过程,如图 1 所示.

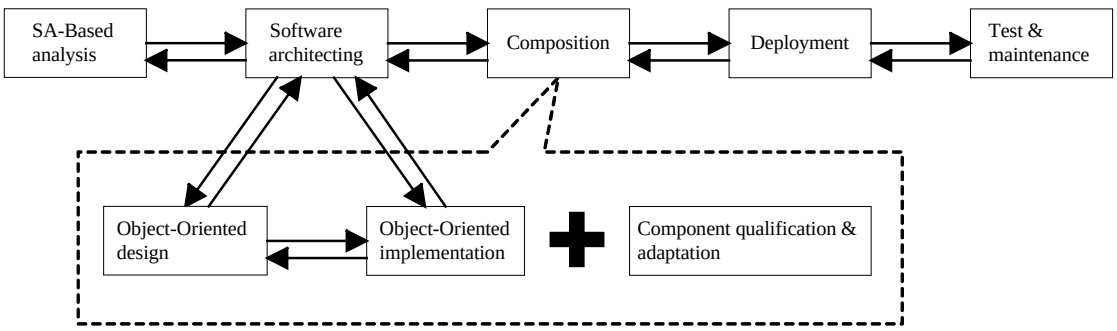

Fig.1 SA-Based software development process

图 1 基于软件体系结构的软件开发过程

在这个开发过程中,首先是基于 SA 的需求获取和分析,将 SA 的概念引入需求空间,从而为分析阶段到设计阶段的过渡提供了更好的支持.在得到需求分析结果的基础上进行体系结构的设计,考虑系统的总体结构以及系统的构成成分,根据构成成分的语法和语义要求在构件库中寻找匹配的构件.当不存在符合要求的构件时,

则需根据具体情况或者根据 CBSD 的原则和方法开发新的构件,或者将某些已有构件进行组装而得到满足需求的构件.在组装阶段,每一个系统构件都具有了实现体,在经过语法和语义检查以后,这些构件将会通过胶合代码组装到一起,最后,被部署到相应的中间件平台上.在实践中,整个开发过程将呈现多次迭代性.

目前,ABC 方法关注的主要还是从 SA 建模到系统的组装、部署阶段的工作.

- **应该在不同的抽象层次上实现构件复用与组装**.基于构件的软件复用没有理由局限在实现层次上,而应该在不同的层次上,针对不同的对象来进行.ABC 方法复用的构件包含有多个层次、多个生命周期阶段的产品,例如需求分析文档、SA 规约、OOD 设计到源代码和二进制代码等.在实现层,利用现有的二进制构件规范(EJB,CORBA,COM),可以对运行级的构件进行复用和组装.在设计层上,一是复用基于 OO 范型的详细设计,也就是说,在构造系统时,不但要组装可运行的构件,也要能够在一定程度上将构件的 OOD 模型组装成系统的总体模型;二是对于使用 ADL 描述的系统的高层设计,也应该能够进行复用和组装.这就要求在构件的开发和系统的组装过程中,统一使用 ABC 方法来指导并遵循一定的规范.
- **从 SA 描述到 OOD 和具体实现的映射是必须的**.软件工程研究的最终目标是为了提高软件的质量和生产率,若 SA 仅仅局限于对系统的描述和验证,则其作用是有限的.由于当前软件开发的主流是 OO 范型,要想在软件开发中真正发挥 SA 的指导作用,实现一套有自动工具支持的从 SA 到 OOD 和具体实现的映射机制是非常必要的.

SA 到 OOD 的转换现在还没有一个很好的方法,关于这方面的研究有不少,但主要还是集中在如何使用 OO 建模语言,如 UML 等,来描述 SA 结构上.ABC 方法则是希望能够用 OOD 来缩小 SA 设计到实现的距离,保持系统开发各阶段间的可追踪性.所以,从构件组装的特点出发,我们提出了一个两阶段的映射方法来解决这个问题,具体的细节后面将会进一步介绍.

- **有效的工具支持是必要的**.作为一个完整的软件开发方法,ABC 提出了自己的概念、建模语言和开发步骤;而要有效地应用到实际工作中,工具支持是不可或缺的.工具可以屏蔽技术细节,提高系统的可视化程度,减少开发人员的重复性劳动,从而提高开发的效率与质量.

1.2 基本概念和建模语言

作为基于 SA 的软件开发方法,ABC 的基本概念主要有构件(component)、连接子(connector)和体系结构风格(style),同时吸取了面向 Aspect(aspect-oriented)的软件开发的研究成果,引入了 Aspect 这一概念,以更好地描述实际系统.

构件是指系统中较为独立的功能实体.构件模型是面向构件的软件开发方法的核心,是构件的本质特征及构件间关系的抽象描述,它将构件组装所关心的构件类型、构件形态和表示方法加以标准化,使关心和使用构件的外部环境(如使用构件构造出的应用系统、构件组装辅助工具和构件复用者等)能够在一致的概念模型下观察和使用构件.针对不同的需求,不同的方法采用的构件模型也是不同的.文献[14]根据用途将现有的构件模型分为描述/分类模型、规约/组装模型和实现模型.ABC 方法从组装的需要出发,定义构件模型,如图 2 所示.

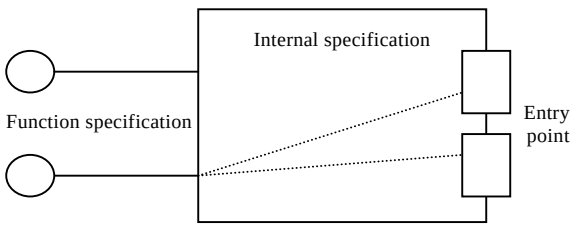

Fig.2 Component model of ABC approach
图 2 ABC 方法中的构件模型

这个构件模型主要分为外部接口(interface specification)和内部规约(internal specification)两部分.

- 外部接口主要描述构件提供给使用者的信息,分为两类:功能规约(function specification),构件供外部使用的接口;接入点(entry point),构件使用到的外部接口.
- 内部规约主要包括构件自身结构的语法约束(例如功能规约和接入点的依赖关系,即要提供某个功能需要哪些外部接口)、语义模型(例如与其他构件交互的协议)以及其他一些服务特性(例如吞吐量等).

同时,构件还可以拥有自己的内部体系结构,这样的构件被称为复合构件.利用复合构件的概念,开发人员

可以逐步精化系统的体系结构模型,更好地进行设计与开发.

连接子是 SA 研究的重要贡献之一,它显式地描述了构件之间的交互关系或交互协议,而在传统的程序设计中,构件间交互的描述往往散布在发生交互的各个构件之中.通过连接子,SA 提供了一种在较高抽象层次观察、设计系统并推理系统行为和性质的方式,也提供了设计和实现可复用性更好的构件、甚至复用连接子的途径.在 ABC 中,连接子模型的定义如图 3 所示.

在 ABC 中,连接子与构件类似,有外部接口和内部规约两个部分,也可以拥有自己的内部结构(具有内部结构的连接子称为复杂连接子).这样可以为设计人员提供更强的抽象描述能力,并且能够将高层的复杂连接子逐步精化到最终的实现上.但是,与构件可以在多个交互中扮演不同的角色相比,连接子只能在一个交互中起作用.这在接口上表现为构件可以有若干个相互无关的接口,而连接子只能有一组相关联的接口.

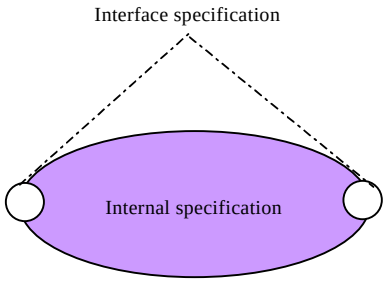

Fig.3 Connector model of ABC approach
图 3 ABC 方法中的连接子模型

体系结构风格是 SA 中另一个重要的概念.它可以从一组特定的构件类型、一组特定的连接子类型、表示构件之间运行时刻联系的拓扑结构以及一组语义约束这 4 个方面加以定义[8].体系结构风格是对 SA 的一种分类,每种体系结构风格表示了一组具有相似特性的软件体系结构,为设计提供了一个统一的术语空间.它的引入有以下优点[15]:可以更为准确和方便地在体系结构的层次上进行交流;可以对不同的体系结构风格设计不同的形式化描述,有利于系统的形式化验证和不同风格之间的比较.另外,不同的风格具有不同的系统特性,通过对体系结构风格的研究,可以更好地使用 SA 来指导软件开发.Bass,Clements 等人在文献[8]中归纳总结了 5 类十几种体系结构风格,并给出了一些如何选择合适的风格的经验规则.ABC 方法允许用户自定义体系结构风格,并在设计中作为构件模板来使用.

Aspect 是指软件系统中一些贯穿全局(cross-cutting)的特性,例如事务、日志等.在现有的软件开发中,这些特性的实现或者对其调用大多分散在系统的各个部分中.这样就容易造成系统结构不清晰,使得软件的开发、维护和升级都比较困难.面向 Aspect 的软件开发的研究试图将这种特性的实现模块化,并与其他功能性的实现体(例如构件)分离开来,使用声明的方式将这些特性插入到系统实现中.这样,使得系统的结构和行为更为清晰和易于理解,从而有助于软件的设计、开发与维护.在基于构件的分布式系统中,构件运行平台集成了一些应用系统所需的公共服务,这些公共服务的实现独立于应用系统,并通过构件平台提供的机制作用于应用系统.这与 Aspect 的概念也有类似性.同时,面向 Aspect 的编程(aspect-oriented programming,简称 AOP)[16]在实际开发中开始得到应用.ABC 方法也引入了 Aspect 这一概念,以更好地描述系统的结构和行为.

在上述基本元素的基础上,ABC 方法定义了自己的建模语言——ABC/ADL[17].作为描述软件体系结构的工具,ADL 是 SA 研究的重点之一.研究者从不同的角度出发,针对不同的目标,提出了多种通用的或专用的 ADL,例如 Wright[18],Rapide[19],Unicon[9]等等.ABC/ADL 设计的目标是在现有的 SA 方法研究的基础上,更好地为系统的开发提供支持.因此,ABC/ADL 定义了一个可扩展的开放语言框架,用户可以根据自己的需要来定义体系结构风格、构件和连接子类型等,更好地描述特定领域的应用系统.限于篇幅,这里不展开对 ABC/ADL 的详细介绍,其具体应用见第 2 节.

ABC/ADL 的语言设施包括 3 个层次:元语言层为用户提供了定义体系结构风格和构件模板、连接子模板等方面的能力;定义层提供了用户用于声明构造系统所需类型的语言设施,这些类型声明必须基于元语言层中已定义的模板;实例层则提供了用户用来声明并连接实例的语言设施,这些实例必须是从定义层中的类型定义实例化而来的.通过区分这 3 个层次,不仅提供了一个易于扩展的语言框架,而且有利于对不同层次的语言设施进行复用.显然,抽象层次越高,可复用性也越高.而且,这还有助于在不同层次上处理系统的语义和约束,例如,类型定义中的连接约束和实例的多对多关系等.

1.3 转换规则、支持工具和构件平台

ABC 方法的一个主要目标是缩短 SA 设计到实现的距离,提高基于构件的软件开发的效率.为此,ABC 方法定义了从 ADL 描述到 UML 的转换规则,提供了根据 SA 模型自动组装基于构件运行平台的应用系统的工具,并开发了自己的构件平台以提供更好的运行支持.

面向对象的设计和语言在现行软件开发中占据主导地位,即使是分布式构件系统,它最终还是要基于面向对象的语言来实现的.因此,如何从 SA 模型转换到 OO 设计或实现,是缩短 SA 和具体实现之间距离的关键.由于 SA 和 OO 描述的侧重点不同,两者之间并没有一个等价的映射方法[20],ABC 方法在两个阶段上建立了从 SA 到 OO 设计语言(现在主要是针对 UML)的转换规则,为构件开发和构件组装提供了支持,提高了系统开发的可追踪性.

首先,在根据 ADL 规约制作原子构件(不包含内部结构的构件)时,将 ADL 规约映射成构件的 OOD 框架,用户在这个框架的基础上进一步精化设计,得到构件的详细 OOD 模型并具体实现.其次,在组装目标系统或制作复合构件时,将已有构件的 OOD 模型合成为目标系统的基本 OOD 模型,作为进一步工作的基础.两个阶段的映射规则分别为:

(1) 从构件规约到 UML 框架.见表 1.

Table 1 The mapping rule from ABC/ADL to UML

表 1 从 ABC/ADL 到 UML 的映射规则

ADL construct	UML construct
Component	Package
Complex connector	Package
Functional specification	Interface
Entry point	Abstract class
Internal specification of component	Note

这里需要注意的是,将接入点映射到抽象类上,这个类在开发构件时并没有真正实现;而是在组装系统的时候,由工具根据构件间的关联关系和底层平台的规范,自动生成调用远端构件的代码,真正实现这个类,并将构件连接在一起.

(2) 从 SA 模型合成 UML 模型.

按照 SA 模型合成系统的 OOD 模型就是将构件和构件之间(或者如果构件之间使用复杂连接子的话,则是构件和复杂连接子之间)的关联映射到类(调用者的接入点类和被调用者实现接口的类)之间的关联(association)关系.进一步地,在关联的基础上可以生成基本的系统协作图(collaboration diagram).

工具支持是 ABC 方法付诸实用的基础.工具对用户屏蔽了 ABC 方法的技术细节,例如 ABC/ADL 语言的详细语法、ABC/ADL 到 UML 的转换等,一方面减轻了用户学习和使用 ABC 方法的负担,另一方面也可以避免用户因疏忽而造成的错误,保证了开发过程的质量.同时,自动组装系统的工作必须由工具来完成.我们已经实现了 ABC 方法的支持工具 ABC Tool 的原型,其主要功能包括:(1) 图形化的 SA 建模:用户可以用直观的图形建模方式生成系统的类型图和配置图;(2) 构件库管理:提供用户管理构件库中可复用构件的能力;(3) SA 模型到 OOD 设计模型的映射,即把应用系统的体系结构模型映射为 OOD 的设计模型;(4) SA 模型到代码的映射,即把应用系统的体系结构模型直接转换成实现代码的框架;(5) 新构件的开发,即支持用户根据自己的需要开发新的构件;(6) 系统语法和语义的一致性检查,保证组装系统的正确性;(7) 构件自动组装,生成可运行于中间件支撑平台的应用系统.

ABC Tool 的系统整体结构如图 4 所示(图中的矩形框表示功能构件,立方体表示各模块的处理对象).

ABC 方法并不特定于某一种构件运行平台进行组装,目前的工具首先支持的是 J2EE 平台.由于 J2EE 平台也存在多种实现,不同实现在生成胶合代码(主要是部署描述文件)时的要求不同,因此 ABC Tool 允许用户针对不同的平台开发不同的组装插件,提高了工具的普适性.同样地,ABC Tool 也可以将组装的范围扩展到其他类型的构件平台上,例如支持 CCM 的 ORB 平台.

现有的构件平台主要关注的还是构件的运行时刻模型和构件间的互操作,对 ABC 方法的支持都还不充分,例如平台服务固定,这样就限制了 Aspect 的效果,又如平台上运行的都是原子构件, 没有保留应用系统结构的

层次关系,失去了 SA 模型中的很多信息,使得系统结构难以理解,对系统维护和演化造成困难.因此,我们设计开发了自己的构件运行平台 PKUAS[21],从平台本身的灵活性、运行时刻信息的保持等方面为 ABC 方法提供了更好的支持.

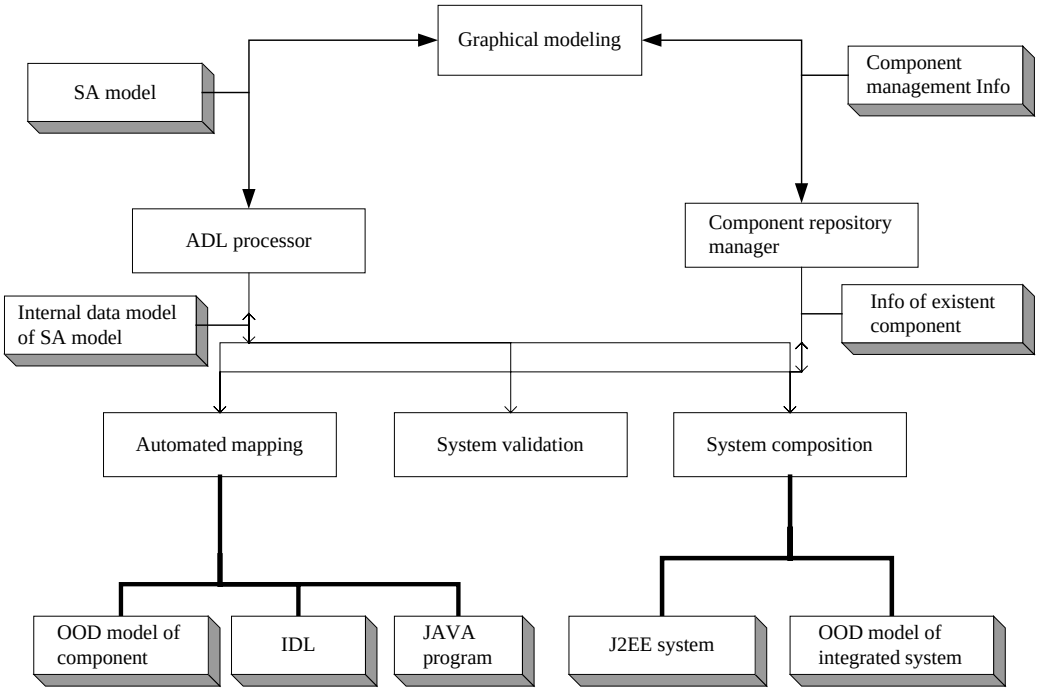

Fig.4 The overall structure of ABC Tool
图 4 ABC Tool 的总体结构

2 ABC 方法指导的软件开发示例

在下面的章节中,我们将结合一个网上订、售火车票系统的例子来介绍如何使用 ABC 方法开发基于 J2EE 平台的分布式构件系统.一个网上订、售火车票系统的基本功能包括:

- 票务信息查询:为用户提供查询列车时刻、票价等信息的功能.
- 实时售票:向用户出售 3 天内的火车票,并立刻付款出票.
- 订票:用户可以订 3 天~15 天之内的火车票,经过一定时间的处理后,票务人员将会通知用户,或者由用户自己查询订票的结果.

2.1 建立SA模型

根据上面的功能分析,我们可以建立起该订、售票系统的体系结构,如图 5 所示.

可以看到,在这个系统中有两个复合构件:界面(UI)与售票代理(ticket agent)以及两个 Aspect:权限控制(authorization & authentication)与事务(transaction).界面可以是一个独立的程序模块,也可以是一组 Web 页面;售票代理则包括了一个负责出票的售票模块(seller)和一个实现网上支付的取款模块(teller).

2.2 构件开发

在系统的体系结构设计完成以后,需要为每个设计构件指定相应的实现构件.ABC 方法的一个前提就是已经存在一个可用的构件库,需要尽量复用构件库中的构件.对于还不存在的构件,则要进行构件的开发,这一工作是在 SA 设计的基础上进行的,以查询模块为例,其 ADL 描述如下:

```
Component QueryAgent is OO.Obj {
Properties{
```

```
    Version=1.0;
}
Interfaces{
    Provide player QueryRequest is Obj.Method{
        Object GetTrainTimeTable (in Object TrainId);
        Collection GetTrains (in String StartPoint, in String EndPoint);
        …
    }
    Request player TicketInfo is Obj.Method{
        TicketInfo GetTicketInfo (in Object TrainId);
        …
    }
    …
}
Dependencies {
    QueryRequest depends on TicketInfo;
    …
}
}
```

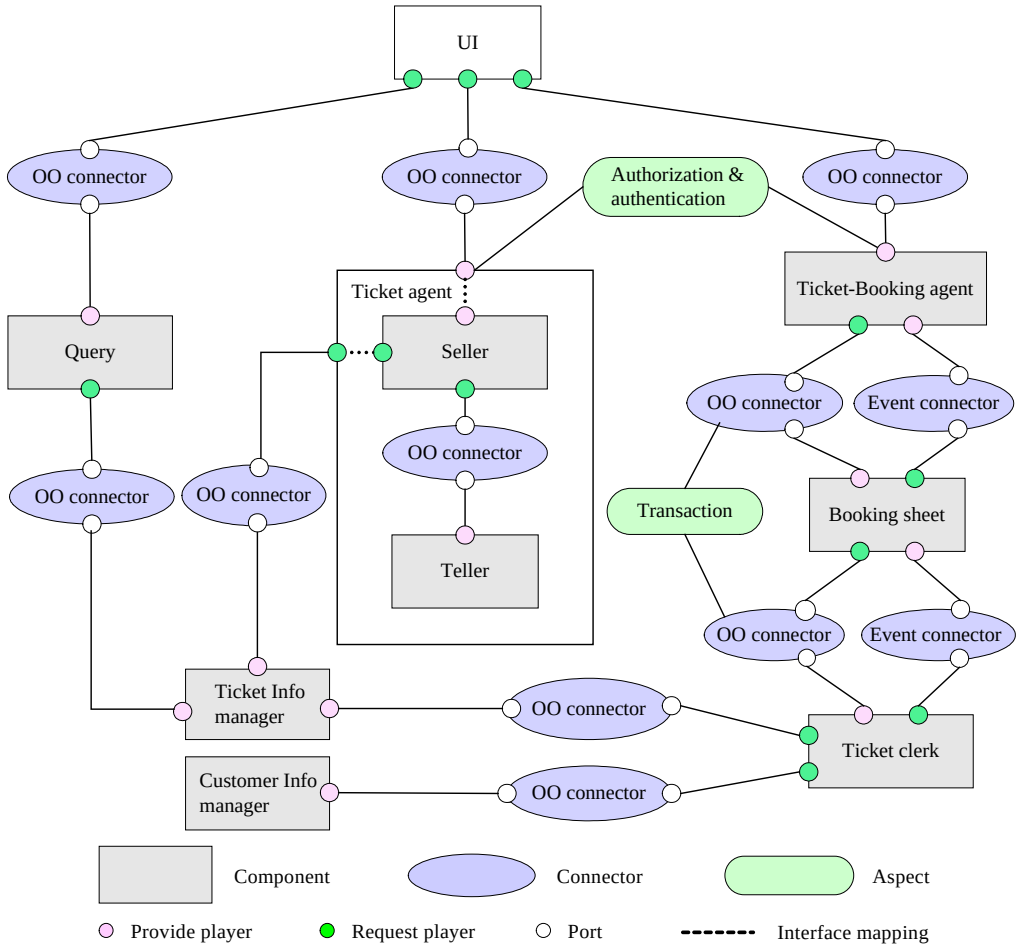

Fig.5 Architecture of the online ticket booking & selling system

图 5 网上订、售票系统的体系结构

工具由该构件描述所生成的 OOD 模型框架如图 6 所示.

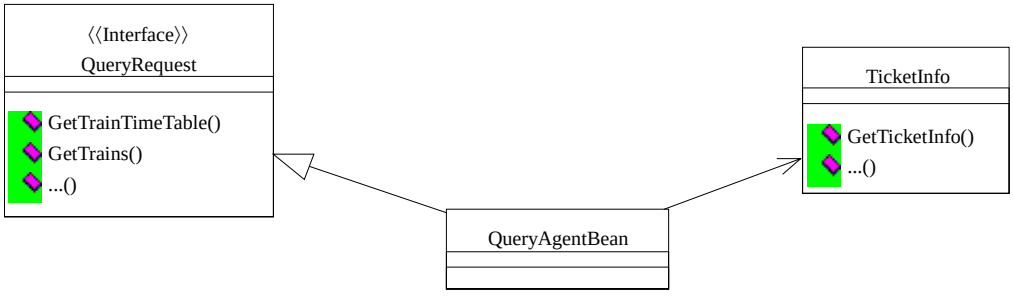

Fig.6 OOD framework of component QueryAgent
图 6 查询模块的 OOD 模型框架

用户在这个框架中进一步进行精化设计,例如,增加需要的辅助类(比如处理时刻表信息的 TimeTable 类等),并最终将其映射到某个运行平台上加以实现.这里以 J2EE 平台为例,得到下面的代码(这里的 Home 和 Remote 接口可以由工具自动生成):

- 使用 EJB 实现的查询模块的 Remote 接口

```
public interface QueryAgent extends EJBObject {
    public TimeTable getTrainTimeTable (Object trainId) throws RemoteException;
    public Collection getTrains(String startPoint, String endPoint) throws RemoteException;
    …
    }
```

- 使用 EJB 实现的查询模块的 Home 接口

```
public interface QueryAgentHome extends EJBHome {
    public QueryAgent create(Object userId) throws CreateException, RemoteException;
        public void remove(Object handle) throws RemoteException;
}
```

- 使用 EJB 实现的查询模块的部分代码

```
public class QueryAgentBean implements QueryRequest, SessionBean {
    String m_userId;
    public void ejbCreate (Object userId){
        m_userId = (String)userId;
        …
    }
    public TimeTable getTrainTimeTable (Object trainId) {
        TicketInfo t_ticketInfo = TicketInfo.getTicketInfo(trained);
        return t_ticketInfo.getTimetable();
    }
    …
}
```

最后,使用 ABC Tool 将得到的构件实现打包,并填写相应的构件信息,放入工具的构件池中,从而将该构件组装到系统中.

2.3 系统检验

用户利用工具,将 SA 模型中的每个设计构件都与构件池中的一个实现构件关联起来,随后就可以开始系

统组装的准备工作.在此之前,需要先用工具对系统进行校验,保证系统的有效性和完备性.系统验证是 SA 研究的一个重点,由于 SA 描述的是软件系统的高层结构,便于人们把握和检验系统的总体特性.这方面的成果有很多,主要的方法是在 ADL 中加入形式化的语义模型,然后使用形式化方法对得到的 SA 模型进行验证.

在 ABC 方法中,对系统的验证包括 3 个层次:一是在语法层次上的校验,主要是对 SA 模型从接口的匹配和构件与连接子的拓扑关系上进行校验;二是在实现层次上,即根据具体实现的语言和底层平台的规范,进行类型匹配等实现级的检查;三是在语义层次上,利用形式化方法描述构件行为和交互协议,对系统的整体特性,比如有效性和完备性(例如是否有死锁,是否有构件之间的失配问题等),进行检查.这些校验都由 ABC Tool 自动完成.

在 ABC 方法中,使用体系结构风格作为语法和语义校验的基本框架,用户可以自定义风格并为该风格制作相应的校验插件,以扩展工具的校验能力.这样,一方面可以有效地利用现有 SA 研究的成果,另一方面也有利于用户形成适合自己领域的描述框架.

2.4 系统生成

系统校验通过以后,就可以根据具体的运行平台来生成应用系统.下面以目前工具支持的 J2EE 平台为例来说明如何组装系统.

J2EE(Java 2 platform enterprise edition)[22]是 SUN 提出的基于 Java 的服务器端运行构件平台,是目前较为成熟也较有潜力的一个构件平台规范.J2EE 通过定义容器(container)、运行时刻构件模型(enterprise Java bean,简称 EJB)以及它们之间交互的规范,并利用 JAVA 的标准服务,如 JNDI 等,使得 EJB 构件具有很好的可移植性和动态交互性.

J2EE 定义的运行环境是通过容器来体现的.EJB 构件在容器中运行并通过容器与其他构件进行交互,容器使用运行时部署描述文件(runtime deployment descriptor)来确定构件间的调用关系以及构件的运行环境;描述文件使用 XML 进行定义.工具在进行构件组装时,主要工作是根据用户定义的 SA 模型,结合已有构件和 Aspect 的信息,自动实现构件接入点的类,并为系统生成相应的描述文件;然后把各个构件和描述文件包装成符合 J2EE 标准的应用程序包,部署到运行环境中去.

以订、售票系统为例,其应用程序包中包含 Interfaces.war,QueryAgent.jar 等实现构件包,它们分别对应于图 4 中的各个构件,而模型中的 Aspect 则映射到构件平台的公共服务(事务、安全)上.按照图 5 的 SA 模型以及 EJB 规范,工具自动生成各个构件的接入点类的实现体,并编译打包.QueryAgent 构件包中接入点类 TicketInfo 的部分实现代码如下:

```
public class TicketInfo {
    TicketInformation m_remote;   //票务信息构件的 remote 接口引用
    static TicketInfo getTicketInfo (Object trainId){
        Context ctx=new InitialContext();
        Object obj=ctx.lookup("java:comp/env/ejb/TicketinfoHome");   //查找票务信息构件
        TicketInformationHome home=(TicketInformationHome) javax.rmi.PortableRemoteObject.narrow(obj,
            TicketInformationHome.class);
        TicketInfo ticketInfo=new TicketInfo();
        ticketInfo.m_remote=home.findByPrimaryKey(trainId);
        return ticketInfo;
    }
    …
}
```

在此基础上,ABC Tool 生成应用程序的部署描述文件,将这些构件连接起来,并插入适当的服务描述.部署描述文件的格式是平台特定的,下面给出的描述文件基于 PKUAS 平台.ABC Tool 允许用户自定义所使用的平

台的文件格式,以支持针对不同平台的系统组装.

```
〈application〉
  〈module〉
        …
        〈Session〉
            〈module-name〉QueryAgent.jar〈/module-name〉
            〈ejb-name〉QueryAgent〈/ejb-name〉
            〈jndi-name〉QueryAgent〈/jndi-name〉
            〈ejb-ref〉  //描述构件之间的调用
                〈ejb-ref-name〉ejb/TicketinfoHome〈/ejb-ref-name〉  //构件中使用到的查询名
                〈jndi-name〉ts/TicketinfoHome〈/jndi-name〉  //实际构件的部署名
            〈/ejb-ref〉
            …
        〈/session〉
        …
  〈/module〉
〈/application〉
```

最后,工具将各个构件包和部署描述文件包装成 TicketSystem.ear,部署到指定的构件平台上,用户即可以开始运行这个应用程序.

3 总 结

软件体系结构和基于构件的软件开发是近年来软件工程界关注的重点.本文从基本概念、建模语言和支持工具等方面介绍了一种将两者结合起来的、基于软件体系结构的、面向构件的软件开发方法——ABC 方法.该方法以 SA 作为系统开发的指导,结合现在较为成熟的 CBSD 方法,将构件运行平台作为组装和运行支持,利用工具支持的自动转换机制,提供了一整套从系统高层设计到最终实现的系统化的解决方案.本文用一个例子说明了如何使用 ABC 方法开发分布式构件系统.

进一步的工作包括:如何从需求分析得到系统高层的 SA 设计.我们拟从两个方面着手,一方面将 SA 的概念引进到问题域空间,另一方面是在领域工程中使用 SA 相关概念来建立领域模型.另外,我们还需要进一步加强 ADL 的语义描述能力,主要是对构件行为和协议的形式化描述以及对系统整体特性的描述,并在此基础上加强系统的验证能力.

References:

[1] Clements PC. From subroutines to subsystems: component-based software development. In: Brown AW, ed. Component-Based Software Engineering: Selected Papers from the Software Engineering Institute. Los Alamitos, CA: IEEE Computer Society Press, 1996. 3~6.

[2] Meyer B, Mingins C. Component-Based development: From buzz to spark. IEEE Computer, 1999,32(7):35~37.

[3] Jacobson I, Booch G, Rumbaugh J. The Unified Software Development Process. Boston, MA: Addison-Wesley, 1999.

[4] D'Souza DF, Wills AC. Objects, Components, and Frameworks: The Catalysis Approach. Boston, MA: Addison-Wesley, 1999.

[5] Garlan D, Shaw M. An introduction to software architecture. In: Ambriola V, Tortora G, eds. Advances in Software Engineering and Knowledge Engineering, Volume 1. New Jersey: World Scientific Publishing, Co., 1993.

[6] Allen R, Garlan, D. Formalizing architectural connection. In: Proceedings of the 16th International Conference on Software Engineering. Los Alamitos, CA: IEEE Computer Society, 1994. 71~80. http://acm.lib.tsinghua.edu.cn/acm/main.nsp?view=ACM.

[7] Shaw M. The coming-of-age of software architecture research. In: Proceedings of the 23rd International Conference on Software Engineering. Washington, DC: IEEE Computer Society, 2001. 657~664. http://acm.lib.tsinghua.edu.cn/acm/main.nsp?view=ACM.

[8] Bass L, Clements P, Kazman R. Software Architecture in Practice. Boston, MA: Addison-Wesley, 1998.

[9] Shaw M, DeLine R, Klein D, Ross T, Young D, Zelesnik G. Abstractions for software architecture and tools to support them. IEEE Transactions on Software Engineering, 1995,21(4):314~335.

[10] Mei H, Chang JC, Yang FQ. Software component composition based on ADL and middleware. Science in China (F), 2001,44(2): 136~151.

[11] Clements P, Northrop L. Software architecture: an executive overview. Technical Report, CMU/SEI-96-TR-003, Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.

[12] Aldrich J, Chambers C, Notkin D. ArchJava: connecting software architecture to implementation. In: Proceedings of the 24th International Conference on Software Engineering. IEEE Computer Society, 2002. 187~197.

[13] McDirmid S, Flatt M, Hsieh W. Jiazzi: New-Age components for old-fashioned Java. ACM SIGPLAN Notices, 2001,36(11): 211~222.

[14] Mei H. A component model for perspective management of enterprise software reuse. Annals of Software Engineering, 2001,11(1): 219~236.

[15] Abowd G, Allen R, Garlan D. Using style to understand descriptions of software architecture. Software Engineering Notes, 1993, 18(5):9~20.

[16] Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes CV, Loingtier JM, Irwin J. Aspect-Oriented programming. In: Proceedings of the European Conference on Object-Oriented Programming (ECOOP). LNCS 1241, Springer-Verlag, 1997. 220~242. http://citeseer. nj.nec.com/63210.html.

[17] Mei H, Chen F, Wang QX, Feng YD. ABC/ADL: an ADL supporting component composition. In: George C, Miao HK, eds. Proceedings of the 4th International Conference on Formal Engineering Methods. New York: Springer-Verlag, 2002. 38~47.

[18] Allen R, Garlan D. A formal basis for architectural connection. ACM Transactions on Software Engineering and Methodology, 1997,6(3):213~249.

[19] Luckham DC, Vera J. An event-based architecture definition language. IEEE Transactions on Software Engineering, 1995,21(9): 717~734.

[20] Garlan D, Kompanek AJ. Reconciling the needs of architectural description with object-modeling notations. In: Proceedings of the UML 2000. New York: Springer-Verlag, 2000. 498~512. http://citeseer.nj.nec.com/63210.html.

[21] Huang G, Wang QX, Cao DG, Mei H. PKUAS: a domain-oriented component operating platform. Chinese Journal of Electronics, 2002,30(12A):1938~1942 (in Chinese with English abstract).

[22] SUN Microsystem, Java 2 Platform Enterprise Edition Specification, Version 1.3. Proposed Final Draft 4, 2001. http://java.sun.com.

附中文参考文献:

[21] 黄罡,王千祥,曹东刚,梅宏.PKUAS:一种面向领域的构件运行支撑平台.电子学报,2002,30(12A):1938~1942.