论 文

ABC: 一种全生命周期软件体系结构建模方法

梅宏^{①②*}, 黄罡^{①②}, 张路^{①②}, 张伟^{①②}^① 高可信软件技术教育部重点实验室 (北京大学), 北京 100871^② 北京大学信息科学技术学院软件研究所, 北京 100871

* 通信作者. E-mail: meih@pku.edu.cn

收稿日期: 2013–12–19; 接受日期: 2014–03–04

摘要 随着计算机硬件能力的快速增长和软件应用规模的不断扩大, 软件的复杂性也在持续增长, 并始终制约着软件开发效率和质量的有效提升. 软件的结构复杂性, 尤其是高层结构的复杂性, 是软件复杂性的一种重要表现. 如何实现对软件高层结构复杂性的有效控制, 是当前开放、动态、难控的网络环境下大规模软件系统开发与演化所面临的主要问题. 针对这个问题, 我们将设计阶段高层结构复杂性的控制模型—软件体系结构模型—扩展到整个软件生命周期, 提出了一种以体系结构为中心的软件开发方法—ABC. 该方法将软件生命周期各阶段的核心制品与活动, 统一到软件体系结构模型及其连续迭代的细化、映射和转换, 实现对软件高层结构复杂性的一致、灵活、系统化的建模和管理. 本文旨在系统性地总结 ABC 方法在软件体系结构建模方面的成果, 并重点介绍近几年在协同式特征建模、运行时体系结构生成、体系结构逆向恢复与建模等方面取得的新进展.

关键词 软件复杂性 软件体系结构 特征模型 运行时模型 程序分析 网构软件

1 引言

软件的开发效率和质量难以满足社会经济发展对软件的实际需求是导致软件危机的直接原因, 其深层原因是软件技术的发展无法有效应对软件规模和复杂性的快速增长.

软件的规模通常使用代码行来度量. 在目前阶段, 大规模软件的代码行在千万到亿的数量级上, 并呈现持续增长的趋势. 例如, Windows 操作系统的规模从初期 (1995 年) 的 1500 万行逐渐增长至 2007 年的 6000 万行 (图 1); 2011 年宝马 7 系轿车中软件的规模达到 2 亿多行; 空客 A380 飞机中软件的规模甚至超过 10 亿行. 导致软件规模持续增长的主要原因在于软件应用范围的不断扩大和应用程度的不断加深: 软件必须具备更多、更强、更复杂的能力, 以应对更加复杂的应用环境.

软件复杂性的成因可以从以下几个方面进行理解. 第一, 软件的复杂性在一定程度上源于现实世界固有的复杂性: 大多数软件是对现实世界的反映和模拟, 而现实世界的很多事物经过了长期的发展变化, 形成了错综复杂的关系, 因而导致反映和模拟这些事物的软件也需要承载这些已经存在的复杂关系. 第二, 软件语言和自然语言之间的差异性也是导致软件复杂性的一个重要原因: 由于软件语言和自然语言之间在语法、语义和语用上存在很大的差异, 使用软件语言对现实世界的问题域进行描述会进一步增加软件制品的复杂性. 第三, 作为一种知识制品, 软件在结构上呈现出分形的特点, 即: 从任何一个抽象层次看, 软件的各个组成部分都呈现出相似但又不完全相同的特点. 因此, 对于动辄具有

引用格式: 梅宏, 黄罡, 张路, 等. ABC: 一种全生命周期软件体系结构建模方法. 中国科学: 信息科学, 2014, 44: 564–587, doi: 10.1360/N112013-00042

图 1 Windows 操作系统规模的不断增长

Figure 1 The scale increasing of Windows operation systems

图 2 (网络版彩图) 软件复杂性示例: 某企业信息系统软件模块关系 (a) 和微软 Web 服务器显示一张图片的函数调用序列 (b)

Figure 2 (Color online) Examples of software complexity: the software module relation map of an enterprise's information system (a) and an invoking sequence to display a picture in Microsoft web server software (b)

数百万、上千万行代码的软件而言, 其复杂性会随着软件规模的增大呈现非线性的增长, 且不同规模的软件所蕴含的复杂性及其控制各不相同. 第四, 软件的持续演化也会导致软件复杂性的不断增长: 用户需求和运行环境的变化往往需要对软件进行适应性调整, 而不论是对可预期的变化进行主动调整, 还是对非预期的变化进行被动适应, 都会进一步导致软件复杂性的增长. 软件复杂性的示例如图 2 所示.

当软件的规模和复杂性超出开发者和技术所能掌控的范围, 就会使得软件在开发效率、成本及质量上无法满足实际需求, 进而导致软件危机的产生. 软件工程, 作为应对软件危机问题而产生的一个研究和实践领域, 其核心目标即是对软件复杂性的有效建模与控制. 考察软件工程中建模和控制软件复杂性的各种原则、方法和技术, 其基本思想可以归纳为分治 (separation of concerns, 或称为“分而治之”). 分治的内涵包括 3 个方面 (如图 3 所示). 第一, 分治表现为“分解”, 即: 将一个大规模或较大规模的软件分解为多个组成部分, 然后分别对每个组成部分的复杂性进行建模和控制, 进而实现对整个软件系统复杂性的控制. 第二, 分治表现为“分层” (或称为“抽象”), 即: 将高层结构与实现细节

图 3 分治的 3 个方面

Figure 3 The three aspects of separation of concerns

相分离,使得开发人员进行开发活动时集中关注于特定的抽象层次。其三,分治表现为“分步”,即:将整个软件生命周期划分为需求、分析、设计、实现、测试、部署、维护、演化等一系列阶段或步骤,并分由不同类型的开发人员进行具体实施。

软件复杂性具有多种不同的表现形式^[1]。如,结构复杂性、算法复杂性、计算复杂性、数据复杂性、功能复杂性等。其中,结构复杂性是影响软件开发效率和质量的主要因素^[2]。

考察软件技术的发展历史,可以发现,如何有效控制结构复杂性一直是程序模型及编程语言领域的研究主线。最早的软件使用机器语言或汇编语言编写,构成软件的基本单元是指令,控制机制主要是顺序和转移。高级语言的出现使得开发人员可以在更高抽象层次上进行程序设计,但转移语句(goto)的滥用也会导致软件结构的混乱。结构化编程的出现使得任意计算流程均可使用顺序、分支和循环这三种基本控制结构的组合来描述,从而消除了转移语句滥用带来的问题。函数、过程等程序元素的出现为信息的封装提供了一种更大粒度的抽象机制,进而可以构造出更大规模的软件。面向对象的编程则进一步提供了一种以对象为基本计算单元,以消息传递为基本交互手段的软件模型:该模型以拟人化的观点来看待客观世界,既符合人的思维模式,又符合客观世界的构成规律。从复杂性控制的角度看,程序模型及编程语言技术的发展呈现出抽象层次不断提高、更加符合人类思维模式这两种趋势,从而使得开发人员能够更方便地构造更大规模和更复杂的软件。

然而,随着软件规模和复杂性的持续增长,人们逐渐发现,仅在编程层次对软件的结构复杂性进行控制还远远无法满足大规模复杂软件开发的实际需求:与编程层次的结构复杂性相比,软件高层结构的复杂性对大规模复杂软件的开发效率和质量具有更为重要的影响。上世纪90年代初,软件工程界借鉴建筑学中体系结构的思想,提出了软件体系结构(Software Architecture, SA)的概念,并发展了相应的软件开发方法。在概念上,软件体系结构被定义为一组软件构件、构件之间的连接子、以及作用于构件和连接子之上的约束和配置;相应的开发方法将软件体系结构作为软件设计阶段的核心制品,致力于利用软件体系结构来建模设计阶段软件高层结构的复杂性,并通过相应的分析与设计方法对这种复杂性进行控制^[3]。

软件工程的已有实践表明,没有一种单一的技术可以完美地应对各种形式的软件复杂性^[4]。虽然软件体系结构在控制软件复杂性方面展现出特有的优势,但仅在设计阶段建模和控制软件高层结构的复杂性,其有效性和实用性仍存在较大的局限:软件生命周期的其它阶段也存在类似的高层结构复杂性,这些结构复杂性也会对软件的开发效率和质量产生直接的影响。特别地,在需求阶段,由于环境的开放和不确定性,其结构复杂性不亚于设计阶段;在运行阶段,由于动态性和并发性的出现,也使得其结构复杂性远大于设计阶段。

基于以上分析,我们提出了以体系结构为中心的软件开发方法 ABC(Architecture Based Compo-

图 4 ABC 方法研究思路

Figure 4 The basic ideas of ABC method

ment Composition) 来控制软件生命周期各个阶段的高层结构复杂性, 并通过基于构件的软件复用来提高软件的开发效率和质量. 如图 4 所示, ABC 方法的核心是全生命周期软件体系结构建模, 即: 将软件生命周期各阶段的核心制品与活动, 统一为软件体系结构模型及对其连续迭代的细化、映射和转换, 从而实现对软件高层结构复杂性一致、灵活、系统化的管理和控制.

本文的组织结构如下: 第 2 节概要介绍 ABC 方法并总结其在全生命周期软件体系结构建模方面的主要成果; 随后几节分别介绍近几年以互联网软件为研究对象, 在协同式特征建模、运行时体系结构生成和体系结构逆向建模 3 个方面的最新进展; 最后总结全文并展望未来工作.

2 ABC 方法概览

ABC 方法于 1999 年萌芽^[5], 2003 年形成雏形^[6], 2006 年基本成型^[7], 此后又经过不断地改进和完善. ABC 方法的核心思想是将软件体系结构引入软件开发的各个阶段, 作为系统开发的蓝图, 并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, 然后在构件运行平台 (即软件中间件) 的支持下实现自动的系统组装生成.

ABC 方法过程模型如图 5 所示, 其中:

(1) 需求分析阶段的软件体系结构及其建模: ABC 在需求分析阶段引入了软件体系结构的概念, 以结构化的方式来组织问题空间和用户需求. 在此阶段, 软件需求以特征 (Feature) 的形式表示, 需求之间的关系则由特征之间的关系来刻画, 即: 以特征作为建模需求的一阶实体, 通过显式描述特征之间的静态和动态依赖关系实现对需求的有效分解和组织, 形成被称为特征模型的分析模型. 为了有效地刻画需求之间的关系, 特征模型刻画了特征之间的 4 种重要关系: 精化、约束、影响和交互, 分别描述特征之间的静态和动态依赖关系. 为了支持需求层次的复用, ABC 在特征模型中引入了变化性的表示机制, 以描述一类相似或相近的软件系统的共性和变化性需求: 通过对特征模型的定制, 产生适应于特定软件应用的需求模型, 并利用特征之间的约束关系来保证软件应用需求模型的完整性和一致性. 在此基础上, ABC 的特征建模方法通过对软件责任 (Responsibility) 的识别以及对特征之间交互

图 5 全生命周期软件体系结构建模过程

Figure 5 The process of whole lifecycle software architecture modeling

关系的分析,可以半自动地导出软件系统的高层体系结构,作为后续设计、组装、部署、以及维护与演化的基础。

(2) 设计阶段的软件体系结构及其建模: ABC 在设计阶段主要进行软件体系结构建模。在此阶段, ABC 通过分析软件系统的需求规约, 制定相应全局设计决策, 创建必需的构件和连接器, 建立静态和动态的软件体系结构模型 (包括类型图、实例图和过程图等), 并进一步精化特征模型与软件体系结构的追踪关系。为了更好地通过复用提高目标软件系统的开发效率和质量, ABC 还提供了辅助设计者从资产池中选取可复用的构件和连接子的能力。从兼容性角度考虑, ABC 并不排斥其它的软件开发范型, 如面向对象分析与设计产生的高层设计或概要设计也可视为一种软件体系结构: 只要将这种面向对象的软件体系结构中的基本组成元素从对象封装为构件 (如, 按照对象来源、功能类别、用况、通信频繁程度、并发和分布情况、一般 — 特殊结构、整体 — 部分结构、关联、包的关系密集程度等准则, 可以将一组对象封装为一个构件), 就可以将这种构件化的软件体系结构直接作为本阶段的设计结果。

(3) 实现阶段的软件体系结构及其建模: ABC 的目标系统主要通过组装可复用构件来实现, 即: 根据设计阶段产生的软件体系结构模型, 选取、鉴定、适配资产池中已有的构件实现体, 在通过一致性校验后, 组装成可发布的软件系统。如果在资产池中找不到合用的构件, ABC 可将构件描述转换成 UML 模型或 C++ 和 Java 代码框架, 由开发者完成构件制作并将之存入资产池, 最终组装到目标系统。

(4) 部署阶段的软件体系结构及其建模: 基于构件的软件系统往往运行于特定的构件运行支撑平台, 如 CORBA/CCM(Common Object Request Broker Architecture/CORBA Component Model) 平台、J2EE/EJB(Java 2 Platform Enterprise Edition/Enterprise JavaBeans) 平台、.NET/COM(Component Object Model) 平台或 Web Services 平台等。这些软件系统必须经过部署才能正常运行, 而部署所需的信息较为繁杂, 往往需要部署人员手工填写。实践发现, 大部分部署信息在系统设计与实现阶段已经存在, 只需经过必要的转换或融合即可对其进行复用。为此, ABC 定义了软件体系结构部署模型 (其主要信息由设计模型与实现模型导出), 支持以直观的图形方式添加少量的信息, 并实时显示目标环境的资源与负载情况, 从而实现自动化的部署。

(5) 维护与演化阶段的软件体系结构及其建模: 从某种意义上来说, ABC 方法可以看成对软件体系结构模型连续、迭代的细化、映射和转换, 在每一次细化和转换后, 软件体系结构的语法和语义信息变得更加精确与完整。到了维护与演化阶段, 运行时模型刻画了系统在运行时刻的实际状态, 因而具有系统最精确和完整的信息, 该模型被称为运行时软件体系结构 (Runtime Software Architecture, RSA)。基于运行平台的管理机制, RSA 可以实时地反映出系统运行时刻的真实状态, 并且, 直接操作 RSA 就可以实现对软件系统的在线维护与演化。

图 6 ABC 特征模型元模型

Figure 6 The meta-model of ABC feature model

为了支持上述过程, ABC 方法实现了特征建模工具、软件体系结构建模工具、元建模工具、构件代码框架生成工具、服务组装工具、构件库管理系统、以及作为构件运行支撑平台的软件中间件等一系列的工具和平台. ABC 方法和工具已在北京 2008 奥运会信息系统建模、某军队信息化系统设计与建模、某商业银行信贷风险管理系统开发等项目中得到验证性应用.

需要指出的是, 软件体系结构作为软件设计阶段的核心制品, 其在设计、实现、甚至部署阶段的意义和作用已有不少研究. 例如, 美国卡耐基梅隆大学的 David Garlan 教授、美国加州大学埃尔文分校的 Richard Taylor 教授、英国帝国理工的 Jeff Krama 教授等领导的团队近 20 年来一直致力于软件体系结构的研究. 与之相比, 从软件复杂性控制的角度看, ABC 方法的特色在于: 1) 在分解方面, 强调软件体系结构的基本成分是“构件”, 而不是粒度更小且常常在代码级别进行复杂性控制的“对象”, 并且将构件之间的连接也作为与构件同等地位的一阶实体 (first class entity); 2) 在分层方面, 强调通过约束和连接器对运行支撑平台进行体系结构级别的抽象, 以充分利用运行支撑平台的能力来实现性能、可靠、安全等系统级质量需求; 3) 在分步方面, 强调在同一概念框架下针对需求分析阶段和维护演化阶段存在的高层结构复杂性, 通过将软件体系结构引入这两个阶段, 揭示了这些复杂性之间的关系, 并通过相应的建模方法实现复杂性控制.

2.1 面向特征的需求建模

在“将软件体系结构引入需求阶段”思想的指导下, ABC 方法采用特征模型作为组织软件需求的结构化模型, 对特征模型的结构、语义、验证方法、以及基于特征模型的软件体系结构设计等问题进行了研究.

(1) 特征模型元模型

特征模型可以抽象为一组特征以及特征之间的一组依赖关系, 如图 6 所示. 在定义特征模型结构的过程中, ABC 方法主要关注两个问题^[8]: 为了实现对需求的共性/变化性建模、需求复用、及软件设计的有效支持, 特征模型应该捕获特征之间何种类型的依赖关系, 以及如何在特征建模过程中有效地识别出这些依赖关系?

ABC 方法定义的特征模型包含特征和特征之间的 4 种依赖关系^[9]. 特征是一种具有用户/客户价值的软件特点, 其内涵上指代了由一组具有紧密关联的单个需求构成的集合. 特征之间的依赖关系包括: 精化 (Refinement)、约束 (Constraint)、影响 (Influence) 和交互 (Interaction). 精化关系提供了组织特征的基本方式: 通过精化关系, 不同粒度/抽象层次的特征被组织成层次结构. 约束刻画了特征在绑定状态上的依赖关系: 一方面, 约束关系提供了一种对需求的变化性进行限制的机制; 另一方面, 约束关系也为保证特征模型定制结果的完整性和一致性提供了判断依据. 影响和交互分别刻画了特征在软件规约层次上的静态和动态依赖关系: 这两种关系主要用于进行基于特征模型的软件体系结构设计.

ABC 方法定义的特征模型还关注了不同依赖关系之间的关联. 依赖关系之间存在关联的主要原因在于: 不同类型的特征依赖关系之间并非完全独立, 而往往是从不同视角对同一对象进行建模的结果. 例如, 精化和约束是存在于问题空间的两种依赖关系, 影响和交互是存在于解空间的两种依赖关系; 影响和交互分别从静态和动态的视角对特征在软件规约层次上的依赖关系进行了刻画. ABC 方法特征模型所关注的依赖关联包括: 由精化导致的约束之间、约束与交互之间的依赖关联, 以及影响与交互之间的依赖关系. 通过依赖关联, 可以从一种已识别的依赖关系快速识别出其他类型的依赖关系, 从而提高依赖关系分析活动的效率. 依赖关联的建模在客观上也能够提高特征模型的易维护性: 当一种依赖关系发生变化后, 可以通过关联对与其相关的其他依赖关系做出相应的调整, 从而保证特征模型包含信息的一致性.

(2) 可定制的需求建模机制

特征模型提供了一种具有良好可定制性的需求建模机制. 传统的需求建模机制由于缺乏对可定制性的显式考虑, 不能有效地支持从需求建模结果中抽取出一组相对独立的需求子集, 从而导致这些建模机制无法满足基于定制的需求复用活动的要求. 为了提高需求的可定制性, 需要明确的一个基本问题是: 需求定制的基本单元是什么? 针对该问题, ABC 方法认为, 将软件对外表现出的特征作为需求定制的基本单元, 并采用面向特征的思想对需求进行建模, 是提高需求可定制性的一种有效手段^[10]. 基于特征和特征模型的概念, ABC 方法将传统的缺乏可定制性的软件需求组织方式转变为一种面向特征的树形组织方式, 并显式建模树形结构中节点 (特征) 之间的约束关系. 采用这种建模机制, 可以将特定应用领域的需求建模为相应的特征模型, 并通过对特征模型的定制对其中包含的需求进行复用: 对于特征模型的任何定制结果 (即: 特征模型所包含特征的一个子集), 只要不破坏特征之间的约束关系, 即可被认为是一个合法的需求子集. 这种方式为解决需求复用问题提供了一条现实可行的途径.

(3) 基于命题逻辑的特征模型形式化机制与验证方法

针对特征模型的语义问题和自动验证问题, ABC 方法提出了将特征模型转换为相应的命题逻辑表示的完整方案, 并以此为基础提出了验证特征模型及其定制结果的 3 条形式化准则^[11]: 特征模型的可满足性、特征的可绑定性以及可删除性. 这 3 条验证准则能够完整、准确地检测出特征模型及其定制结果中存在的异常和不一致这两种类型的缺陷.

针对验证优化问题, ABC 方法发现了特征模型中存在的原子集合、伪可选特征、及验证无关特征等 3 种现象^[11,12]: 原子集合包含了一组在特征模型定制过程中始终具有相同绑定状态的特征; 伪可选特征在语法上表现为可选特征, 在语义上则体现为死特征或必选特征; 验证无关特征不会对特征模型及其定制结果的验证带来任何影响. 在此基础上, ABC 方法提出通过合并原子集合特征和消除验证无关特征, 缩减特征模型验证问题中变量数目的方法. 这种验证优化方法可作为一种预处理技术, 与传统的特征模型验证算法相结合, 从而提高验证算法的规模可扩展性.

图 7 从特征模型向软件体系结构模型的变换

Figure 7 Transformation from feature model to software architecture model

(4) 特征模型到设计阶段体系结构的变换方法

针对特征模型到软件体系结构的变换问题, ABC 方法提出了一种以特征操作化 (Feature Operationalization) 和责任分配 (Responsibility Assignment) 为手段的顶层软件体系结构设计方法^[13], 如图 7 所示. 特征操作化旨在将特征所指代的需求操作化为可编程实现的软件规约, 并使用责任对软件规约进行模块化: 通过操作化, 一个特征的实现被具体化为一组责任及其之间存在的交互. 责任分配旨在将由特征操作化产生的责任分配至构件: 通过责任分配, 同一特征操作化产生的责任可能会被分配至不同的构件, 从而导致责任之间的交互转变为构件之间的交互; 不同特征操作化产生的责任则有可能会分配至同一个构件, 从而导致不同特征在实现层次上的交织. 通过特征操作化和责任分配, 即可产生特征模型对应的高层软件体系结构, 即: 一组构件以及构件之间的交互关系. 该变换方法对基于特征模型的软件设计的实施提供了一个系统化的操作指南, 并能在设计产生软件体系结构的同时建立需求阶段与设计阶段制品之间的追踪关系, 从而为软件的维护和演化积累必要的信息资源.

2.2 设计阶段的软件体系结构建模

ABC 方法中设计阶段体系结构模型的主要概念和建模元素包括: 构件和构件类型、连接子和连接子类型、配置、约束、以及设计决策 (如图 8 所示). ABC 设计阶段体系结构具有典型的软件体系结构模型的所有要素: 完成计算功能的构件、保障构件之间交互的连接子、对交互拓扑结构进行描述的系统配置以及从系统全局角度出发需要满足的各种约束.

在此基础上, ABC 设计阶段体系结构模型还具有以下特点:

(1) 区分类型和实例

构件和连接子是 ABC 体系结构的基本组成元素, 其中: 构件主要负责进行业务层面的计算, 连接子主要对构件之间的交互进行抽象. 对构件和连接子的显式区分是区别体系结构描述语言和其他建模语言的一个重要因素. 构件和连接子的内在特性是由其类型决定的, 即: 一个构件的类型决定了该构

图 8 ABC 体系结构模型概览

Figure 8 An overview of ABC software architecture model

件的计算能力, 一个连接子的类型决定了该连接子的通信能力. 区分类型和实例具有两个优点: 一, 提高模型的可验证性; 二, 提高构件类型和连接子类型的可复用性.

(2) 兼容复用和构造

构件类型和连接子类型是潜在的可复用资产 (Asset), 有可能在不同的软件体系结构模型中进行复用. 与可复用资产相对的即是设计时新构造的资产. 虽然新构造的资产在设计完毕也可能成为可复用资产, 但对某一软件体系结构模型而言, 新构造的资产与可复用资产具有一个本质的区别: 前者可以根据需要随时进行修改, 而后者则是一个不可修改或可有限修改的黑盒实体 (COTS). 这种区别对构件的组装会产生截然不同的影响: 对已存在的可复用构件进行组装有可能产生失配的问题, 而对新构件的组装则不会产生失配问题 (如果发现有不适合的地方, 可以随时对新构件进行必要的修改). 鉴于可复用资产和新构造资产的上述区别, ABC 体系结构模型对这两种类型的资产进行了显式的区分. 但对设计者而言, 无论是哪一类资产, 它们均符合 ABC 构件模型和连接子模型.

(3) 简洁有效的配置

配置 (Configuration) 决定了构件之间的交互拓扑结构. ABC 体系结构模型是面向构件组装的体系结构模型, 因此它采用了一种简洁直观的方式来描述构件之间的组装方式, 即: 用松耦合的 Link 来确定构件的交互方式. Link 的语义可类比为演员承担角色. 演员是对构件计算功能的抽象, 角色则是对连接子通信能力的抽象. 通过恰当的 Link 定义, 设计者就能依据目标系统的需求指定系统中的每个构件应该承担的角色. 因此, ABC 体系结构模型中的配置由一系列 Link 组成.

(4) 可扩展的约束

软件体系结构对复杂性建模和控制的一个主要方面在于: 它允许设计者在系统开发早期对系统的整体性质进行分析, 以便尽早发现潜在的设计问题. 支持这种分析的一个重要基础是体系结构模型对约束 (Constraint) 的建模 (如: 安全、事务、响应时间等非功能性约束的建模). ABC 体系结构模型提供了灵活的约束描述方式, 允许设计者根据特定的非功能约束扩展已有的描述方式, 以便为基于体系结构的系统分析提供必要的约束信息^[14].

(5) 支持设计决策

体系结构设计决策 (Architectural Design Decision) 对解决软件开发和演化中的“知识蒸发”、“设

计腐化”等问题具有重要作用. 但相关的研究主要关注于设计决策及其理由的表示, 而缺乏对设计决策与体系结构设计方法集成的有效支持. 由于这种缺乏, 对决策与理由的建模往往不会对软件设计产生直接的指导或帮助, 因此也很难应用到软件工程实践中. 针对这种问题, ABC 通过问题、方案、决策、理由等 4 个概念, 采用 3 种建模原则对设计决策进行建模^[15]. 一、决策抽象原则: 从决策的视角认识体系结构, 并将体系结构设计限定于对系统高层和全局关键问题的决策, 忽略细节性的或能够在后续开发中解决的问题. 二、问题分解原则: 将体系结构设计作为对一组体系结构层次关键问题的求解; 因此, 从全局设计目标到这些问题的分解是体系结构设计的首要步骤, 而设计的实现则依赖于对这些问题的解决方案的合成. 三、知识复用原则: 将软件设计相关知识而不是特定的设计制品作为复用的对象; 实现复用的方式则是利用已有的知识解决新的体系结构设计问题.

2.3 实现阶段的软件体系结构建模

ABC 方法强调基于构件组装的软件复用. 也就是说, 体系结构设计中的构件、甚至连接子都可通过复用已有的软件制品来实现. 因此, 在实现阶段, 软件体系结构建模的核心就是构件模型和连接子模型.

(1) 构件模型

ABC 构件模型从构件使用者的角度, 定义了一个面向组装的软件模型. 它在业务层面上对各种不同实现类型的软件实体进行抽象, 使得各种软件实体能够通过统一的规约, 在业务层面上进行交互 (在实现层面上的交互依赖于模型之间转换规则的设计和特定运行平台的支持). ABC 构件模型具有两种类型的接口: 对外提供的接口和需要外部提供的接口. 对这两种接口的显式建模使得构件成为一个相对独立的软件实体. 在这两种接口中, 对外提供的接口对构件模型而言是必须的, 而是否需要外部提供的接口则取决于构件的实现方式.

ABC 构件模型还可以包含内部结构. 包含内部结构的部件称为复合构件. 与之相对的其他构件称为原子构件. 复合构件的内部结构可以具有独立的描述方式. 但一般而言, 设计者大多将复合构件作为一个子系统看待, 其内部结构则被看作是该子系统的体系结构. 因此, 可以直接采用 ABC 体系结构模型对复合构件的内部结构进行建模 (见图 8). 由于复合构件的内部结构中还可能包含复合构件, 因此, 在一般意义上, 这种复合结构可以包含任意层数的嵌套, 从而支持设计者对一个复杂构件进行不断的精化. 采用这种方式定义复合构件也意味着 ABC 体系结构可以将自身封装为一个更大粒度的构件, 供其他应用使用. 同时, 构件的多层次嵌套也在一定程度上体现出知识制品具有的分形复杂性以及基于分层的复杂性控制手段.

(2) 连接子模型

连接子是对构件之间交互的抽象. 对大型复杂系统而言, 构件之间的交互往往对系统的整体性质有重大影响. 绝大多数软件体系结构模型都把连接子看作是一种与构件具有同等重要性的一阶实体. 与构件模型类似, ABC 连接子模型也是一种面向黑盒构件组装的模型. 连接子的设计者仅仅对构件的外部可见特性进行抽象, 因此在设计连接子内部结构时就不能对参与构件的实现细节做出任何假定. ABC 连接子模型主要由连接子的外部规约和内部规约两部分组成.

外部规约定义了参与交互的构件承担的各种角色. 一般而言, 构件的角色可以分为两类: 一类是交互的发起者, 另一类是交互的响应者. 每种角色可以分别定义具体参与交互的构件的类型, 常见的有方法、事件、消息等; 此外还可以为角色定义多重性, 即: 可以允许多多少个实例参与交互. 只有满足角色要求的构件才能正确的进行交互.

图 9 基于软件体系结构的应用部署过程

Figure 9 Software architecture based application deployment

内部规约定义了连接子对构件交互的处理细节, 主要包括通信链路、时序编排和监控设施三部分 [16]:

- 通信链路是对进行通信的构件对的抽象, 每个通信链路都有各自所使用的通信协议, 常见的包括过程调用、远程过程调用、消息传递等. 通信链路的定义是必须的, 是构件交互得以顺利进行的基础.
- 对于复杂的构件交互, ABC 连接子模型还允许设计人员定义构件之间的交互时序, 即: 多个构件交互顺序、并发等信息. 构件交互时序的定义有助于设计者在体系结构层次对系统的行为进行分析和验证, 从而判断系统结构是否满足特定的质量属性.
- ABC 连接子还可以对构件交互进行监控, 即: 在构件交互前或者交互后进行特定的处理. 这也体现了将构件和连接子分离的好处: 可以以一种松耦合的方式对交互进行特定的处理. 这不仅有利于系统的维护, 还有助于相关处理机制的复用. 例如, 可以定义一个对普通过程调用进行加密的连接子, 设计者既可以整体复用该连接子, 也可以复用该连接子内部使用的监控设施.

2.4 部署阶段的软件体系结构建模

构件系统在运行之前, 一般都必须根据运行环境的特点进行一定的配置, 并将其安装至运行环境中. 这一过程称为软件部署, 是构件化软件开发与运行之间的一个重要阶段. 随着对软件系统服务质量要求的提高, 以及计算机处理能力、网络带宽、存储容量的快速提升, 软件部署面临着三类问题: 软件日益庞大, 难以理解; 部署信息分散, 收集过程烦琐而重复; 网络运行环境复杂、多变. 为了帮助部署人员更好地理解系统并复用开发阶段的丰富信息, ABC 将开发阶段的软件体系结构引入部署阶段 [17], 极大降低了软件部署所面临的结构复杂性.

如图 9 所示, ABC 方法支持的部署过程分为 5 个活动:

(1) 建立部署目标: 在部署之前, 部署者需要对系统在部署之后所要达到的目标有清晰的定位. 根据软件体系结构中的约束, 部署者抽取出对系统运行时刻的要求, 并综合考虑技术与非技术因素, 建立部署后系统所要达到的目标.

(2) 拆分系统: 为了能更好地利用资源、提高性能, 系统往往需要被拆分成多个部分, 每个部分分别安装到不同的计算节点上. 作为对软件总体结构的描述, 软件体系结构能够帮助部署者在较高层次上快速理解系统中的构件以及构件之间的交互和依赖关系, 分析它们对运行时刻的要求. 而对网络环境的监测, 可以辅助部署者明确地了解待部署系统的运行环境, 从而充分考虑各种实际因素的影响. 部署者在软件体系结构的指导下, 明确了底层运行环境之后, 根据部署目标进行权衡, 把系统拆分成若干个可安装到不同计算节点的部分. 在拆分的过程中, 部署者可以使用已有的部署原则, 考虑影响部署的关键因素, 从而做出合理的决策.

(3) 安装系统: 主要包括两个子活动. 一、信息配置: 系统在能够正常运行之前, 需要将一些必须的信息添加到构件运行平台上; 这些信息称为部署信息, 包括构件运行时刻绑定的名字、所需资源的

声明、资源入口等等. 系统设计阶段和实现阶段的软件体系结构模型中包含了丰富的语法、语义信息. 通过对这些信息的转换或融合, 即能得到大部分的部署信息. 此时只需添加少量的运行平台特定的信息, 即可完成系统中的信息配置活动. 二、打包: 完成拆分和信息配置后的系统各个部分, 需要打包并传送到各个具体的计算节点上去进行进一步的安装活动.

(4) 评估部署结果: 在把系统部署到各个特定节点之后, 需要对系统进行一段时间的运行测试; 然后, 部署者根据得到的运行数据对部署结果进行评估. 评估活动包含两个方面: 一方面, 部署者需要对部署目标与部署结果进行对比, 判断新系统部署后能否满足预先设定的部署目标. 另一方面, 部署者需要对比其他已部署系统的部署目标和运行情况, 判断新系统的部署是否破坏了这些系统的功能或者降低了它们的服务质量.

(5) 重部署: 当评估结果表明此次部署没有达到要求时, 就需要对相关系统进行重部署. 重部署包含三种可能的情况. 一、部署信息配置不当. 这往往会造成系统虽然能够正常运行, 但却无法达到部署目标. 例如, 对某些构件配置了不必要的高安全服务, 影响了系统的整体性能. 此时, 需要重新进行信息配置, 再次打包传送与评估. 二、对系统的拆分不合理. 这会造成整个系统的性能不符合部署目标中的要求. 因此, 需要对系统进行重新拆分, 再次安装和评估部署结果. 三、多次进行部署评估后, 发现总是未能达到预定的部署目标, 或达到了目标的部署结果并不能满足系统真正的运行要求. 此时, 很有可能是部署者原先设定的部署目标过高或过低, 因此需要重新调整部署目标, 再拆分系统、重新安装、以及重新评估部署结果.

2.5 维护与演化阶段的软件体系结构建模

据 Gartner 统计, 软件维护的成本占到整个软件生命周期成本的 50% ~ 80%. 软件维护主要对现有系统进行有限、受控的改变, 是一种不可避免的演化式开发^[18]. 维护的关键在于对现有系统的理解: 既包括对源码、文档、运行状态、工作环境等各种系统成分的理解, 也包括对触发维护的原因、可实施的维护动作、具体的维护计划、维护后的效果等各种过程成分的理解.

本质上, 软件维护需要以变化为中心, 综合性地认识与刻画软件系统的运行机理与性质. ABC 将软件体系结构从开发阶段引入运行阶段, 在可执行代码以及运行支撑环境之上, 建立了运行时软件体系结构 (RSA) 模型. 主要优点包括三点. 首先, 在线维护与演化的基本目标是系统的功能及其质量, 因此, 可通过软件体系结构及其质量来描述和考察. 其次, 在线维护与演化对运行系统的改变是受限的, 具体表现为系统可以被感知或理解的内容有限, 以及可以实施的适应性调整有限. 这种限制性可以完整准确地表述为软件体系结构的模型成分, 包括构件、连接子、配置、约束、风格等. 其三, 在线维护与演化在一般意义上是一种“监测 (Monitor)—分析 (Analyze)—决策 (Plan)—实施 (Effectuate)”的过程, 在具体操作上则可以落实为对软件体系结构的“在线获取 — 系统分析 — 配置规划 — 动态重配 — 校验确认”这样一种过程^[19,20].

如图 10 所示, 从软件体系结构的角度可以将在线维护与演化的结构复杂性归结为 4 种基本性质:

(1) 反射性 (Reflection): RSA 与目标系统的运行状态和行为之间具有因果关联关系, 即: 目标系统运行状态和行为的变化会立即导致其 RSA 的相应变化, 反之亦然. 该性质在软件体系结构层次上明确了在线维护与演化的范围, 能够保证变化和调整在模型与系统之间的一致性和准确性;

(2) 变化性 (Dynamism): RSA 通过模型查询与转换全面准确地描述了反射性所能感知的变化和计划实施的调整, 如对构件、连接子、约束的增加、删除、修改. 从时序看, 反射性描述了运行系统某个时刻的状态和行为表征, 变化性则描述了运行系统在某个时间段内的状态变迁与行为;

图 10 基于体系结构的在线维护与演化

Figure 10 Software architecture based online maintenance and evolution

(3) 多态性 (Polymorphism): RSA 在对运行系统进行维护与演化的过程中呈现出不同的内容甚至不同的形态. 该性质反映了维护与演化的一个重要现象, 即: 系统功能、质量目标、任务分工、执行时间等不同因素要求从不同角度、不同层次理解目标系统. 多态性可以表示为反射性和变化性在一段时间内的累积结果;

(4) 可信性 (Trustworthiness): RSA 通过分析来校验运行系统的功能、性能、可靠、安全等指标是否符合用户预期.

基于上述性质, ABC 监测活动利用 RSA 的反射性收集系统运行信息, 并将这些信息展现为 RSA 的变化性描述; 分析活动评估反射性和变化性所表述的系统状态和行为, 如果评估结果为功能或质量需进一步完善和优化, 则激活规划活动, 否则终止此次维护^[21]; 规划活动首先针对分析活动的评估结果, 找到监测活动的变化性描述中与功能或质量问题相关的变化, 给出适应性调整方案并生成相应的变化性描述; 控制活动则将规划活动生成的变化性描述转换成反射性操作并执行, 最终调整实际系统并对效果进行评估.

如前所述, ABC 方法在 2006 年基本成型^[7]. 其后, 随着互联网技术与应用模式的蓬勃发展, ABC 方法对其中出现的一些理念和技术进行了借鉴和吸收. 特别地, 我国学者从软件形态的角度考察开放、动态、多变的 Internet 环境对软件理论、方法和技术的挑战, 提出了网构软件的概念^[7,22,23]. 在网构软件理念的指导下, ABC 方法近几年来针对互联网软件的开发、维护和演化进行了进一步的探索. 限于篇幅, 本文着重介绍 3 个方面的工作进展: 基于群体协同的特征建模与演化—将基于 Web 的群体智能的思想引入需求阶段, 解决 ABC 特征建模活动对领域专家的强依赖性问题和特征建模结果的持续演化问题; 模型驱动的运行时软件体系结构自动生成—通过模型驱动的代码生成和模型自动同步技术, 解决 ABC 运行时体系结构需要针对不同系统进行手工开发的问题; 体系结构模型逆向恢复和追踪—通过程序理解和数据挖掘技术, 解决 ABC 方法难以直接应用到遗产系统的问题.

3 基于群体协同的特征建模与演化

面向特征的需求建模方法提供了一种建模领域需求及其空间多样性的有效途径. 但对于互联网环

境中的软件需求获取问题, 该方法还存在两点不足. 首先, 大多数特征建模方法隐含地假设: 在特定领域内存在一个或若干个关于本领域的全知者 (即, 领域专家), 他们能够有效捕获并建模领域需求及其空间多样性. 然而, 在软件应用领域层出不穷、不断演化与相互融合的现状中, 这一假设过于乐观, 在多数情况下并不成立. 其次, 大多数的特征建模方法对领域需求的演化缺乏有效的支持. 这将导致特定领域的特征模型在建立之后, 由于不能持续地演化, 而无法准确反映领域的发展现状, 从而逐渐丧失其价值. ABC 提出基于群体协同的特征建模与演化方法, 旨在将面向特征的需求建模方法与群体协同的思想相结合, 建立一种面向特征、基于群体协同的领域需求建模方法. 在此基础上, 期望通过构建基于 Web 的群体协同支撑环境, 实现对大规模、长周期、持续性的领域需求捕获、建模与演化这 3 个活动的有效支持. 该方法的核心思想如下:

首先, 在传统针对单一建模者的面向特征的需求建模方法的基础上, 应用“基于 Web 的人类群体协同”的思想, 以 Web 空间作为软件应用领域中利益相关者群体的公共环境, 利用利益相关者个体的显式建模行为和各種隱式行為對公共環境作出相應的改變, 並通過改變後的環境刺激個體在其中發生新的行為, 從而導致公共環境發生新的變化, 進而在群體中形成具有高度可伸縮性的“環境激發效應 (Stigmergy)”型的間接協同行為^[24].

其次, 應用“個體利益使能”的思想, 構建針對特徵建模問題的具体協同機制, 使得群體中的任一個體, 只要其自己對環境進行持續的探索和貢獻, 就獲得了單靠其個體能力無法構建和維護的、具有持續演化性的個性領域特徵模型, 從而使其能夠從群體協同中獲得個體利益, 並在個體利益的驅動下持續地參與到群體協同中, 進而實現群體協同的可持續性.

最後, 通過持續性的群體協同, 形成一種群體中所有個體均能通過“各盡其能”而“各得其所”的和諧生態系統. 該系統包含的一種重要成分即是存在於群體公共環境中的、面向特定領域、且持續演化的需求知識資源.

如圖 11 所示, 基於群體協同的特徵建模與演化是一個開放系統: 其輸入是來自每個個體所在的系統外部環境中的各種刺激 (包括利益及其他多樣性的信息刺激); 其輸出是個體從群體協同中獲得的利益 (即, 個體形成的個性領域需求知識). 該系統的主要構成包括: 群體公共環境、個體私有環境、個體 (個體行為)、及其之間的相互關係. 該系統的動態特徵表現為: 個體在系統外部環境和群體公共環境的刺激下, 表現出直接 (通過對公共環境的探索或貢獻) 或間接 (通過對個體私有環境的建立和維護) 作用在公共環境上的各種行為; 大量個體的行為經過不斷的聚合後, 對公共環境造成持續的改變, 從而形成個體與環境之間的信息正反饋回路, 並在宏觀上表現出“環境激發效應”型的群體協同行為.

該方法具有 4 個特點. 第一, 規模可擴展. 在環境容量的允許範圍內, 支持任何規模的用戶群體進行協同, 且用戶可以隨時加入群體協同或從其中退出. 第二, 無中央控制. 不存在一個中央節點對參與用戶的活動進行控制或協調: 用戶以一種完全自發的方式與其他用戶發生間接交互. 第三, 湧現. 通過用戶個體之間發生的大量間接交互, 在量變的基礎上產生質變, 湧現形成當前用戶群體的需求模型. 第四, 持續演化. 通過湧現形成的用戶群體需求模型不是一個靜態的制品, 而會隨著用戶之間間接交互的不斷發生而持續演化. 這種持續演化一方面會不斷提高群體協同的質量, 另一方面則能適應用戶需求自身具有的演化性.

基於上述分析, ABC 方法對已有的特徵模型進行了擴展和增強, 形成了一種面向群體的需求模型, 支持用戶群體通過基於間接交互的協同對群體需求進行建模^[25]. 具體而言, 引入支持“自定義結構”的模型擴展機制, 使得群體中的每一個體可以根據自己的需求, 對特徵的屬性、特徵屬性的取值以及

图 11 基于群体协同的特征建模与演化示意图

Figure 11 Collaboration-based feature modeling and evolution

特征/属性之间的关系进行自由定义; 引入支持“独立增量叠加”的模型扩展机制, 使得群体中的每一个体可以在已有建模制品的基础上进行独立和增量式的建模, 而所有个体的建模结果又能通过叠加的方式实时融合为一个整体的需求模型; 在上述扩展机制的基础上, 形成了基于特征模型的用户群体需求模型, 并建立了该需求模型的存储模型。

基于上述研究结果, 开发了支持群体协同的 Web 支撑环境, 并通过真实群体和仿真群体两种方式对该方法的效果进行了初步的验证. 在面向真实群体的实验中, 一个包含 4 个成员的群体在 3 小时内对一个软件系统的需求进行了建模, 形成了 113 个特征. 虽然该方法对协同式建模的具体过程并没有做出严格的限制, 但实验发现, 整个协同式建模的过程可以大致分为两个阶段: 在头脑风暴阶段中, 参与者的主要工作集中在新特征的创建, 最终结果中的大多数特征都是在该阶段被创建出来的; 在评估阶段中, 参与者主要关注对冗余特征的消除、对不清晰特征的改进、对缺失特征的创建、对特征之间关系的创建和调整, 以及对他人创建的特征的引用等. 在最终结果中, 每一参与者所引用的特征占其创建或引用的特征总数的 57.5% 至 79.8%. 平均而言, 对于每一个参与者, 在其认同的所有特征中, 有三分之二的特征都由他人创建. 在面向仿真群体的实验中, 将参与者的行为抽象为创建、查看、思考、选择与拒绝、空闲等几种基本动作, 并用 Markov 链对用户行为的概率特点进行建模. 同时, 为每个参与者赋予不同的能力, 并基于能力的不同为其建立相应的个性知识库. 在此基础上, 对以下 3 种群体的协同建模行为进行了仿真: 专家群体 (由 5 个具有高能力的专家组成)、混合群体 (由 2 个专家和 150 个普通个体组成)、普通群体 (由 250 个普通个体组成). 结果显示, 在这 3 个群体中, 普通群体建模出的特征要远远多于专家群体 (例如, 在整个仿真过程的 1/10 时刻, 专家群体建模出了 114 个特征, 混合群体建模出了 503 个特征, 普通群体则建模出了 639 个特征). 同时, 实验结果还揭示了一个现象: 普通群体中不同个体创建的特征数量体现出长尾分布的特点 (在每一次仿真中, 普通群体中总会有极少数个体, 其创建的特征的数量要远大于其他个体创建的特征的数量).

图 12 模型驱动的运行时体系结构构造过程概览

Figure 12 Model-driven construction of runtime software architecture

4 模型驱动的运行时软件体系结构自动生成

运行时体系结构目前广泛用于企业计算、桌面应用、移动计算等不同类型的软件系统中,用以支持监测、维护、重配置、自适应等不同类型的运行时管理工作^[26~29]。这些工作的核心是:针对不同的目标系统和不同的运行时管理方式,开发合适的运行时体系结构基础设施,以维护体系结构和目标系统之间的因果关联,即:系统的变化实时体现在体系结构中元素及其属性的变化上,而对体系结构的修改也会导致系统的实时变更。这种因果关联保证系统管理者可以通过读写高层运行时体系结构的方式实现对底层系统的监测和调整。

但现有工作多采用硬编码的方式构造运行时体系结构。这种方式的构造过程繁琐、难以复用,构造后的运行时体系结构难以扩展、维护,在很大程度上影响了运行时体系结构在实践中的深入应用。具体表现为:首先,在开发运行时体系结构基础设施的过程中,需要考虑系统的底层管理能力、体系结构元素的表示,以及二者之间因果关联的定义与维护等复杂问题,而手工实现这些功能的编码量非常大,也容易出错。其次,手工编码的运行时体系结构基础设施中各种功能的代码交织在一起,难以理解、维护和复用。其三,由于缺乏对运行时体系结构及其与底层系统之间关系的清晰定义,管理者往往难以正确理解和有效使用运行时体系结构。特别地,由于不同的开发者对于因果关联缺乏一致、严格的定义,而具体运行时体系结构基础设施因果关联的维护逻辑则隐藏在实现代码中,因此有可能导致管理者错误理解运行时体系结构的行为,进而导致不恰当的管理操作。

为此,ABC方法提出了一个通用的、自动化的运行时体系结构建模方法与支撑框架。根据开发者提供的模型,支撑框架自动生成相应的运行时体系结构基础设施。在系统运行过程中,该基础设施利用系统底层管理能力维护符合开发者描述的运行时体系结构,并根据开发者提供的系统与体系结构之间的关系维护二者之间的因果关联^[30]。

如图12所示,建模过程中包括系统建模和体系结构建模两个方面。系统建模描述目标系统对外提供的管理信息的类型及管理方式;建模结果包括系统模型(System Model)和访问模型(Access Model):

表 1 运行时体系结构模型与代码规模

Table 1 Runtime software architecture and the code scale

ID	Case	System model	Management map	Capability model (loc)	View element	Relation (loc)	Generated code (loc)	Additional information
1	JO2nAS-C2	305	28	310	29	157	27024	38 kB compiled
2	PLASTIC-CS	6	13	547	17	56	9126	1800 lines + 102000 lines
3	JO2nAS-CS	305	28	310	17	73	24117	—
4	Eclipse	19	23	178	44	39	11209	—
5	IOT	29	15	267	36	20	8732	—

loc: lines of code

系统模型描述目标系统对外提供的管理能力; 访问模型描述如何使用目标系统提供的管理能力对系统进行管理操作. 体系结构建模定义体系结构视图中的元素类型, 以及体系结构视图元素与系统元素之间的关系; 其建模结果分别称为体系结构视图模型和关系模型. ABC 提供了一组建模语言来辅助开发者创建和维护这些模型. 开发者仅需手工完成最上层的系统建模和体系结构建模这两个活动.

运行时体系结构的关键特征是因果关联. 而对于不同的运行时体系结构而言, 其因果关联的区别在于指导因果关联维护的系统与体系结构之间的关系. 根据系统与体系结构之间关系的不同特点, ABC 将运行时体系结构分成两类: 原生运行时体系结构 (Inherent Runtime Architecture) 和运行时体系结构视图 (Runtime Architecture), 这样有助于在建模和实现过程中做到关注点分离.

原生运行时体系结构是系统模型的实例, 它按照系统自身的结构描述系统状态 (可理解为对系统设计时软件体系结构的实例化), 但提供标准的模型读写操作. 在为一个目标系统构造运行时体系结构时, 技术上首先要能够正确读写该系统的运行时信息. 由于不同系统提供用于读写这些信息的底层管理能力在调用方式上存在较大的差异, 因而需要首先对这些异构的系统管理能力进行适配. ABC 方法通过将系统的运行时信息映射为原生运行时体系结构的方式实现这一适配. 通过经典的模型驱动代码生成技术, ABC 方法根据用户建立的模型 (Access Model 和 System Model) 自动生成系统适配器 (Sys-Arch Synchronizer) 的实现代码.

运行时体系结构视图主要用来描述符合不同管理层次或角度的体系结构模型. 在原生运行时体系结构基础上构造运行时体系结构视图的关键在于模型间的同步. 为此, ABC 方法提供了一种通用的同步机制, 能够基于两个体系结构模型 (System Model 和 Architecture Model) 和二者之间的关联模型 (Sys-arch Relation) 进行配置, 获得特定系统的原生体系结构与视图之间的同步器 (Arch-Arch Synchronizer).

ABC 运行时体系结构自动生成工具均在 Eclipse Modeling Framework(EMF) 框架的基础上实现, 具体实现细节见文献 [30, 31]. 如表 1 所示, ABC 为不同的目标系统构造了一系列面向不同管理方式的运行时体系结构. 其中, JO2nAS-C2 是在开源应用服务器 JO2nAS 上构造的符合 UCI C2 风格的运行时体系结构, PLASTIC-CS 和 JO2nAS-CS 分别是在 Windows Mobile 移动中间件 PLASTIC 和应用服务器 JO2nAS 上构造的符合 CMU Rainbow 客户/服务器风格的运行时体系结构, Eclipse-ABC 和 IOT-ABC 则是在开源集成开发环境 Eclipse 和物联网设备上构造的符合 ABC 风格的运行时体系结构. 通过分析这些运行时体系结构实例在构造过程的模型规模和生成代码的规模, 可以评估本方法对运行阶段软件复杂性控制的效果. 对于系统模型和体系结构模型, 给出了其中模型元素的总和 (包

括类、属性和关联). 对于访问模型, 给出了其中 mapping 的个数以及所有 mapping 的总代码行数之和. 对于关系模型, 给出了 QVT 代码的总行数. 对于生成代码, 给出了针对不同系统和体系结构生成的、用于体系结构表示和系统调用等功能的代码总行数.

从表中数据可以看出, 开发者需要提供的模型在规模上远小于生成的运行时体系结构支撑代码. 在系统建模方面, 除 JO2nAS 以外, 其他实例中系统模型的规模都不超过 50 个模型元素. 而 JO2nAS 系统模型可在自动推理出的系统模型基础上通过简单的调整和精化得到^[32], 因此实际工作量并不大; 其次, 管理能力模型只关注如何调用系统底层管理方法, 因此其规模远小于生成后的支撑代码规模; 最后, 由于视图模型和关系模型抽象程度较高, 模型规模也就更小. 表中每个实例自动生成的代码都在 1 万行左右甚至更多, 实例 1 的代码将近 3 万行. 文献 [26] 中指出一个典型的 C2 风格运行时体系结构基础设施, 编译后的规模达到 38 kB; 文献 [27] 中指出实现一个 C/S 风格的运行时体系结构, 在 10 万行代码的 Rainbow 框架基础上, 仍需要编写 2000 行代码. 这从另一个角度证明实现实例 1~3 的确需要较大的工作量. 从代码生成率 (生成代码行数/模型元素个数) 的角度看, 每个模型元素平均生成 26 行 java 代码. 因此, 模型驱动的方法有效提高了运行时体系结构的构造效率.

模型驱动的开发方式除提高开发效率外, 也提高了可复用性. 复用形式有两种: 整个制品的复用; 制品中相关内容的复用. 例如, 如果为 JO2nAS 系统构造一个基于 C/S 结构的运行时体系结构, 其中目标系统和体系结构风格分别在实例 1 和实例 2 中定义过, 因此构造实例 3 时, 可以直接复用实例 1 的系统模型和访问模型, 以及实例 2 的体系结构模型, 然后仅需要重新定义新的关系模型.

5 体系结构模型逆向恢复和追踪

现实中存在大量的遗产软件系统. 这些软件系统不但可能长期存在, 甚至还需要不断进行维护和演化以适应新的环境. 由于这些软件不是基于 ABC 方法进行开发的, 因此难以直接用 ABC 方法来实现自动化的维护与演化. 针对这个问题, 我们对 ABC 方法进行了进一步的扩展, 使其能够支持遗产软件系统的维护与演化. 从软件复杂性控制的角度看, 基于 ABC 方法进行软件开发时, 除了最终运行的软件, 开发过程中还产生了包括软件体系结构在内的一系列用于“分步”控制复杂性的软件制品, 而遗产软件系统往往缺少这些软件制品. 我们的基本思路是: 以遗产软件系统的代码为基础, 通过软件分析^[33], 辅助逆向建立遗产软件系统的软件体系结构模型, 并进而逆向恢复 ABC 方法所需要的其他软件制品 (特别是与遗产软件系统的维护与演化有关的软件制品), 从而使得遗产软件系统的维护与演化可以在 ABC 方法的支持下完成. 具体而言, ABC 方法探索了多种软件分析技术, 以实现从代码到设计的逆向追踪、以及从代码到需求的逆向追踪.

完成从代码到设计的逆向追踪的首要目标是逆向恢复遗产软件系统的体系结构. 针对这一问题, 需先识别代码中的构件, 并在此基础上恢复软件体系结构^[34,35]: 将遗产软件系统抽象为一个有向带权图, 以迭代的方式逐步考察图中不同粒度的子图, 并通过独立性度量选取独立性高的子图作为构件候选者. 在迭代过程中, 每次迭代得到的候选构件将被抽象为下一次迭代的图中的一个顶点. 该有向图的每个顶点代表一个类, 每条边代表类之间的关系, 边的权代表类间的依赖程度. 以该图为基础, 先根据图的大小确定每次迭代考察的子图的最大规模 k , 然后对图进行迭代分析. 每次迭代考察输入的图的所有规模小于 k 的连通子图, 采用独立性度量函数计算每个子图的独立性, 选取其中独立性高于阈值 d 的子图作为候选子图 (对应于候选构件), 然后将候选子图抽象为一个顶点, 并对输入的图进行重构, 形成一个规模较小的新图, 以新图作为下一次迭代的输入, 重复进行迭代直到重构得到的新图的

图 13 构件演化规则挖掘

Figure 13 The mining of component evolution rules

规模小于 k 。由于构件是通过迭代的方法识别出来的,且在组成结构上也具有层次关系:越晚识别出来的构件间的相互关系越紧密。这种层次化的组成关系经过精化后,即被作为通过逆向恢复产生的软件体系结构。

为了帮助遗产软件系统的维护与演化,需要挖掘构件的使用规约^[36,37],以便在维护和演化过程中避免开发人员对构件的误用。由于构件使用规约蕴含在构件的使用代码中,如果一个遗产软件系统中的构件使用代码不够充足,就需要联合多个遗产软件系统进行挖掘。针对一个目标构件,分析每个使用该构件的代码片段,将其表示为一个对构件中函数的调用序列,然后通过综合分析这些调用序列,挖掘出共性子序列,作为使用该构件应满足的约束关系。由于构件的使用代码可能分散在多个代码块甚至多个函数中,通过程序分析将分散的构件使用代码合并成统一的使用代码;由于构件中同一个函数可能涉及多个使用规则,进一步引入聚类技术以将涉及相同使用规则的调用序列聚在一起进行挖掘,从而提高了约束关系挖掘的效率和准确性。为了应对构件使用代码缺乏的情况,利用自然语言处理技术对构件文档进行分析,进一步辅助构件约束关系的挖掘:首先分析软件库文档,从中提取出每个函数的描述和类之间的继承关系;然后利用自然语言处理技术从函数的描述里抽取动作—资源对;最后基于特定的模板自动生成构件的使用规约。

在遗产软件系统的维护与演化过程中,构件本身也可能发生演化或者更替,当构件的演化或更替导致构件的使用规约也发生变化时,其他的部分也需要进行相应的修改以适应构件的这种变化。因此,需要挖掘构件使用规约的变迁或对应关系^[38,39],当一个构件演化成(或被替换成)另一个构件时,可用于检查对原有构件的使用是否满足演化后(替换后)构件的使用规约;如果不满足,则需要调整系统对该构件的使用。对于构件发生演化的情况,通过分析构件的演化历史挖掘构件使用规约的变迁:如图 13 所示,针对构件的每次修改识别该次修改导致的构件使用规约的变迁,然后将这些识别出来的变迁迭加起来得到演化前后构件规约的变迁。在分析每次修改导致的构件使用规约的变迁时,综合考虑以下三方面的信息:开发人员提交修改时的文本说明、构件中相关函数在构件代码中的调用及被调

图 14 静态特征定位

Figure 14 Static feature locating

用信息、构件中相关函数演化前后的相似程度. 实验结果表明, 该方法的准确率超过 95%, 明显高于现有的同类方法. 针对构件发生更替的情况, 通过分析已发生构件更替的软件, 挖掘不同构件使用规约间的对应关系: 将构件更替前后的构件使用代码中的多种依赖关系分别表示成两个图, 然后基于两个代码片段中的相似代码, 对两个图进行对比分析, 识别构件更替前后对应的接口, 并进而推导出构件使用规约在更替前后的变迁. 实验结果表明, 该方法可以取得较高的准确率.

针对从代码到需求的逆向追踪, 需要逆向建立代码与软件特征的关联关系^[40,41]: 对于大多数软件系统的代码, 标识符和注释中的文本信息往往体现了程序基本单元与软件特征的对应关系, 而程序的结构信息则体现了开发人员如何组织程序基本单元以实现相应的软件特征; 基于这种联系, 就可以通过代码静态确定与每个软件特征相关联的代码 (即特征定位). 如图 14 所示, 分析软件文档和程序代码中的文本信息, 利用信息检索技术确定初始关联关系, 并进一步分析程序静态结构, 对初始的关联关系进行精化和扩充, 从而完成特征定位. 基于该技术, 维护人员不需要额外维护软件特征以及软件体系结构与程序代码之间的关联关系, 而可以在需要时通过自动分析获得这种关联关系. 实验结果表明, 这种静态特征定位技术的准确率超过 90%. 由于许多软件维护与演化活动需要依据错误报告开展, 进一步将针对代码的特征分析延伸到错误报告的分析, 重点关注重复错误报告的检测^[42], 其实质是分析错误报告是否涉及相同的软件特征. 实验结果表明, 综合分析代码结构信息和文本描述信息可以大幅提高重复错误报告检测的准确率.

6 结束语

本文总结了作者在过去十余年对全生命周期软件体系结构建模的研究与实践, 并重点介绍了近几年在协同式特征建模、运行时体系结构生成以及逆向建模方面的最新进展. 随着云计算、移动互联网、

物联网、大数据等以互联网为核心的新一代信息技术的兴起, 软件形态发生了显著变化, 其复杂性也呈现出新的特点. 我们认为, 全生命周期软件体系结构建模以及 ABC 方法在新一代信息技术环境下依然具有其重要的理论和应用价值, 但同时也面临着一系列新的挑战 and 机遇, 尤其是在特征建模与知识工程的结合、软件复用与人工智能的结合、以及软件工程云和网络化操作系统的构造等方面, 还需要更系统深入的研究工作.

致谢 本文的研究得到了国家 973 计划、863 计划、国家自然科学基金、教育部重大研究计划等各类国家和省部级课题, 以及欧盟第 7 框架计划、IBM 大学合作项目、微软大学合作项目等国际合作项目的支持. 北京大学软件研究所的十多位教师和数十位研究生, 以及日本国立情报学研究所的胡振江教授和美国伊利诺伊大学厄巴纳 — 香槟分校的谢涛副教授先后参与了本项工作并做出了重要贡献. ABC 方法主体工作“全生命周期软件体系结构建模理论与方法”曾获得 2010 年高等学校自然科学一等奖和 2012 年国家自然科学二等奖. ABC 方法发表了一系列论文, 曾获 ACM SIGSOFT Distinguished Paper, ASE Best Paper Award, CBSE Best Paper Award, ICFI Most Novel Domain Award 等多次论文奖, 以及 SOCA 和 ICSR 等国际会议最佳论文奖提名. 在此一并致谢.

参考文献

- 1 Mens T. On the complexity of software systems. *IEEE Computer*, 2012, 45: 79–81
- 2 Brooks F. *The Mythical Man-Month*. Anniversary ed. Reading: Addison-Wesley, 1995
- 3 Mei H, Shen J R. Progress of research on software architecture. *J Softw*, 2006, 17: 1257–1275 [梅宏, 申峻嵘. 软件体系结构研究进展. *软件学报*, 2006, 17: 1257–1275]
- 4 Brooks F. No silver bullet: Essence and accidents of software engineering. *IEEE Computer*, 1987, 20: 10–19
- 5 Mei H. A complementary approach to requirements engineering - software architecture orientation. *ACM SIGSOFT Softw Engin Note*, 2000, 25: 40–45
- 6 Mei H, Chen F, Feng Y D, et al. ABC: An architecture based, component oriented approach to software development. *J Softw*, 2003, 14: 721–732 [梅宏, 陈锋, 冯耀东, 等. 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: 721–732]
- 7 Mei H, Huang G, Zhao H, et al. A software architecture centric development method for Internetware. *Sci China Ser-F Inf Sci*, 2006, 49: 702–730 [梅宏, 黄罡, 赵海燕, 等. 一种以软件体系结构为中心的网构软件开发方法. *中国科学 E 辑: 信息科学*, 2006, 36: 1100–1126]
- 8 Zhang W, Mei H, Zhao H. Feature-driven requirement dependency analysis and high-level software design. *Requir Eng*, 2006, 11: 205–220
- 9 Zhang W, Mei H, Zhao H. A Feature-oriented approach to modeling requirements dependencies. In: *Proceedings of the 13th IEEE International Conference on Requirements Engineering*. Washington: IEEE Computer Society, 2005. 273–284
- 10 Mei H, Zhang W, Gu F. A feature-oriented approach to modeling and reusing requirements of software product lines. In: *Proceedings of the 27th Annual International Conference on Computer Software and Applications (COMPSAC)*. Washington: IEEE Computer Society, 2003
- 11 Zhang W, Zhao H, Mei H. A propositional logic based method for verification of feature models. In: Davies J, Schulte W, Barnett M, eds. In: *ICFEM*, 2004. LNCS, vol. 3308. Heidelberg: Springer, 2004. 115–130
- 12 Yan H, Zhang W, Zhao H, et al. An optimization strategy to feature models' verification by eliminating verification-irrelevant features and constraints. In: Edwards S H, Kulczycki G, eds. *Proceedings of the 11th International Conference on Software Reuse: Formal Foundations of Reuse and Domain Engineering (ICSR)*. Berlin/Heidelberg: Springer-Verlag, 2009. 65–75
- 13 Zhang W, Mei H, Zhao H, et al. Transformation from CIM to PIM: A feature-oriented component-based approach. In:

- Briand L, Williams C, eds. *Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems (MoDELS)*. Berlin/Heidelberg: Springer-Verlag, 2005. 248–263
- 14 Zhou L, Chen X P, Huang G, et al. Negotiation-enabled modeling and verification of architectural behavior of Internetware. *J Softw*, 2008, 19: 1099–1112 [周立, 陈湘萍, 黄罡, 等. 支持协商的网构软件体系结构行为建模与验证. *软件学报*, 2008, 19: 1099–1112]
 - 15 Cui X F, Sun Y C, Mei H. A decision-centric software architecture design method. *J Softw*, 2010, (06): 1196–1208 [崔晓峰, 孙艳春, 梅宏. 以决策为中心的软件体系结构设计方法. *软件学报*, 2010, (06): 1196–1208]
 - 16 Huang G, Liu X Z, Mei H. Online approach to feature interaction problems in middleware based system. *Sci China Ser-F Inf Sci*, 2008, 51: 225–239 [黄罡, 刘讚哲, 梅宏. 基于中间件的软件系统特征交互问题在线检测与消除. *中国科学 E 辑: 信息科学*, 2007, 37: 1235–1247]
 - 17 Lan L, Huang G, Ma L Y, et al. Architecture based deployment of large-scale component based system: the tool and principles. In: *Proceedings of the 8th International Symposium on Component-based Software Engineering (CBSE)*. St. Louis, 2005. 123–138
 - 18 Lehman M. Programs, life cycles, and laws of software evolution. *Proc IEEE*, 1980, 68: 1060–1076
 - 19 Huang G, Mei H, Yang F Q. Runtime software architecture based on reflective middleware. *Sci China Ser-F Inf Sci*, 2004, 47: 555–576 [黄罡, 梅宏, 杨芙清. 基于反射式软件中间件的运行时软件体系结构. *中国科学 E 辑: 信息科学*, 2004, 34: 121–138]
 - 20 Mei H, Huang G, Lan L, et al. A software architecture centric self-adaptation approach for Internetware. *Sci China Ser-F Inf Sci*, 2008, 51: 901–920 [梅宏, 黄罡, 兰灵, 等. 基于体系结构的网构软件自适应方法. *中国科学 E 辑: 信息科学*, 2008, 38: 901–920]
 - 21 Chen X P, Huang G, Sun Y C, et al. A software architecture Model supporting analysis and planning in self-adaptation. *J Comput Res Dev*, 2010, (07): 1156–1165 [陈湘萍, 黄罡, 孙艳春, 等. 支持自适应过程中分析与规划的软件体系结构模型. *计算机研究与发展*, 2010, (07): 1156–1165]
 - 22 吕建, 马晓星, 陶先平, 等. 网构软件的研究与进展. *中国科学 E 辑: 信息科学*, 2006, 36: 1037–1080
 - 23 Mei H, Huang G, Xie T. Internetware: A software paradigm for Internet computing. *Computer*, 2012, 45: 26–31
 - 24 Theraulaz G, Bonbeau E. A brief history of stigmergy. *Artif Life*, 1999, 5: 97–116
 - 25 Zhang W, Yi L, Zhao H, et al. Feature-oriented stigmergy-based collaborative requirements modeling: An exploratory approach for requirements elicitation and evolution based on web-enabled collective intelligence. *Sci China Inf Sci*, 2013, 56: 082108(18)
 - 26 Oreizy P, Medvidovic N, Taylor R N. Architecture-based runtime software evolution. In: *Proceeding of International Conference on Software Engineering*, 1998. 177–186
 - 27 Garlan D, Cheng S, Huang A, et al. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 2004, 37: 46–54
 - 28 Huang G, Mei H, Yang F. Runtime recovery and manipulation of software architecture of component-based systems. *Autom Softw Eng*, 2006, 13: 257–281
 - 29 Blair G, Bencomo N, France R. Models@run.time. *Computer*, 2009, 42: 22–27
 - 30 Huang G, Song H, Mei H. SM@RT: Applying architecture-based runtime Management of Internetware systems. *Int J Softw Inform*, 2009, 3: 439–464
 - 31 Song H, Huang G, Chauvel F, et al. Supporting runtime software architecture: A bidirectional-transformation based Approach. *J Syst Softw*, 2011, 84: 711–723
 - 32 Song H, Huang G, Xiong Y F, et al. Inferring meta-models for runtime system data from the clients of management APIs. In: *Model Driven Engineering Languages and Systems. LNCS 6395*. Berlin/Heidelberg: Springer, 2010. 168–182
 - 33 Mei H, Wang Q X, Zhang L, et al. Software analysis: A road map. *Chinese J Comput*, 2009, 32: 1697–1710 [梅宏, 王千祥, 张路, 等. 软件分析技术进展. *计算机学报*, 2009, 32: 1697–1710]
 - 34 Luo J, Zhao W, Qin T, et al. A decomposition method for object-oriented systems based on iterative analysis of the directed weighted graph. *J Softw*, 2004, 15: 1292–1300 [罗景, 赵伟, 秦涛, 等. 一种基于有向带权图迭代的面向对象系统分解方法. *软件学报*, 2004, 15: 1292–1300]
 - 35 Zhang L, Luo J, Li H, et al. A biting-down approach to hierarchical decomposition of object-oriented systems based on structure analysis. *J Softw Mainten Evol Res Pract*, 2010, 22: 567–596

- 36 Zhong H, Xie T, Zhang L, et al. MAPO: Mining and recommending API usage patterns. In: Proceedings of 23rd European Conference on Object-Oriented Programming (ECOOP), 2009. 318–343
- 37 Zhong H, Zhang L, Xie T, et al. Inferring resource specifications from natural language API documentation. In: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering, 2009. 307–318
- 38 Zhong H, Thummalapanta S, Xie T, et al. Mining API mapping for language migration. In: Proceedings of 32nd International Conference on Software Engineering (ICSE), 2010. 195–204
- 39 Meng S, Wang X, Zhang L, et al. A history-based matching approach to identification of framework evolution. In: Proceedings of 34th International Conference on Software Engineering (ICSE), 2012. 353–363
- 40 Zhao W, Zhang L, Liu Y, et al. SNIAFL: Towards a static noninteractive approach to feature location. *ACM Trans Softw Engin Method*, 2006, 15: 195–226
- 41 Zhao W, Zhang L, Sun J, et al. An objective-oriented approach to program comprehension using multiple information sources. *Sci China Ser-F Inf Sci*, 2008, 51: 825–847
- 42 Wang X, Zhang L, Xie T, et al. An approach to detecting duplicate bug reports using natural language and execution information. In: Proceedings of 30th International Conference on Software Engineering, 2008. 461–470

ABC: a method of software architecture modeling in the whole lifecycle

MEI Hong^{1,2*}, HUANG Gang^{1,2}, ZHANG Lu^{1,2} & ZHANG Wei^{1,2}

1 *Key Laboratory of High Confidence Software Technology (Peking University), Ministry of Education of China, Beijing 100871, China;*

2 *School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China*

*E-mail: meih@pku.edu.cn

Abstract With the continually-increasing capability of computer hardware and scale of computer software, the complexity of software is also continually increasing, and has manifest itself as one of the key factors that limit the significant improvement of software quality and productivity. The structural complexity, especially the high-level structural complexity, is considered as the most important complexity of a software system. How to control the high-level structural complexity in efficient and effective ways has become a critical issue for large-scale software development in the Internet computing environment, which is open, dynamic and constantly changing. To resolve this problem, this article extends the software architecture (SA) models—a kind of model aiming to control the high-level structural complexity on software designing-into the whole software lifecycle, and proposes an SA centric software development method, called ABC. The ABC method unifies the core artifacts in different software lifecycle phases into different kinds of SA models, and then the core activities of software development, deployment, maintenance and evolution are performed as the continual refinement, mapping, and transformation to these different kinds of SA models. In particular, this article aims to give a systematic introduction to the ABC method, and present some latest research outputs in collaborative feature modeling, runtime SA generation, and SA recovering.

Keywords software complexity, software architecture, feature model, runtime model, program analysis, Inter-network

MEI Hong is a Professor in the School of Electronics Engineering and Computer Science, Peking University, China. He received a Ph.D. degree in computer science from Shanghai Jiaotong University. His research interests include software engineering, software reuse, distributed object technology and middleware, and programming languages. He is an Academician of Chinese Academy of sciences (CAS).

HUANG Gang is a Professor in the School of Electronics Engineering and Computer Science, Peking University. He received a Ph.D. degree in Computer Science from Peking University. His research interests include system software and software architecture for internet computing.

ZHANG Lu is a Professor at the School of Electronics Engineering and Computer Science, Peking University. He received a Ph.D. degree in Computer Science from Peking University. He was a postdoctoral researcher in Oxford Brookes University and University of Liverpool, UK. His research interests include software analysis, software testing, program comprehension, software maintenance, software reuse, and service computing.

ZHANG Wei is an Associate Professor at the School of Electronics Engineering and Computer Science, Peking University. He received a Ph.D. degree in Computer Science from Peking University. His research interests include software reuse, requirements engineering, and collective intelligence based software modeling.