ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.6, June 2006, pp.1257−1275
DOI: 10.1360/jos171257

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

软件体系结构研究进展*

梅 宏+, 申峻嵘

(北京大学 信息科学技术学院 软件研究所,北京 100871)

Progress of Research on Software Architecture

MEI Hong+, SHEN Jun-Rong

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

+ Corresponding author: Phn: +86-10-62753048, Fax: +86-10-62751792, E-mail: meih@pku.edu.cn, http://www.sei.pku.edu.cn

Mei H, Shen JR. Progress of research on software architecture. *Journal of Software*, 2006,17(6):1257−1275.
http://www.jos.org.cn/1000-9825/17/1257.htm

Abstract: As an important means to control the complexity of software systems, to improve software quality and to support software development and software reuse, software architecture has become a mainstream research field in the software engineering community. Now, software architecture research not only focuses on the design phase, but also covers every phase of software lifecycle. This paper surveys the update-to-date research and practice of software architecture from a software lifecycle's perspective, and discusses the potential research directions of software architecture.

Key words: software architecture; software lifecycle; survey

摘 要: 作为控制软件复杂性、提高软件系统质量、支持软件开发和复用的重要手段之一,软件体系结构自提出以来,日益受到软件研究者和实践者的关注,并发展成为软件工程的一个重要的研究领域.如今,软件体系结构的研究也开始超出传统的对软件设计阶段的支持,逐步扩展到整个软件生命周期.基于软件体系结构近十年来的研究进展,综述了在软件生命周期的不同阶段软件体系结构的研究与应用,并探讨了软件体系结构领域的发展与研究方向.

关键词: 软件体系结构;软件生命周期;综述
中图法分类号: TP311 文献标识码: A

1 软件体系结构发展历程

作为控制软件复杂性、提高软件系统质量、支持软件开发和复用的重要手段之一,软件体系结构(software architecture,简称 SA)自提出以来,日益受到软件研究者和实践者的关注,并发展成为软件工程的一个重要的研

* Supported by the National Natural Science Foundation of China under Grant Nos.60233010, 90412011, 90612011, 60403030, 60503028 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2005AA113030 (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))

Received 2006-01-20; Accepted 2006-03-13

究领域.长期以来,CMU-SEI 在其网站上公开征集 SA 的定义,至今已有百余种.其中较有影响力的定义包括:1)软件系统的结构(structure or structures),包含软件元素、软件元素外部可见的属性以及这些软件元素之间的关系[1];2) 软件系统的基本组织,包含构件、构件之间、构件与环境之间的关系,以及相关的设计与演化原则[2]等.这些定义一般都将构件以及构件之间的连接作为 SA 的基本组成部分.

文献[3]总结了 SA 的发展历史,认为 SA 的研究可追溯到 1969 年 NATO 软件工程会议.在 20 世纪 70 年代,Brooks, Dijistra, Parnas 等软件工程先驱提出了概念完整性、结构化程序设计、模块化、信息隐藏和封装等软件结构相关的重要原则.到 20 世纪 90 年代,面向对象技术已成为软件开发的主流技术,设计、开发和维护大型软件系统的需求促使研究者从更高的抽象层次关注软件,SA 也在这一阶段得到广泛关注.1995 年出版的 IEEE Software 体系结构专刊[4]和 1996 年出版的专著《Software Architecture: Perspectives on an Emerging Discipline》[5],可以认为是 SA 作为软件工程一个研究方向正式提出的标志.

此后十年内,SA 领域得到了蓬勃发展.越来越多的研究者关注并参与到 SA 的研究中来,与 SA 相关的会议、期刊、书籍等逐步增多,越来越多的知名国际会议将 SA 列入主要议题,并举行了大量直接以 SA 为主题的研讨会或国际会议(如 SA 国际研讨会 ISAW, WICSA 等).许多知名国际期刊中,与 SA 相关的研究成果逐渐增多,并出版了大量 SA 方面的书籍(如 SEI 软件工程系列丛书).SA 的研究还得到了工业界的广泛关注与认同,如 UML2 标准中引入了 SA 领域中连接子的概念[6],在实际软件开发过程(如统一软件开发过程)[7]中也引入 SA 的概念和原则.2006 年出版的 IEEE Software 软件体系结构专刊[8]总结了这十年间的 SA 研究与实践.

在研究者和实践者的共同努力与推动下,如今 SA 的研究已经解决了一些基本的理论问题,逐步从软件设计阶段扩展到对整个软件生命周期的支持,且开始了在实际软件开发中应用 SA 的探索[9].本文将试图从软件生命周期的视角来总结近十年来 SA 领域的研究成果,综述 SA 在软件生命周期不同阶段的研究和应用.本文第 2 节概述 SA 在软件生命周期不同阶段的研究点.第 3~7 节分述需求、设计、开发、部署、后开发阶段的 SA 研究.第 8 节概述 SA 的应用实践.最后总结全文并讨论 SA 可能的研究发展方向.

2 软件生命周期中的软件体系结构

最初,SA 概念的提出是为了解决从软件需求向软件实现(包括代码)的平坦过渡问题,如文献[10]就认为 SA 是软件系统的抽象描述,可作为系统实现的蓝图,担当从需求到实现的桥梁.所以,早期的 SA 研究主要集中在软件生命周期的设计阶段,关注如何通过 SA 解决软件系统的前期设计问题,典型的研究点如体系结构描述语言、体系结构风格、体系结构的验证、分析、评估方法等.

此后,随着更多不同背景研究者的参与,SA 的研究也开始超出软件设计阶段,逐步扩展到整个软件生命周期.如在系统设计前期考虑在需求中引入对体系结构的考虑,在设计后期考虑如何用 SA 支持系统的实现、组装、部署,以及开发阶段之后的维护、演化与复用等.文献[11]就提出将 SA 的概念贯穿整个软件生命周期,并用一种特殊的连接子维系不同阶段体系结构模型的可追踪性;文献[6]甚至认为,SA 除了传统意义上设计阶段的软件制品之外,更是一种软件运行形态的实体;文献[7]提出的统一软件开发过程、文献[12]所提出的 SA 生命周期集成方法、文献[13]提出的 ABC 软件开发方法等均将 SA 作为贯穿整个软件生命周期的核心制品.

在软件系统开发的过程中,SA 的主要角色包括:支持开发人员之间的交流、直接支持系统开发、支持软件复用等[5].表 1 总结了在整个软件生命周期各阶段 SA 的研究点.

Table 1 Software architecture research and practice in the whole software life cycle

表 1 软件生命周期中 SA 的研究与应用

Requirement	SA-Oriented requirement engineering, transformation from requirements to software architectures
Design	Description of SA models, SA design approach, codification and reuse of SA design experience
Implementation	SA-Supported software development process, transformation from SA design models to system implementations; SA-based software testing
Deployment	SA-Based application deployment
Post-Development	Dynamic software architecture, software architecture recovery and reconstruction

3 需求阶段的软件体系结构

需求工程和SA构造是软件生命周期的两个关键活动:需求工程[14]关注如何刻画问题空间;而SA则主要关注如何刻画解空间.所以,需求工程和SA领域中的绝大多数研究工作都相对独立.随着人们对需求工程和SA研究的深入,近年来也出现了专门讨论从需求到SA过渡问题的国际会议STRAW(International Workshop from Software Requirements to Architectures).在需求阶段研究SA,主要有如下两种工作:一是用SA的概念和描述手段在较高抽象层次刻画问题空间的软件需求;二是探讨如何从软件需求规约自动或半自动地变换到SA模型.

3.1 将软件体系结构的概念引入需求规约

把SA的概念引入到需求分析阶段,有助于保证需求规约和系统设计之间的可追踪性和一致性.如文献[15]给出了一种面向SA的需求工程方法,用SA的基本概念来描述问题空间,用构件和连接子的概念来结构化地组织需求,所得到的需求规约包括构件的需求规约、连接子的需求规约及约束需求规约等.这些需求规约文档将是设计阶段进行SA建模的基础.文献[16]提出体系结构层次需求规约的概念.体系结构层次需求规约由一组SA层次Use Case构成,每个SA层次Use Case包括相应的参与者、语义规约、质量属性规约等SA层次的元素.

3.2 从需求模型向软件体系结构模型的转换

从软件需求模型向SA模型的转换主要关注两个问题:1) 如何根据需求模型构建SA模型;2) 如何保证模型转换的可追踪性.针对这两个问题的解决方案,随着所采用的需求模型的不同而各异.在采用Use Case图描述需求的方法中,从Use Case图向SA模型(包括类图等)的转换一般经过词性分析和一些经验规则来完成[17],而可追踪性则可通过表格[17]或者Use Case Map[18]等来维护.在采用特征模型描述需求的方法中,文献[19]采用责任作为从需求模型向SA模型转换的桥梁,通过“特征-责任-构件”的映射关系来维护可追踪性;文献[20]给出了一种更为通用的方法,通过“全局分析”实现从需求模型到SA模型的转化.全局分析包括:1) 分析影响因素;2) 定义问题和制定策略,通过影响因素表、问题卡等来记录全局分析过程以维护一致性.该方法基于西门子公司的软件开发实践,并得到了较为广泛的认同.文献[21]提出一种通用的方法CBSP,该方法包括:1) 为下一次迭代选择需求;2) 需求的体系结构层次分类;3) 检测并消除分类失配;4) 需求的体系结构层次精化;5) 权衡体系结构元素和风格的选择等,通过使用体系结构层次概念(构件、连接子等)来逐步精化需求,从而得到SA的设计模型.除了上述方法之外,还有大量的研究者参与进来,如文献[22]使用面向目标的需求工程方法来结构化软件需求规约,定义APL(architectural prescription language)描述SA,并给出了将需求规约映射到APL的方法和过程;文献[23]从体系结构设计模式出发,利用模式将软件需求与体系结构设计决策联系起来.

总体上,需求阶段的SA研究还处于起步阶段.在本质上,需求和SA设计面临的是不同的对象:一个是问题空间;另一个是解空间.保持二者的可追踪性和转换,一直是软件工程领域追求的目标.从软件复用的角度看,SA影响需求工程也有其自然性和必然性,已有系统的SA模型对新系统的需求工程能够起到很好的借鉴作用.在需求阶段研究SA,有助于将SA的概念贯穿整个软件生命周期,从而保证了软件开发过程的概念完整性,有利于各阶段参与者的交流,也易于维护各阶段的可追踪性.

4 设计阶段的软件体系结构

设计阶段是SA研究关注的最早和最多的阶段,这一阶段的SA研究主要包括:SA模型的描述、SA模型的设计与分析方法,以及对SA设计经验的总结与复用等*.

---

* 软件体系结构在设计阶段的研究最为丰富和成熟.不少研究方向已经被较有影响力的综述性质的文献[24,35,58,59]所涵盖,故本文只简要介绍体系结构在设计阶段的研究,将重点放在软件体系结构在生命周期其他阶段的研究成果上.

4.1 体系结构模型的描述

有关 SA 模型描述的研究分为 3 个层次:1) SA 的基本概念,即 SA 模型由哪些元素组成,这些组成元素之间按照何种原则组织;2) 体系结构描述语言(architecture description language,简称 ADL),在 SA 基本概念的基础上,选取适当的形式化或半形式化的方法来描述一个特定的体系结构;3) SA 模型的多视图表示,从不同的视角描述特定系统的体系结构,从而得到多个视图,并将这些视图组织起来以描述整体的 SA 模型.此外,为了记录和整理上述 3 个层次的描述内容,还有学者提出了 SA 编档方法[24].

4.1.1 SA 的基本组成元素

传统的设计概念只包括构件(软件系统中相对独立的有机组成部分,最初称为模块)以及一些基本的模块互联机制.随着研究的深入,构件间的互联机制逐渐独立出来,成为与构件同等级别的一阶实体(first-class entity)[25],称为连接子.文献[26]对连接子的研究进行了总结和分类,认为连接子的作用包括通信、协调、转换和辅助交互 4 种.其中:通信和协调分别关注构件之间数据流和控制流的传递;转换则负责在通信和协调出现失配时转换数据格式、协议[27]等;辅助交互负责协调异构的构件交互(如提供适配器)或优化构件的交互(如负载平衡).简单连接子还可以通过参数化等方式构成高阶连接子[28].现阶段的 SA 描述方法主要关注对构件和连接子的建模.近年来,也有学者认为应当把 Aspect[29]、设计决策[30]等作为一阶实体引入 SA 模型.

4.1.2 体系结构描述语言

连接子概念诞生以后,软件系统的结构一般用构件、连接子及它们之间的配置加以描述.支持构件、连接子及其配置的描述语言就是如今所说的体系结构描述语言.ADL 对连接子的重视成为区分 ADL 和其他建模语言的重要特征之一.典型的 ADL 包括 UniCon, Rapide, Darwin, Wright, C2 SADL[31], Acme, xADL[32], XYZ/ADL[33], ABC/ADL[34]等.文献[35]提出了一套 ADL 的分类和比较框架,对 2000 年之前的 ADL 研究作了较为详尽的总结和评估.随着 SA 研究的不断深入,研究人员还挖掘出了 ADL 需要进一步深入考虑的问题,比如支持动态、分布、移动系统的建模,针对不同应用领域的建模等,从而提出了一些新型的 ADL,比如基于可扩展构件模型 Fractal 的 ADL[36]、从结构和行为的角度来描述 SA 的π-ADL[37]、基于多 Agent 系统的 Skwyrl-ADL[38]等.

4.1.3 SA 的多视图表示

多视图作为一种描述 SA 的重要途径,也是近年来 SA 研究领域的重要方向之一.系统的每一个不同的视图反映了一组系统相关人员所关注的系统的特定方面[39].多视图体现了关注点分离的思想,把体系结构描述语言和多视图结合起来描述系统的体系结构,能使系统更易于理解,方便系统相关人员之间进行交流,并且有利于系统的一致性检测以及系统质量属性的评估.学术界已经提出若干多视图的方案,典型的包括 4+1 模型(逻辑视图、进程视图、开发视图、物理视图,加上统一的场景)[40]、Hofmesiter 的 4 视图模型(概念视图、模块视图、执行视图、代码视图)[20]、CMU-SEI 的 Views and Beyond 模型(模块视图、构件和连接子视图、分配视图)[24]等.此外,工业界也提出了若干多视图描述 SA 模型的标准,如 IEEE 标准 1471-2000(软件密集型系统体系结构描述推荐实践)[2]、开放分布式处理参考模型(RM-ODP)[41]、统一建模语言 UML[42]以及 IBM 公司推出的 Zachman 框架[43]等.需要说明的是,现阶段的 ADL 大多没有显式地支持多视图,并且上述多视图并不一定只描述设计阶段的模型.文献[44]研究了 SA 视图的本质并尝试使用 UML 来实现这些视图,认为从软件系统模型的一致性考虑,所有视图必须既是相互独立的又是相互联系的.文献[24]总结和比较了现有的体系结构描述多视图方法.

SA 模型的描述在 SA 研究早期最受关注也相对成熟.目前,存在众多的体系结构描述语言和多视图方案,这样势必导致认知和使用上的混乱.我们认为,有必要从元模型层次探讨统一的体系结构描述方式.随着软件系统的不断发展和演化,新的问题也不断涌现,对 SA 模型的描述也将会一直是 SA 研究的热点.

4.2 体系结构设计方法

SA 的设计方法是指通过一系列的设计活动,获得满足系统功能性需求(functional requirement,简称 FR),并且符合一定非功能性需求(non-functional requirement,简称 NFR,与质量属性有相似涵义)约束的 SA 模型.现阶

段,SA 设计方法大多侧重对系统 NFR 的考虑,往往和 SA 分析方法结合使用,希望能够在软件生命周期前期发现潜在的风险[5].需要说明的是,需求阶段的体系结构研究和 SA 设计方法研究有若干重叠的研究点,如需求规约的表示、从需求向 SA 模型的转换等;但是,前者主要考虑如何组织需求以保持转换过程中的一致性和可追踪性,后者更强调具体的转换步骤以及在转换过程中所采用的设计决策,特别是针对 NFR 的设计决策.

根据在设计过程中对 FR 和 NFR 考虑的阶段不同,可以将 SA 设计方法分为 3 类:1) FR 驱动的 SA 设计,首先根据 FR 得到初步的体系结构设计模型,而后通过一定的手段精化设计结果以逐步达到 NFR 的目标,典型的方法包括评估与转化(evaluation and transformation)[45]、自顶向下组装(top-down composition,简称 TDC)[46]等;2) NFR 驱动的 SA 设计,将 NFR 作为首要考虑因素,将 NFR 直接映射成为体系结构的建模元素,典型的包括属性驱动的设计 ADD[47]等;3) 集成 FR 和 NFR 的方法,将 FR 和 NFR 视为同等重要的设计输入,在体系结构设计过程中同时兼顾 FR 和 NFR,并将其转化成相应体系结构的建模元素,这类方法往往与面向 Aspect 的方法相结合,典型的包括 Use Case 和目标驱动(use case and goal driven,简称 UCGD)[48]、形式化设计分析框架 FDAF[49]、Aspect 构件(aspectual components)[50]等.

FR 驱动的 SA 设计关注评估并调整已经得出的 SA 模型.评估的主要目的是得出 SA 模型的对质量属性目标的满足程度,常用的 SA 评估手段参见第 4.3 节.调整 SA 模型的主要依据是评估结果和质量属性需求,调整方法一般基于体系结构风格和模式、经验规则等,如 TDC 方法根据从 NFR 到解决方案映射的“特征-方案图”来进行调整.NFR 驱动的 SA 设计强调 NFR 在整个设计过程中的主导作用,如 ADD 方法基于质量属性需求(场景)确定适当的体系结构模式,而后再根据功能需求实例化模式中所包含的构件类型.集成 FR 和 NFR 的方法在面向 Aspect 的研究得到学术界关注之后迅速兴起,这类方法一般在将 FR 转化成为体系结构建模元素的同时,也将 NFR 转化成为适当的体系结构元素(如 aspect 构件等),而后考虑这两类元素的集成(如通过适当的连接子或者 aspect 编织机制等).因为需要同时考虑 FR 和 NFR,所以在需求阶段如何将 FR 和 NFR 组织到统一的视图上也成为这类方法关注的重点,如 UCGD 方法就通过 NFR 关联点将 NFR 和 Use Case 图相关联.除 Aspect 构件外,这 3 类方法一般都支持迭代的设计过程.它们的差异更多地体现在对质量属性的支持种类上,如 ADD 提供了对质量属性的广泛支持,而其他方法则一般只提供对一种质量属性(性能)的支持.

现阶段,体系结构设计方法的研究还处于起步阶段,与体系结构研究的其他领域相比,至今还没有对 SA 设计方法较为详尽的综述文献.在实际的软件开发过程中,上述方法也没有得到广泛的应用,方法的提出者一般也只能给出较小的研究案例.

4.3 体系结构分析方法

SA 分析方法可以通过分析 SA 设计所产生的模型,预测系统的质量属性并界定潜在的风险.从精度上看,体系结构分析方法可以分为两类:一类是基于形式化方法、数学模型和模拟技术,得出量化的分析结果;另一类是基于调查问卷、场景分析、检查表等手段,侧重得出关于 SA 可维护性、可演化性、可复用性等难以量化的质量属性.对于前一类分析方法,典型的研究包括基于进程代数[51]、CHAM(chemical abstraction machine)[52]、有穷状态自动机[53]、LTS(labeled transition systems)[54]等分析 SA 模型中是否包含死锁、基于排队论模型分析 SA 模型的性能[55,56]、基于马尔科夫模型分析系统的有效性[57]等;第 2 类方法是 SA 分析方法的主流,这类方法更强调 SA 的各类风险承担者(stakeholder)的参与,往往是手工完成的.典型的有基于场景的体系结构分析方法 SAAM 及其 3 个扩展方法(针对复杂场景的扩展 SAAMCS、针对可复用性的扩展 ESAAMI 和 SAAMER)、体系结构权衡分析方法 ATAM、基于场景的体系结构再工程方法 SBAR、体系结构层次软件可维护性预测方法 ALPSM 以及 SA 评估模型 SAEM 等.这类方法各有利弊,它们的差异体现在分析技术的选择、支持质量属性的种类、参与者的参与程度等.对这些方法的选用需要根据实际领域和应用情况来决定,也取决于实际参与人员的经验.文献[58,59]综述并详细比较了上述体系结构分析方法.

虽然目前存在着诸多的体系结构分析方法,但在实际应用中,这些方法却很难发挥作用.这主要是因为:第 1 类方法需要较高的数学背景,并且需要提供大量数据,往往只适用于较小规模的系统.如果将这些方法应用到大规模系统,则会出现描述复杂、难于理解和运用等问题;而第 2 类方法基本通过手工完成,且过度依赖参与者的

个人经验,方法本身仅仅给出了分析的流程,而缺乏行之有效的支撑工具.所以,统一的、自动或半自动的体系结构分析方法将是一个值得研究的问题.

4.4 体系结构设计经验的总结与复用

总结和记录(codify)软件经验是软件工程的重要目标之一.SA 的研究也强调对软件设计经验的总结和复用,所采用的主要手段为体系结构风格和模式、领域特定的软件体系结构(DSSA)[60]和软件产品线技术[61].

4.4.1 体系结构风格和模式

体系结构风格是描述某一特定应用领域中系统组织方式的惯用模式,作为"可复用的组织模式和习语"[62],为设计人员的交流提供了公共的术语空间,促进了设计复用与代码复用.体系结构模式是对设计模式的扩展,描述了软件系统基本的结构化组织方案,可以作为具体 SA 的模板[63].在实际使用中,风格和模式常常混用.从目的上看,风格和模式都是为了把设计决策记录下来;从使用上看,两者也大都使用了类似的技术来记录和阐明设计决策.一般而言,在 SA 领域,风格和模式不进行区分,统称为体系结构风格.体系结构风格的研究分为 3 个方向:总结设计经验、寻找并记录经典的风格;提供风格描述手段;在 SA 设计过程中使用风格.

现在,已总结出若干被广泛接受的体系结构风格,经典的体系结构风格包括数据流风格、调用/返回风格、独立构件风格、虚拟机风格、仓库风格等[5],之后仍有扩充,出现了基于消息的风格 C2[31]等.此外,模式领域也针对不同的系统类型提出若干种体系结构风格,如分布式系统、交互式系统和适应性系统的体系结构风格等[63].

风格的描述方法主要有两类:1) 提供非形式化描述模型,并将其引入到体系结构设计过程中,例如 Aesop[64]提供的通用的对象模型.这类方法在精确描述和性质分析方面存在缺陷,很难验证风格对于体系结构设计所施加的约束,风格的实现也只能依靠程序员的经验;2) 提供形式化规约,精确说明风格的特征,并用于高层性质验证.在这类方法中.风格被定义为从语法到语义的解释,负责规约构件、连接子和它们之间的配置的语义.风格的规约与验证依赖于所采用的形式化语言,如 Z 语言[65,66]、图论[67]等的描述和验证能力.上述两种风格的描述方法均缺乏一个行之有效的开发方法来指导风格的建模,而且在混合风格问题出现之后,陷入了发展的瓶颈.

近年来,出现了若干关注如何使用风格的研究,探讨如何针对实践的需要选择适合的风格,并保证其在 SA 设计中得到体现.研究者们提出了一些风格的使用指导框架,代表性工作有风格选择的分类指导框架[68,69]、风格构造方法[70]等,但关于风格应用的研究尚处于起步阶段.

4.4.2 DSSA 和软件产品线

DSSA 是领域工程的核心部分.领域工程分析应用领域的共同特征和可变特征,对刻画这些特征的对象和操作进行选择和抽象,形成领域模型,并进一步生成 DSSA.软件产品线是指一组具有公共的可控特征(系统需求)集的软件系统,这些特征针对特定的商业行为或者任务.产品线开发的特点是维护公共软件资产库,并在开发过程中使用这些资产,如领域模型、SA 模型、过程模型和构件等.这两种方法将特定应用领域或者产品家族的 SA 记录下来,并用于产品复用.需要说明的是,领域工程和软件产品线技术本身就是贯穿软件生命周期的软件开发方法,不过它们均在开发阶段提供了记录和复用 SA 设计经验的支持.因为这两种方法均已成为相对独立的软件工程研究领域,故本文不再详细讨论.

5 实现阶段的软件体系结构

最初的 SA 研究往往只关注较高层次的系统设计、描述和性质验证,而对缩小从体系结构层次到系统实现(如代码)层次的鸿沟关注不够.为了有效实现从 SA 设计向实现的转换,实现阶段的体系结构研究在以下几个方面进行探索:1) 研究基于 SA 的开发过程支持,如项目组织结构、配置管理等;2) 寻求从 SA 向实现过渡的途径,如将程序设计语言元素引入 SA 阶段、模型映射、构件组装、复用中间件平台等;3) 研究基于 SA 的测试技术.下面分别叙述.

5.1 SA对开发过程的支持

SA 提供了待生成系统的蓝图,根据该蓝图实现系统需要较好的开发组织结构和过程管理技术.文献[71]提

出了一种以体系结构为中心的软件项目管理方法,认为开发团队的组织结构应该和体系结构模型有一定的对应关系,并建议由体系结构设计人员担任各开发小组的组长,从而提高软件开发的效率和质量.除此之外,SA 还可以用于在开发过程中制定软件开发计划,管理风险和制定相关决策.

对于大型软件系统而言,由于参与实现的人员较多,所以需要提供适当的配置管理手段.文献[72]认为 SA 的引入能够有效扩充现有配置管理的能力,通过在 SA 描述中引入版本、可选择项(options)等信息,可以分析和记录不同版本构件和连接子之间的演化,从而可用来组织配置管理的相关活动.典型的例子包括支持给构件指定多种实现的 UniCon[73]、支持给构件和连接子定义版本信息和可选信息的 xADL[32]等.

关于 SA 对软件开发过程的支持的研究目前尚处于探索阶段,这主要是因为第 5.2 节所介绍的从 SA 模型到系统实现的转化技术还远远没有成熟,从而限制了 SA 在实现阶段对开发过程的支持力度.

5.2 从SA设计模型到系统实现的过渡

为了填补高层 SA 模型和底层实现之间的鸿沟,研究者们提出了若干方法,其主要思想是尽量封装底层的实现细节,并通过模型转换、精化等手段缩小概念之间的差距.有以下几类典型的方法:1) 在 SA 模型中引入实现阶段的概念,如引入程序设计语言元素等;2) 通过模型转换技术,将高层的 SA 模型逐步精化成能够支持实现的模型;3) 封装底层的实现细节,使之成为较大粒度构件,在 SA 指导下通过构件组装的方式实现系统,这往往需要底层中间件平台的支持.

5.2.1 在 SA 模型中引入实现阶段的概念

设计阶段模型一般用 ADL 来加以描述,为了促进从设计模型向实现阶段的转化,可以在设计阶段引入实现阶段的概念,即在 ADL 中引入与实现相关的元素.文献[74]提出了一种新型的体系结构描述语言 ArchJava,该 ADL 是 Java 语言的扩展,在 Java 语言中增加了构件、连接子、端口等建模元素用于描述 SA 模型,文献[75]又将 ArchJava 和 Acme 结合起来,并提供支持工具 AcmeStudio,可以在设计阶段直接采用 Java 语言元素进行建模,从而缩短了 ADL 与程序设计语言的距离.文献[76]将面向对象的类型系统引入到 C2 SADL 中,并可以在设计阶段通过 OO 的类、子类型化等概念来规约 SA 建模元素.在文献[32]所提出的 xADL2 中,在构件、连接子等基本建模元素的基础上引入了抽象实现的扩展机制作为实现细节的占位符,允许 SA 设计人员定义与平台或语言(如 CORBA,Java 等)相关的数据类型、函数声明等,从而可以在自动化工具的支持下,从 SA 模型无缝过渡到系统实现.文献[34]中所提出的 ABC/ADL,借鉴程序设计语言中的类型-实例关系,区分类型图和(实例)配置图,从而有利于体系结构模型到程序设计语言的转换.上述 4 种 ADL 将实现相关信息引入 ADL 的描述之中,虽然有利于从设计到实现的转换,但需要在 ADL 中引入诸如类型系统等细节信息,增加了 ADL 的复杂程度;另一方面也要求设计人员必须考虑到实现细节,在一定程度上增加了设计人员的工作量.

为了更好地发挥 SA 在系统实现阶段的指导与交流作用,研究者们提出了若干针对实现阶段的 SA 视图,如代码视图、构建(build)视图、执行视图、并发视图等.代码视图[20]和构建视图[77]用于描述最终实现系统的源代码结构.该视图以目录、源文件、中间编译文件、可执行文件等作为构件,以它们之间的包含和依赖关系为连接子,代码视图有助于控制系统的规模和安排实现计划.执行视图[20]和并发视图[40]描述运行时刻系统进程或线程之间的并发、同步关系.该视图以进程和线程为构件,以数据流、事件、对共享资源的同步等为连接子描述进程和线程之间的交互.虽然并发视图描述的是系统运行阶段的情形,但是该视图一般在实现阶段就已经给出,为实现人员提供对系统性能、可用性等方面的参考.

5.2.2 代码生成和模型转换

从设计阶段的 SA 模型向代码的转换,是将设计阶段的 SA 模型逐步精化的过程.目前的解决方案或者将高层 SA 模型直接映射成为程序代码,或者经过一系列中间模型的转换,渐进地映射到程序代码.

在工具支持下,不少 ADL 提供了从 SA 模型直接映射到代码的机制.如 C2 SADL[78]允许将体系结构设计的建模元素映射到 OO 程序设计语言,并提供了 C++和 Java 的程序库将 SA 设计时规约的概念构件、概念连接子等映射到实际的 OO 类和类之间的关联.Rapide[79]允许将 SA 设计规约映射成为某一子语言(sub-language)的实现或者常见的程序设计语言 C++和 Ada 等.除了上述特定的 ADL 之外,文献[80]定义了从通用体系结构描述语

言 ACME 向 CORBA 的映射规则,如将 System 映射成为 Module,将 Component,Port 映射成为 Interface 等,将用 ACME 规约的体系结构模型映射到 CORBA IDL.从 ADL 向程序代码的映射需要考虑若干问题,包括建立从 SA 建模元素到目标语言元素的映射关系、确保映射过程中的语义正确性、提供自动化的转换工具或环境等.但由于 ADL 通常在较高的层次规约系统的行为,缺乏实现层次上的细节说明,通过直接的映射将 ADL 转化成为程序语言往往只能生成较为简单的程序代码框架.所以,为了填补从 SA 设计到实现细节的鸿沟,研究者们提出了通过逐步精化 SA 设计模型,将 SA 设计规约转换成为实际的系统实现的方法.

模型驱动体系结构(model driven architecture,简称 MDA)[81]区分了 3 类模型:计算独立模型(CIM)、平台独立模型(PIM)以及平台特定模型(PSM).典型的 MDA 开发步骤包括:1) 用 CIM 捕获需求;2) 创建 PIM;3) 将 PIM 转化成为一个或多个 PSM,并加入平台特定的规则和代码;4) 将 PSM 转化为代码等.其中,第 3、第 4 步可以视为逐步精化 SA 设计模型(PIM),得到实现阶段的体系结构(PSM),并进一步转化成为代码的过程.作为从 SA 设计模型到可执行代码的中间层,PSM 考虑了更多与平台相关的细节,如操作系统、程序设计语言、数据存储、用户界面等.研究者和实践者们提出了很多 MDA 方法和工具,典型的如 OptimalJ[82],在自动化工具的支持下,通过内建的转换模式实现从 PIM 到 PSM 的转换,并允许用户修改所得到的 PSM 以定义更多的细节信息.现阶段,利用 MDA 方法来转换 SA 设计模型存在着若干局限:一方面,常见的 MDA 方法都是基于 UML 而不是基于 ADL,所以只能借鉴 MDA 的基本思路,无法直接使用 MDA 提供的转换工具;另一方面,现有的 MDA 方法还处于发展阶段,比如还需要用户手工加入若干与业务相关的代码,这也限制了 MDA 方法在 SA 设计模型转换中的广泛应用.需要说明的是,虽然 MDA 提供了从 SA 设计模型向代码渐进转换的机制,但 MDA 已成为相对独立的软件工程研究领域,故本文不作详细讨论.

5.2.3 构件组装

在 SA 设计模型的指导下,选择合适的可复用构件进行组装,可以在较高层次上实现系统,并能够提高系统实现的效率.在构件组装的过程中,SA 设计模型起到了系统蓝图的作用.现阶段,通过构件组装实现 SA 设计模型主要关注两方面的内容:1) 如何支持可复用构件的互联,即对 SA 设计模型中规约的连接子的实现提供支持;2) 在组装过程中,如何检测并消除体系结构失配问题.

5.2.3.1 对设计阶段连接子的支持

不少 ADL 支持在实现阶段将连接子转换到具体的程序代码或系统实现,如 UniCon[73]定义了 Pipe,FileIO, ProcedureCall 等 7 种内建的连接子类型,它们在设计阶段被实例化,并可以在实现阶段在工具的支持下转化成为具体的实现机制,如过程调用、操作系统数据访问、Unix 管道和文件、远程过程调用等.第 5.2.2 节中所介绍的支持从 SA 模型生成代码的体系结构描述语言,如 C2 SADL, Rapide 等,也都提供了一定的机制生成连接子的代码.但是,这种通过生成连接子代码复用构件的方式存在着通信完整性(communication integrity)的问题,要求所复用的构件必须满足一定的约束[74],而这种约束必须通过手工来保证,容易产生错误.近年来,随着构件标准化工作的开展和中间件技术的发展,这一问题得到了改善.

中间件[83]一般遵循特定的构件标准,为构件互联提供支持,并提供相应的公共服务,如安全服务、命名服务等.文献[84]认为,由于体系结构层次的连接子也提供类似的构件互联功能,所以中间件可以为连接子的实现提供支持.中间件支持的连接子实现有如下优势:1) 中间件提供了构件之间跨平台交互的能力,且遵循特定的工业标准,如 CORBA,J2EE,COM 等,可以有效地保证构件之间的通信完整性;2) 产品化的中间件可以提供强大的公共服务能力,这样能够更好地保证最终系统的质量属性.设计阶段连接子的规约可以用于中间件的选择,如消息通信连接子最好选择提供消息通信机制的中间件平台.从某种意义上说,随着中间件技术的发展,也导致一类新的 SA 风格,即中间件诱导的体系结构风格(middleware-induced architectural style)的出现.这些风格定义了中间件平台对应用系统体系结构的假设和约束,如中间件所支持的构件模型和通信机制分别为该风格所定义的构件模板和连接子模板[85,86].因为中间件技术已成为相对独立的研究领域,故本文不作详细讨论.

5.2.3.2 检测并消除体系结构失配

体系结构失配问题由 David Garlan 等人在 1995 年提出[87].失配是指在软件复用的过程中,由于待复用构件

对最终系统的体系结构和环境的假设(assumption)与实际状况不同而导致的冲突.在基于 SA 的构件组装阶段,失配问题主要包括:1) 由构件引起的失配,包括由于系统对构件基础设施、构件控制模型和构件数据模型的假设存在冲突引起的失配;2) 由连接子引起的失配,包括由于系统对构件交互协议、连接子数据模型的假设存在冲突引起的失配;3) 由于系统成分对全局体系结构的假设存在冲突引起的失配等.要解决失配问题,首先需要检测出失配问题,并在此基础上通过适当的手段消除检测出的失配问题.

失配的检测关注在 SA 和构件已经存在的前提下,检查构件与构件之间、构件与体系结构规约之间是否存在不匹配的情况.失配的检测一般包括:1) 基于构件规约的检测——检测待复用构件的规约是否和 SA 模型的对构件的要求相匹配[88].构件规约描述了构件从外部可见的特性,一般由构件的接口描述、接口语义约束等构成.研究者们针对特定的 ADL 和风格提出了若干检测规约匹配的规则,如 Rapide[79]基于子类型化的概念,定义了函数匹配和接口匹配的规则;C2 风格研究小组则针对 C2 风格定义了请求服务 P 匹配提供服务 Q 的规则[89].以上几种规约匹配都有相应的自动化工具的支持;2) 基于构件特征的检测——标识待复用的构件特征,并分析这些特征之间可能引起的失配问题.AAA[90]是基于构件特征的检测的典型方法,该方法首先定义了构件的 14 种概念性特征,如可重入性、并发性、动态性、封装性、控制单元、响应时间等;然后,应用这些特征来描述待复用构件,可以得出“特征-构件”表;进一步根据这些特征本身存在的约束,如“两个共享变量的并发线程隐含着同步的问题”,AAA 方法总结了 46 种可能的失配情形,根据“特征-构件”表就可以检测可能存在的失配.基于特征的检测重点和难点都在于寻找到适当的构件特征集合,这是一项需要丰富经验的活动.在得出特征集合后,就可以通过穷举的方式得出这些特征可能引起的失配问题;3) 基于构件行为的检测——采用形式化方法显式描述构件行为,以用于检测失配.形式化方法能够从特定的角度刻画构件的行为及其约束,一般均提供自动化工具来检测构件行为之间可能产生的失配.这类失配检测方法一般沿用第 4.3 节所介绍的基于形式化的 SA 分析方法.由于掌握形式化方法需要专业的培训,从而基于构件行为的检测存在实用性和扩展性方面的问题.如果将这些方法应用到大规模系统,则会出现描述复杂、难以理解和运用等问题.

失配的消除是指通过对待复用构件或者待复用构件所处的环境进行调整,以消除存在的体系结构失配.文献[91]介绍了消除两个构件之间失配的 8 种方法;文献[1]介绍了调整构件代码以消除失配的 3 种方法;文献[92]采用模式的组织方式总结了消除封装失配的 8 种方法.已有的消除失配的方法包括面向构件的失配消除和面向连接子的失配消除.面向构件的失配消除的基本思想是:在已有构件的基础上应用适当的修改或扩展机制,获得符合复用环境要求的新构件.最常见的失配消除方法包括包装(wrapper)、协商(negotiation)、引入中间表示和构件扩展技术等.包装是指将一个构件用另一种抽象包装起来,客户程序对该构件的访问需要通过包装进行;协商是指当失配的构件具有多个可选的交互方式(或数据表示、接口)时,通过协商选择彼此均能接受的交互方式(或数据表示、接口);引入中间表示包括引入中间语言表示(如用 IDL 定义失配构件的接口)和引入标准数据交换格式(如用 XML 定义失配构件的数据表示)两种方式;构件扩展技术指通过 Plug-In,Add-In 等扩展技术动态扩展待复用构件,以适应与之失配的构件的要求.面向连接子的失配消除的基本思想是:通过引入恰当的连接子(如用于转换和辅助交互的连接子)来消除某些构件交互过程中产生的失配.常见的连接子有桥接(bridge)、中介程序(mediator)、仲裁者(arbitrator)等.桥接将任意构件的某些需要的假设转换成另外某些构件提供的假设;中介程序兼有包装和桥接的特点,一般有明确的独立于失配构件的实现体,负责协调构件在数据流和控制流上的失配;仲裁者用于在失配发生时通过仲裁机制消除失配.虽然存在多种体系结构失配的消除方法,但目前还缺乏系统化的方法来指导这些方法的使用[91].文献[92,93]是系统化组织、描述这些方法的尝试,但对于何时以及如何使用这些方法,尚有待进一步深入研究.

5.2.4 结合模型转换与构件组装

代码生成和模型转换可以视为过程复用,而构件组装则是产品复用的典型代表,这是两种互补的方式.在实际应用中,通过将这两种方式结合起来,可以兼有两者的优点,从而提高从 SA 设计模型到系统实现转换的效率和质量.如在 ABC 方法[13]中,将 SA 描述作为构件开发的框架和组装系统的蓝图,将中间件技术作为构件组装所得系统的运行时支撑,使用一系列的映射规则(如从体系结构描述语言 ABC/ADL 映射到 UML 的建模元素)和工

具来缩短设计和实现间的距离,自动进行从设计到实现的转换.

5.3 基于SA的测试技术

作为软件开发的一个重要阶段,测试通过观察在输入一组测试用例的情况下程序的执行行为来动态地验证程序是否正确.可测试性是SA的重要属性之一,测试和SA之间可以互相借鉴,如体系结构可以用于自动生成测试用例、形成测试计划等;测试能够通过模拟技术评估体系结构模型,评估实现和体系结构规约的相符度.文献[94]提出了6种体系结构层次的测试维度,它们是所有数据单元、所有处理单元、所有连接子、所有转换、所有转换系统以及所有数据依赖,并认为这些测试维度可以用于生成测试用例以形成测试计划.文献[95]将体系结构作为参考模型,用于测试已实现系统和体系结构规约的相符性.文中提供了一套系统化的方法用于从体系结构规约中得出高层的设计用例,并逐步精化到代码层次的测试用例.该方法采用了形式化方法LTS,分为4步:1) 从体系结构规约中得出刻画构件之间的交互行为的SA dynamics;2) 通过转换派生出不包含预测无关信息的抽象LTS(ALTS),并将ALTS与SA测试标准相关联;3) 根据第2步的结果选择满足一定覆盖标准的测试用例;4) 生成代码层次的测试用例.上述两种方法都是将SA用于测试的典型案例.一般而言,测试和体系结构分析方法都能够用于评估SA模型的质量属性;两者不同的是,体系结构分析方法关注设计阶段所得出的SA模型性质的评估,而测试更加关注系统实现完成之后对所得到的可执行系统性质的评估.随着体系结构研究的逐步成熟并拓展到软件生命周期的各个阶段,SA在测试中的角色的研究得到越来越多学者的关注,如即将在2006年举行的第2届ROSATEA(the role of software architecture for testing and analysis)国际研讨会(第1届在1998年召开)列出了这一领域的研究方向,包括对软件质量属性的测试;对可变、动态的体系结构模型的测试;对产品线体系结构、SOA的测试等.

6 部署阶段的软件体系结构

随着网络与分布式软件的发展,软件部署逐渐从软件开发过程中独立出来,成为软件生命周期中一个独立的阶段.为了使分布式软件满足一定的质量属性要求,如性能、可靠性等,部署需要考虑多方面的信息,如待部署软件构件的互联性、硬件的拓扑结构、硬件资源占用(如CPU、内存)等.基于SA的软件部署有助于:1) 提供高层的体系结构视图描述部署阶段的软硬件模型;2) 基于SA模型可以分析部署方案的质量属性,从而选择合理的部署方案;3) 通过SA记录软件部署的经验,以便在下次部署时复用已有的部署经验.文献[40]采用物理视图刻画软件实现到物理硬件的映射关系.该视图以待部署软件、处理器、传感器、执行器、存储器等为构件,以软件实体到硬件资源的部署关系、各硬件资源的连通关系为连接子,用于指导软件的部署;文献[96]提出一种基于SA的部署方法,在自动化工具的支持下,该方法首先将系统划分成若干子系统,然后尝试部署到分布式服务器上,通过模拟测试评估部署结果,并在重部署的过程中调整部署计划,达到更优的部署结果.其中,系统的划分、部署方案的制定、模拟评估等步骤均是基于体系结构模型进行的.该方法还进一步提供了若干基于SA的部署指南;文献[97]提出一种基于SA的轻量级部署方法,该方法提供了一种体系结构风格PitM,通过体系结构描述语言C2 SADL来描述软件系统在部署阶段的拓扑结构,在可视化工具Prism的支持下,评估部署方案的影响,并自动将系统部署到硬件平台上;文献[98]则提供工具DeSi用于基于SA的可视化部署.DeSi提供了规约、操作、可视化、评估分布式系统部署阶段体系结构的能力.软件部署的研究历程较短,基于SA的软件部署的历程则更为短暂.现阶段,基于SA的软件部署研究更多地集中在组织和展示部署阶段的SA、评估分析部署方案等方面.但是,对部署方案的分析往往停留在定性的层面,并需要部署人员的参与.如何自动生成部署评估计划和最优的部署方案,将是值得研究的问题.

7 后开发阶段的软件体系结构

后开发阶段是指软件部署安装之后的阶段.这一阶段的SA研究主要围绕维护、演化、复用等方面来进行.典型的研究方向包括:动态软件体系结构、体系结构恢复与重建等.

7.1 动态软件体系结构

传统的 SA 研究设想体系结构总是静态的,即软件的体系结构一旦建立,就不会在运行时刻发生变动.但人们在实践中发现,现实中的软件往往具有动态性,即它们的体系结构会在运行时发生改变.SA 在运行时发生的变化包括两类.一类是软件内部执行所导致的体系结构改变.比如,很多服务器端软件会在客户请求到达时创建新的构件来响应用户的需求.某个自适应的软件系统可能根据不同的配置状况采用不同的连接子来传送数据.另一类变化是软件系统外部的请求对软件进行的重配置.比如,有很多高安全性的软件系统,这些系统在升级或进行其他修改时不能停机.因为修改是在运行时刻进行的,体系结构也就动态地发生了变化.在高安全性系统之外也有很多软件需要进行动态修改,比如很多操作系统期望能够在升级时无须重新启动系统,在运行过程中就完成对体系结构的修改.

由于软件系统会在运行时刻发生动态变化,这就给体系结构的研究提出了很多新的问题.如何在设计阶段捕获体系结构的这种动态性,并进一步指导软件系统在运行时刻实施这些变化,从而达到系统的在线演化或自适应甚至自主计算,是动态体系结构[99]所要研究的内容.现阶段,动态软件体系结构研究可分为两个部分:1) 体系结构设计阶段的支持.主要包括变化的描述、根据变化如何生成修改策略、描述修改过程、在高抽象层次保证修改的可行性以及分析、推理修改所带来的影响等;2) 运行时刻基础设施的支持.主要包括系统体系结构的维护、保证体系结构修改在约束范围内、提供系统的运行时刻信息、分析修改后的体系结构符合指定的属性、正确映射体系结构构造元素的变化到实现模块、保证系统的重要子系统的连续执行并保持状态、分析和测试运行系统等.

很多学者提出了在体系结构层次刻画系统的动态性的方法.现阶段,用于描述动态软件体系结构的形式化方法包括图论、进程代数等.文献[100]介绍了用图来描述体系结构动态性的方法——用图的顶点表示构件,边表示连接子,并引入协调机制管理体系结构,动态性则体现在图的重写(graph rewriting)上;文献[101]在 Darwin 体系结构描述语言中采用推迟实例化(lazy instantiation)、动态实例化(dynamic instantiation)等特殊机制来描述动态性,并用π演算描述了其语义;文献[102]扩展了 Wright 体系结构描述语言得到 Dynamic Wright,通过引入负责连接构件和连接子的配置器(configurator)来描述体系结构的动态性,并可以基于 CSP 的验证工具进行校验;文献[103]提出了用体系结构级别事件描述和分析动态性的方法;文献[104]在理论层面上进行了大量工作,并综合使用 3 种方法(基于 Transaction 的方法、基于 CHAM 的方法和基于 CommUnity 的方法)来描述体系结构动态性;文献[105]用 CSP 描述了体系结构构件动态更新机制的形式化模型,并验证了该模型的无死锁性、客户端的透明性和服务器端的正确性等属性.其他类似的研究还有很多,如文献[106]提出一套动态软件体系结构的分类框架,总结了动态体系结构形式化规约的研究成果,并从动态性的种类、过程和基础设施 3 个角度比较了这些方法.除了形式化方法描述体系结构动态性之外,还有学者基于 XML 等描述体系结构动态性,如文献[107]用 AML(体系结构修改描述语言)描述对体系结构进行修改所必须执行的操作,用 ACL(体系结构约束语言)来描述体系结构修改操作执行时所必须满足的约束.

在运行时刻,基础设施层次支持动态性,包含如下研究内容:1) 获取运行时刻的 SA 模型,并维护该模型和实际运行系统之间的因果关联;2) 获取或者生成体系结构动态调整策略;3) 根据第 2 步得到的调整策略调整运行时刻系统等.文献[107]所提出的动态软件体系结构支撑系统包括 3 个部分:体系结构模型的图形化表示(argo)、运行时修改的文本命令接口(archShell)、实施具体的修改操作(extension wizard).文献[108]也提出一个 3 层系统——运行时刻层负责监测系统的运行时刻的属性并执行系统适配的低级操作;模型层负责解释被监测到的系统行为,并用较高层次、较容易分析的属性来描述;任务层负责确定服务的质量需求,并提供了一个工具集用于支持上述方法,该工具集包括用来提供图形显示的 AcmeStudio、提供约束分析和策略的 Armani、对运行系统进行监测的 Gauge 基础设施,以及执行调整任务的 Tailor.随着中间件技术的发展,学者们也开始探讨将反射式中间件用于支持体系结构动态性,如文献[109]采用反射式运行时刻环境 WebGOP Runtime 来维护面向图的 SA,并可以通过该环境根据图的变化来调整运行系统;文献[110,111]提出用反射式中间件 PKUAS 来获取运行系统体系结构模型(称为运行时刻体系结构 RSA)并维护该模型与实际运行系统的因果关联.RSA 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.在反射式软件中间件的支持下,RSA 不仅积累了设计、组装和部署阶段的信息,还实时地反映出系统运行时刻的真实状态,并且通过对 RSA 的操作,可以对软件系统进行在线维护与演化.

随着系统规模的进一步扩大,系统维护与演化也变得更加复杂.如何将软件结构用于系统的自主维护与演化,日益成为研究者们关注的话题.如 ACM 连续召开两届 WOSS(ACM SIGSOFT Workshop on Self-Healing/Self-Management Systems)研讨会,讨论体系结构在自修复、自管理系统中的作用,就体现了这一趋势.

7.2 体系结构恢复与重建

当前系统的开发很少是从头开始的,大量的软件开发任务是基于已有的遗产系统进行升级、增强或移植.这些系统在开发的时候没有考虑 SA,在将这些系统进行构件化包装、复用的时候,会得不到体系结构的支持.因此,从这些系统中恢复或重构体系结构是有意义的,也是必要的.

SA 重建是指从已实现的系统中获取体系结构的过程.一般地,SA 重建的输出是一组体系结构视图.文献[112]认为以下的视图是有意义的:概念视图、构件视图、开发视图、任务视图、特征视图等.现有的体系结构重建方法可以分为 4 类:1) 手工体系结构重建.如文献[113]提出了用于 OO 系统的体系结构手工重建方法.该方法以 emacs 和 grep 为基本工具,通过检查源码的方式来发现构件,所有的视图都是使用纸和笔绘制.2) 工具支持的手工重建.通过工具对手工重建提供辅助支持,包括获得基本体系结构单元、提供图形界面允许用户操作 SA 模型、支持分析 SA 模型等.如 KLOCwork inSight 工具(www.klocwork.com/products/insight.asp)使用代码分析算法直接从源代码获得 SA 构件视图,用户可以通过操作图形化的 SA 设定体系结构规则,并可在工具的支持下实现对体系结构的理解、自动控制和管理.3) 通过查询语言来自动建立聚集.这类方法适用于较大规模的系统,基本思路是:在逆向工程工具的支持下分析程序源代码,然后将所得到的体系结构信息存入数据库,并通过适当的查询语言得到有效的体系结构显示.如文献[114]通过软件分析工具直接从源代码获取体系结构元素信息并存储在 PostgreSQL 数据库中,通过将 SQL 和 Perl 组合使用,可以进行多种查询并产生多种体系结构视图.4) 使用其他技术,比如数据挖掘[115]等.文献[116]从多视图、质量属性变更、设计与实现一致性、体系结构中通用性和可变性、二进制构件和混合语言等角度分析和比较了现阶段 SA 重建方法.

当前,SA 重建的研究面临着很多挑战,这主要是因为:1) SA 自身的研究不够成熟,对 SA 的描述方法、语义问题等多个方面并没有统一的认识;2) 直接从源代码层次进行重建的可能性不大,因为大量的商用和 COTS 软件并不存在现实可用的源代码;3) 缺乏必要的工具和方法支持,许多项目尚处于研究探索阶段.

8 软件体系结构应用实践概述

自 SA 提出以来,一直注重理论研究与工业实践相结合[1].文献[9]列举了现阶段 SA 实践成熟的标志,包括工业级培训与认证、标准的软件体系结构、成熟的体系结构模式和策略分类、端对端生命周期模型、可重复的体系结构评估和校验方法、成熟的工具支持、企业级体系结构基础设施层和应用层支持、软件架构师的职业化以及大量的会议和期刊等.总体而言,SA 在工业界的应用和推广体现在以下几个方面:

1) 工业标准的制定.如 IEEE 专门制定了与体系结构相关的国际标准[2];SAE 制定了 ADL 的国际标准 AADL[117];在 OMG 所制定的 UML 标准中,沿用了文献[40]所定义的 4+1 视图;最新推出的 UML2.0 标准则充分吸纳了现有 SA 的研究成果,引入了连接子、复合构件等 SA 领域提出的概念;在不少工业级框架中,也将 SA 领域提出的连接子概念显式化,如 JSR 112 标准中即制定了 J2EE Connector Architecture,用于连接异构的系统.

2) 实际产品的开发.如西门子[20]、贝尔实验室[118]等公司大力推动 SA 在实际软件产品开发中的应用,并通过联合项目、学术研讨会等形式,将其在工业实践中所积累的经验贡献给体系结构研究者,如 CMU 的软件工程研究所(SEI)中就拥有大量来自工业界的研究人员.在软件企业中,软件架构师(software architect)[119]作为一种专门的职业独立出来,成为与软件项目经理并列的技术领导者,典型的如微软公司创始人将自己的职位界定为首席软件架构师(chief software architect).

3) 相关书籍和课程.SA 得到了工业界的诸多关注还体现在相关书籍的出版和课程的开设上.如 CMU-SEI

成立了体系结构技术促进会,组织推出了一整套与体系结构相关的图书、课程和产品[120];国际上也成立了软件架构师协会(Worldwide Institute of Software Architects)和软件架构师国际联盟(International Association of Software Architects),并通过出版图书、会员活动等方式推动 SA 的教育与应用.

9 结束语和进一步的研究工作

在研究者和实践者的共同努力与推动下,如今 SA 的研究已经渗透到软件生命周期的各个阶段,并取得了丰硕的研究成果.与软件工程研究中其他领域(如结构化方法、面向对象方法)一样,体系结构的研究首先关注软件生命周期的一个阶段(设计),然后逐步过渡到设计之后的阶段(实现、部署、后开发),最后再关注设计之前的阶段(需求分析),从而成为覆盖各阶段的一整套方法.各阶段的研究成果也按照这种发展结构分布.本文总结并根据软件生命周期的不同阶段来组织 SA 的研究成果,在体系结构研究领域是一次新的尝试,希望能为 SA 的进一步研究提供有益的参考.

随着软件系统的规模逐步扩大,体系结构在实际软件开发中的作用也日益显著,因而应该进一步加强 SA 在软件生命周期各个阶段的研究与实践.另一方面,Internet 技术的发展促使新的软件形态——网构软件[121,122]的出现.网构软件为了适应开放、动态和多变的运行环境,呈现出柔性、多目标和连续反应式的形态.这会导致网构软件的体系结构和组成构件处于不断的调整和适应的过程中,从而带来了在新的环境下研究新型 SA 的需求.我们认为还需要从以下 4 个方面对 SA 进行深入探讨:

1) 传统 SA 研究领域的进一步探讨以及对未决问题的进一步研究.如从需求到 SA 的自动或半自动转化,统一的基于元模型的体系结构描述方法,混合体系结构风格,更为实用的体系结构评估与分析手段,结合构件组装和模型转换的从设计到应用系统的转化方法,基于体系结构的自动化构件组装和部署,运行时刻体系结构的表示和系统平台支持,对遗产系统体系结构重建和复用,体系结构在整个软件生命周期的可追踪性支持,体系结构设计、分析、评估工具支持等;

2) SA 在网构软件生命周期中的角色.与传统软件相比,网构软件更为复杂、多变和开放,这进一步加大了对网构软件结构的理解、分析和开发的难度.如何界定 SA 在网构软件整个生命周期中的角色,将是一个值得关注和研究的课题.其中的主要研究点包括:对网构 SA 描述与分析的方法、基于体系结构的网构软件的质量属性保障机制等;

3) 基于体系结构的软件开发方法学.软件的开发涉及多方面的问题,通过发挥 SA 在软件生命周期中的核心作用,能够有效地组织软件的开发、部署、维护与演化.我们目前正致力于这方面的工作,并初步提出了一个以 SA 为核心的软件开发过程[123],即以 SA 描述为系统蓝图,以特征模型为设计起点,以中间件技术为构件组装、部署的运行时支撑,使用一系列的映射规则和工具来缩短需求、设计和实现间的距离,使相应的转换过程自动化,给出了一组需求特征模型到 SA 的映射和转换规则,将 SA 引入基于中间件的应用部署和维护;

4) 体系结构对实际软件开发的支持.如何将学术研究成果应用于实际的软件开发一直是困扰研究者的问题.现阶段,虽然体系结构实践已经取得了初步成果,但在实际产品开发中,仍然主要依靠软件架构师的个人经验,而系统化地使用 SA 来指导开发,尚缺乏行之有效的方法和案例.所以,还需要进一步探讨将体系结构应用到实际软件开发中的方法,如将体系结构相关的概念与流程集成到软件开发环境中去、研究体系结构与现有的软件开发方法的融合与集成、开展与体系结构相关的教育培训等.

致谢 我们谨向对本文的工作给予支持和建议的同行,尤其是北京大学信息科学技术学院软件研究所软件体系结构研究小组成员表示感谢.本文试图涉及 SA 研究的方方面面,但由于作者涉猎面所限,难免挂一漏万.也由于涉及方面太多,难免在某些方面流于表面而欠缺深度.因此,特别感谢读者的宽容和理解.愿本文能为从事 SA 研究和实践的同行提供一定参考.

References:

[1] Bass L, Clements P, Kazman R. Software Architecture in Practice. 2nd ed. Boston: Addison Wesley Professional, 2003.

[2] IEEE. IEEE recommended practice for architectural description of software-intensive systems. IEEE, IEEE Std1471-2000, 2000.

[3] Kruchten P, Obbink H, Stafford J. The past, present, and future of software architecture. IEEE Software, 2006,23(2):22−30.

[4] Garlan D, Perry D. Special issue on software architecture. IEEE Software, 1995,12(6).

[5] Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. New Jersey: Prentice Hall, 1996.

[6] Garlan D. A ten-year perspective on "Formalizing Architectural Connection". Most Influential Paper of the ICSE'94 Award Presentation, Given at ICSE 2004. 2004. http://www-2.cs.cmu.edu/~garlan/ICSE05-MIP.pdf

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

[8] Kruchten P, Obbink H, Stafford J. Special issue on software architecture. IEEE Software, 2006,23(2).

[9] Shaw M, Clements P. The golden age of software architecture. IEEE Software, 2006,23(2):31−39.

[10] Garlan D. Software architecture: A roadmap. In: Proc. of the 22nd Int'l Conf. on Software Engineering, Future of Software Engineering Track. New York: ACM Press, 2000. 91−101. http://portal.acm.org/toc.cfm?id=336512&type=proceeding&coll=GUIDE&dl=GUIDE&CFID=75507652&CFTOKEN=34663824

[11] Medvidovic N, Grünbacher P, Egyed A, Boehm BW. Bridging models across the software lifecycle. Journal of Systems and Software, 2003,68(3):199−215.

[12] Kazman R, Nord RL, Klein M. A life cycle view of architecture analysis and design methods. Technical Report, CMU/SEI-2003-TN-026, Pittsburgh: Carnegie Mellon University, 2003.

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

[14] Nuseibeh B, Easterbrook S. Requirements engineering: A roadmap. In: Proc. of the 22nd Int'l Conf. on Software Engineering, Future of Software Engineering Track. New York: ACM Press, 2000. 35−46.

[15] Mei H. A complementary approach to requirements engineering-software architecture orientation. Software Engineering Notes, 2000,25(2):40−45.

[16] Svetinovic D. Architecture-Level requirements specification. In: Proc. of the 2nd Int'l Software Requirements to Architectures Workshop. 2003. 14−19. http://se.uwaterloo.ca/~straw03/ProceedingsSTRAW03.pdf

[17] Shao WZ. Object Oriented System Analysis. Beijing: Tsinghua University Press, 1998 (in Chinese).

[18] Buhr RJA. Use case maps as architectural entities for complex systems. IEEE Trans. on Software Engineering, 1998,24(12):1131−1155.

[19] Zhang W, Mei H, Zhao HY, Yang J. Transformation from CIM to PIM: A feature-oriented component-based approach. LNCS 3713, 2005. 248−263.

[20] Hofmeister C, Nord R, Soni D. Applied Software Architecture. Boston: Addison-Wesley Professional, 2000.

[21] Medvidovic N, Dashofy EM, Taylor RN. The role of middleware in architecture-based software development. Int'l Journal of Software Engineering and Knowledge Engineering, 2003,13(4):367−393.

[22] Brandozzi M, Perry DE. From goal-oriented requirements to architectural prescriptions: The preskriptor process. In: Proc. of the 2nd Int'l Software Requirements to Architectures Workshop. 2003. 107−113.

[23] Rajasree MS, Reddy PK, Janakiram D. Pattern oriented software development: Moving seamlessly from requirements to architecture. In: Proc. of the 2nd Int'l Software Requirements to Architectures Workshop. 2003. 54−60.

[24] Clements P, Bachmann F, Bass L, Nord RL, Garlan D, Ivers J, Little R, Nord R, Stafford J. Documenting Software Architectures: Views and Beyond. Boston: Addison Wesley Professional, 2002.

[25] Shaw M. Procedure calls are the assembly language of software interconnection: Connectors deserve first-class status. LNCS 1078, 1996. 17−32.

[26] Mehta NR, Medvidovic N, Phadke S. Towards a taxonomy of software connectors. In: Proc. of the 22nd Int'l Conf. on Software Engineering. New York: ACM Press, 2000. 178−187. http://portal.acm.org/toc.cfm?id=337180&type=proceeding&coll=GUIDE&dl=GUIDE&CFID=75508182&CFTOKEN=3248003

[27] Spitznagel B, Garlan D. A compositional formalization of connector wrappers. In: Proc. of the 25th Int'l Conf. on Software Engineering. Washington: IEEE Computer Society Press, 2003. 374−384.

[28] Lopes A, Wermelinger M, Fiadeiro JL. Higher-Order architectural connector. ACM Trans. on Software Engineering and Methodology, 2003,12(1):64−104.

[29] Quintero C, Romay MP, Fuente P, Barrio-Solórzano M. Reflection-Based, aspect-oriented software architecture. LNCS 3047, 2004. 43−56.

[30] Tyree J, Akerman A. Architecture decisions: Demystifying architecture. IEEE Software, 2005,22(2):19−27.

[31] Medvidovic N, Rosenblum DS, Taylor RN. A language and environment for architecture-based software development and evolution. In: Proc. of the 21st Int'l Conf. on Software Engineering. New York: ACM Press, 1999. 44−53.

[32] Dashofy EM, Hoek A, Taylor RN. A comprehensive approach for the development of modular software architecture description languages. ACM Trans. on Software Engineering and Methodology, 2005,26(1):199−245.

[33] Luo HJ, Tang ZS, Zheng, JD. Visual architecture description language XYZ/ADL. Journal of Software, 2000,11(8):1024−1029 (in Chinese with English abstract).

[34] Mei H, Chen F, Wang QX, Feng YD. ABC/ADL: An ADL supporting component composition. LNCS 2495, 2002. 38−47.

[35] Medvidovic N., Richard NT. A classification and comparison framework for software architecture description languages. IEEE Trans. on Software Engineering, 2000,26(1):70−93.

[36] Bruneton E, Coupaye T, Stefani JB. Recursive and dynamic software composition with sharing. In: Proc. of the 7th Workshop on Component-Oriented Programming. 2002. http://fractal.objectweb.org/current/fractalWCOP02.pdf

[37] Oquendo F. π-ADL: An architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. Software Engineering Notes, 2004,29(3):1−14.

[38] Faulkner S, Kolp M. Towards an agent architectural description language for information systems. In: Proc. of the 5th Int'l Conf. on Enterprise Information Systems, Vol. 3. Setúbal: ICEIS Press, 2003. 59−66.

[39] Perry DE, Wolf AL. Foundations for the study of software architecture. Software Engineering Notes, 1992,17(4):40−52.

[40] Kruchten PB. The 4+1 view model of architecture. IEEE Software, 1995,12(6):42−50.

[41] ISO. ISO/IEC 10746 reference model of open distributed processing. ISO, 1996.

[42] Object Management Group. The unified modeling language. http://www.uml.org

[43] Zachman JA. A framework for information systems architecture. IBM Systems Journal, 1987,26(3):276−292.

[44] Egyed A, Medvidovic N. Extending architectural representation in UML with view integration. Technical Report, Center for Software Engineering, University of Southern California, 1999.

[45] Bosch J, Molin P. Software architecture design: Evaluation and transformation. In: Proc. of the IEEE Conf. and Workshop on Engineering of Computer-Based Systems. Washington: IEEE Society Press, 1999. 4−10.

[46] Bruin H, Vliet H. Quality-Driven software architecture composition. Journal of Systems and Software, 2003,66(3):269−284.

[47] Ellison RJ, Moore AP, Bass L, Klein M, Bachmann F. Security and survivability reasoning frameworks and architectural design tactics. CMU/SEI-2004-TN-022, Pittsburgh: Carnegie Mellon University, 2004.

[48] Supakkul S, Chung L. Integrating FRs and NFRs: A use case and goal driven approach. In: Proc. of the 2nd Int'l Conf. on Software Engineering Research, Management & Applications. 2004. 30−37. http://citeseer.ist.psu.edu/630349.html

[49] Dai L, Cooper K. Modeling and analysis of non-functional requirements as aspects in a UML based software architecture design. In: Proc. of the 6th Int'l Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. 2005. 178−183.

[50] Xu L, Ziv H, Richardson D, Liu Z. Towards modeling nonfunctional requirements in software architecture. In: Proc. of the Aspect-Oriented Requirements Engineering and Architecture Design Workshop (Early Aspects 2005), Held in Conjunction with AOSD 2005. 2005. http://www.isr.uci.edu/~lihuax/images/EAW-XuZIvRichardson.pdf

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

[52] Inverardi P, Wolf AL, Yankelevich D. Static checking of system behaviors using derived component assumptions. ACM Trans. on Software Engineering and Methodology, 2000,9(3):239−272.

[53] Zhang B, Ding K, Li J. An XML-message based architecture description language and architectural mismatch checking. In: Proc. of the 25th Annual Int'l Computer Software and Applications Conf. Washington: IEEE Society Press, 2001. 561−567.

[54] Uchitel S, Kramer J, Magee J. Behaviour model elaboration using partial labelled transition systems. In: Proc. of the 4th ACM SIGSOFT Symp. on the Foundations of Software Engineering. New York: ACM Press, 2003. 19−27.

[55] Spitznagel B, Garlan D. Architecture-Based performance analysis. In: Proc. of the 10th Int'l Conf. on Software Engineering and Knowledge Engineering. Illinois: Knowledge Systems Institute Press, 1998. 146−151.

[56] Marco DA, Inverardi P. Compositional generation of software architecture performance QN models. In: Proc. of the 4th Working IEEE/IFIP Conf. on Software Architecture. Washington: IEEE Society Press, 2004. 37−46.

[57] Grassi V, Mirandola R. Derivation of markov models for effectiveness analysis of adaptable software architectures for mobile computing. IEEE Trans. on Mobile Computing, 2003,2(2):114−131.

[58] Dobrica L, NiemelaÈ E. A survey on software architecture analysis methods. IEEE Trans. on Software Engineering, 2002,28(7): 638−653.

[59] Clements P, Kazman R, Klein M. Evaluating Software Architectures: Methods and Case Studies. Boston: Addison Wesley Professional, 2002.

[60] Li KQ, Chen ZL, Mei H, Yang FQ. An introduction to domain engineering. Computer Science, 1999,26(5):21−25 (in Chinese with English abstract).

[61] Clements P, Bachmann F, Bass L. Software Product Lines: Practices and Patterns. Boston: Addison-Wesley Professional, 2002.

[62] Garlan D. What is style? In: Proc. of the Dagshtul Workshop on Software Architecture, 1995. http://www.cs.cmu.edu/afs/cs/project/able/ftp/style-iwass95/style-iwass95.pdf

[63] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. Pattern-Oriented Software Architecture, Vol. 1: A System of Patterns. West Sussex: John Wiley & Sons, 1996.

[64] Garlan D, Allen R, Ockerbloom J. Exploiting style in architectural design environments. In: Proc. of the 2nd ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 1994. 175−188.

[65] Abowd G, Allen R, Garlan D. Using style to understand descriptions of software architecture. In: Proc. of the ACM SIGSOFT 1st Symp. on Foundations of Software Engineering. New York: ACM Press, 1993. 9−20.

[66] Abowd G, Allen R, Garlan D. Formalizing style to understand descriptions of software architecture. ACM Trans. on Software Engineering and Methodology, 1995,4(4):319−364.

[67] Metayer DL. Software architecture styles as graph grammars. In: Proc. of the 4th ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 1996. 15−23.

[68] Shaw M. Making choices: A comparison of styles for software architecture. IEEE Software, 1995,12(6):27−41.

[69] Shaw M, Clements P. A field guide to boxology: Preliminary classification of architecture styles for software systems. In: Proc. of the 21st Int'l Computer Software and Application Conf. Washington: IEEE Society Press, 1997. 6−13.

[70] Mehta NR, Medvidovic N. Concise composition of architectural styles from architectural primitives. In: Proc. of the 11th ACM SIGSOFT Symp. on the Foundation of Software Engineering. New York: ACM Press, 2003. 347−350.

[71] Paulish DJ. Architecture-Centric Software Project Management: An Introduction. Boston: Addison Wesley Professional, 2002.

[72] Hoek A, Heimbigner D, Wolf AL. Investigating the applicability of architecture description in configuration management and software deployment. Technical Report, CU-CS-862-98, Boulder: University of Colorado, 1998.

[73] Shaw M, DeLine R, Klein DV, Ross TL, Young DM, Zelesnik G. Abstractions for software architecture and tools to support them. IEEE Trans. on Software Engineering, 1995,21(4):314−355.

[74] Aldrich J, Chambers C, Notkin D. ArchJava: Connecting software architecture to implementation. In: Proc. of the 24th Int'l Conf. on Software Engineering. New York: ACM Press, 2002. 187−197.

[75] Abi-Antoun M, Aldrich J, Garlan D, Schmerl B, Nahas N, Tseng T. Modeling and implementing software architecture with acme and archJava. In: Proc. of the Int'l Conf. on Software Engineering. New York: ACM Press, 2005. 676−677.

[76] Medvidovic N, Oreizy P, Robbins JE, Taylor RN. Using object-oriented typing to support architectural design in the C2 style. In: Proc. of the 4th ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 1996. 24−32.

[77] Tu Q, Godfrey MW. The build-time software architecture view. In: Proc. of the IEEE Int'l Conf. on Software Maintenance. Washington: IEEE Society Press, 2001. 398−407.

[78] Medvidovic N, Mehta NR, Mikic-Rakic M. A family of software architecture implementation frameworks. In: Proc. of the 3rd IEEE/IFIP Conf. on Software Architecture. Deventer: Kluwer BV Press, 2003. 221−235.

[79] Luckham DC, Kenney JJ, Augustin LM, Vera J, Bryan D, Mann W. Specification and analysis of system architecture using rapide. IEEE Trans. on Software Engineering, 1995,21(4):336−355.

[80] Rodrigues U, Lucena L, Batista T. From acme to CORBA: Bridging the gap. LNCS 3047, 2004. 103−114.

[81] Frankel DS. Model Driven Architecture: Applying MDA to Enterprise Computing. West Sussex: John Wiley & Sons, 2003.

[82] Meservy TO, Fenstermacher KD. Transforming software development: An MDA roadmap. Computer, 2005,38(9):52 −58.

[83] Emmerich W. Software engineering and middleware: A roadmap. In: Proc. of the 22nd Int'l Conf. on Software Engineering, Future of Software Engineering Track. New York: ACM Press, 2000. 117−129.

[84] Medvidovic N, Dashofy EM, Taylor RN. The role of middleware in architecture-based software development. Int'l Journal of Software Engineering and Knowledge Engineering, 2003,13(4):367−393.

[85] Nitto ED, Rosenblum DS. Exploiting ADLs to specify architectural styles induced by middleware infrastructures. In: Proc. of the 21st Int'l Conf. on Software Engineering. Los Alamitos: IEEE Computer Society Press, 13−22.

[86] Zhu YL, Huang G, Mei H. Modeling diverse and complex interactions enabled by middleware as connectors in software architectures. In: Proc. of the 10th IEEE Int'l Conf. on the Engineering of Complex Computer Systems. Washington: IEEE Computer Society Press, 2005. 37−46.

[87] Garlan D, Allen R, Ockerbloom J. Architectural mismatch: Why reuse is so hard? IEEE Software, 1995,12(6):17−26.

[88] Zaremski MA, Wing JM. Specification matching of software components. ACM Trans. on Software Engineering and Methodology, 1997,6(4):333−369.

[89] Egyed A, Medvidovic N, Gacek C. Component-Based perspective on software mismatch detection and resolution. IEE Proc. of Software, 2000,147(6):225−236.

[90] Gacek C. Detecting architectural mismatches during system composition [Ph.D. Thesis]. Los Angeles: University of Southern California, 1998.

[91] Shaw M. Architectural issues in software reuse: It's not just the functionality; it's the packaging. In: Proc. of the ACM SIGSOFT Symp. on Software Reusability. New York: ACM Press, 1995. 3−6.

[92] Deline R. A catalog of techniques for resolving packaging mismatch. In: Proc. of the '99 ACM SIGSOFT Symp. on Software Reusability. New York: ACM Press, 1999. 44−53.

[93] Keshav R, Gamble R. Towards a taxonomy of architecture integration strategies. In: Proc. of the 3rd Int'l Workshop on Software Architecture. New York: ACM Press, 1998. 89−92.

[94] Richardson DJ, Wolf AL. Software testing at the architectural level. In: Proc. of the 2nd Int'l Workshop on Software Architecture. New York: ACM Press, 1996. 68−71.

[95] Muccini H, Bertolino A, Inverardi P. Using software architecture for code testing. IEEE Trans. on Software Engineering, 2004, 30(3):160−172.

[96] Lan L, Huang G, Ma LY, Wang M, Mei H, Zhang L, Chen Y. Architecture based deployment of large-scale component based systems: The tool and principles. LNCS 3489, 2005. 123−138.

[97] Mikic-Rakic M, Medvidovic N. Architecture-Level support for software component deployment in resource constrained environments. In: Proc. of the IFIP/ACM Working Conf. on Component Deployment. London: Springer-Verlag, 2002. 31−50.

[98] Mikic-Rakic M, Malek S, Beckman N, Medvidovic N. A tailorable environment for assessing the quality of deployment architectures in highl1273 distributed settings. LNCS 3083, 2004. 1−17.

[99] Medvidovic N. ADLs and dynamic architecture changes. In: Proc. of the 2nd Int'l Workshop on Software Architecture. New York: ACM Press, 1996. 24−27.

[100] Hirsch D, Inverardi P, Montanari U. Graph grammars and constraint solving for software architecture styles. In: Proc. of the Int'l Workshop on Software Architecture. New York: ACM Press, 1998. 69−72.

[101] Magee J, Kramer J. Dynamic structure in software architectures. In: Proc. of the ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 1996. 3−14.

[102] Allen R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures. LNCS 1382, 1998. 21−37.

[103] Vera J, Perrochon L, Luckham DC. Event-Based execution architectures for dynamic software systems. In: Proc. of the Working IEEE/IFIP Conf. on Software Architecture. Deventer: Kluwer BV Press, 1999. 22−24.

[104] Wermelinger M. Specification of software architecture reconfiguration [Ph.D. Thesis]. Universidade Nova de Lisboa, 1999.

[105] Shen JR, Sun X, Huang G, Jiao WP, Sun YC, Mei H. Towards a unified formal model for supporting mechanisms of dynamic component update. In: Proc. of the ACM SIGSOFT Symp. on Foundations of Software Engineering. New York: ACM Press, 2005. 80−89.

[106] Bradbury JS, Organizing definitions and formalisms of dynamic software architectures. Technical Report, 2004-477, Kingston: Queen's University, 2004.

[107] Oreizy P, Medvidovic N, Taylor RN. Architecture-Based runtime software evolution. In: Proc. of the 20th Int'l Conf. on Software Engineering. Washington: IEEE Computer Society Press, 1998. 177−186.

[108] Cheng SW, Garlan D, Schmerl B, Steenkiste P, Hu NN. Software architecture-based adaptation for grid computing. In: Proc. of the 11th IEEE Conf. on High Performance Distributed Computing. Washington: IEEE Computer Society Press, 2002. 389−398. http://portal.acm.org/citation.cfm?id=822086&coll=GUIDE&dl=ACM&CFID=71066936&CFTOKEN=4207474

[109] Ma XX, Cao JN, Lü J. Architecting distributed web applications: A graph-oriented approach. Chinese Journal of Computers, 2003, 26(9):1104−1115 (in Chinese with English abstract).

[110] Huang G, Mei H, Yang FQ. Runtime recovery and manipulation of software architecture of component-based systems. Int'l Journal of Automated Software Engineering, 2006,13(2):251−278.

[111] Huang G, Mei H, Yang FQ. Runtime software architecture based on reflective middleware. Science in China (F), 2004,47(5): 555−576.

[112] Riva C, Yang YJ. Generation of architectural documentation using XML. In: Proc. of the 9th Working Conf. on Reverse Engineering. Washington: IEEE Computer Society Press, 2002. 161−169. http://portal.acm.org/toc.cfm?id=882506&type=proceeding&coll=GUIDE&dl=GUIDE&CFID=71067007&CFTOKEN=85189257

[113] Laine PK. The role of SW architecture in solving fundamental problems in object-oriented development of large embedded SW systems. In: Proc. of the Working IEEE/IFIP Conf. on Software Architecture. Washington DC: IEEE Computer Society Press, 2001. 14−23. http://portal.acm.org/toc.cfm?id=832312&type=proceeding&coll=GUIDE&dl=GUIDE&CFID=71067614&CFTOKEN=78917093

[114] Kazman R, Carrière SJ. Playing detective: Reconstructing software architecture from available evidence. Journal of Automated Software Engineering, 1999,6(2):107−138.

[115] Sartipi K, Kontogiannis K. A graph pattern matching approach to software architecture recovery. In: Proc. of the IEEE Int'l Conf. on Software Maintenance. Washington: IEEE Computer Society Press, 2001. 408−419. http://portal.acm.org/toc.cfm?id=846228&type=proceeding&coll=portal&dl=ACM&CFID=71067633&CFTOKEN=90422167

[116] Stoermer C, O'Brien L, Verhoef C. Practice patterns for architecture reconstruction. In: Proc. of the 9th Working Conf. on Reverse Engineering. Washington: IEEE Computer Society Press, 2002. 151−160. http://portal.acm.org/toc.cfm?id=882506&type=proceeding&coll=&dl=GUIDE&CFID=71067674&CFTOKEN=35574387

[117] Society for Automotive Engineers, Architecture Analysis & Design Language (AADL). SAE Standard AS5506, 2004.

[118] Maranzano J. Best Current Practices: Software Architecture Validation. AT&T Bell Labs., 1990.

[119] Malveau R, Mowbray T. Software Architect Bootcamp. New Jersey: Prentice Hall, 2000.

[120] Software Engineering Institute (SEI), Carnegie Mellon University. Software architecture technology (SAT) initiative: Products and services. http://www.sei.cmu.edu/architecture/products_services/index.html

[121] Yang FQ, Mei H, Lu J, Jin Z. Some discussion on the development of software technology. Acta Electronica Ronica Sinica, 2002, 30(12A):1901−1906 (in Chinese with English abstract).

[122] Lü J, Tao XP, Ma XX, *et al*. On agent-based software model for internetware. Science in China (E), 2005,35(12):1233−1253 (in Chinese with English abstract).

[123] Mei H. ABC: Supporting software architectures in the whole lifecycle. In: Proc. of the 2nd Int'l Conf. on Software Engineering and Formal Methods. Washington: IEEE Computer Society Press, 2004. 342−343. http://portal.acm.org/toc.cfm?id=1030033&type=proceeding&coll=portal&dl=ACM

附中文参考文献:

[17] 邵维忠.面向对象的系统分析.北京:清华大学出版社,1998.

[33] 骆华俊,唐稚松,郑建丹.可视化体系结构描述语言 XYZ/ADL.软件学报,2000,11(8):1024−1029.

[60] 李克勤,陈兆良,梅宏,杨芙清.领域工程概述.计算机科学,1999,26(5):21−25.

[109] 马晓星,曹建农,吕建.一种面向图的分布 Web 应用架构技术.计算机学报,2003,26(9):1104−1115.

[121] 杨芙清,梅宏,吕建.金芝.浅论软件技术发展.电子学报,2002,30(12A):1901−1906.

[122] 吕建,陶先平,马晓星,胡昊,徐锋,曹春.基于 Agent 的网构软件模型研究.中国科学(E 辑),2005,35(12):1233−1253.

梅宏(1963－),男,重庆人,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程及软件开发环境,软件复用及软件构件技术,(分布)对象技术,软件工业化生产技术及支持系统,新型程序设计语言.

申峻嵘(1982－),男,硕士生,主要研究领域为软件体系结构,软件构件技术.

****

2006 年全国开放式分布与并行计算学术会议
征 文 通 知

由中国计算机学会开放系统专业委员会主办、西北大学信息学院(计算机系)承办、陕西省计算机学会协办的“2006 年全国开放式分布与并行计算学术会议”将于 2006 年 10 月 19－21 日在西安召开，现将会议征文的有关事项通知如下：

一、征文范围(包括但不限于下列方面)

1. 开放式分布与并行计算模型、体系结构、算法及应用；
2. 下一代开放式网络、数据通信、网络与信息安全、业务管理技术；
3. 开放式海量数据存储与 Internet 索引技术，分布与并行数据库及数据/Web 挖掘技术；
4. 开放式机群计算、网格计算、Web 服务、P2P 网络及中间件技术；
5. 开放式移动计算、移动代理、传感器网络与自组网技术；
6. 分布式人工智能、多代理与决策支持技术；
7. 分布、并行编程环境和工具；
8. 分布与并行计算算法及其在科学与工程中的应用；
9. 开放式虚拟现实技术与分布式仿真；
10. 开放式多媒体技术与流媒体服务，包括媒体压缩、内容分送、缓存代理、服务发现与管理技术。

二、征文要求

详见会议主页：http://disnet.nwu.edu.cn/DPCS2006 或 http://cs.nju.edu.cn/dpcs

三、重要日期

会议时间：2006 年 10 月 19～21 日
截稿日期：2006 年 7 月 15 日
录用通知：2006 年 7 月 30 日

四、联系方式

投稿地址：西安西北大学信息学院 陈晓江 收(请在信封上注明 DPCS2006)
邮政编码：710069
联系电话：029-88308273(房鼎益、陈晓江)
电子邮件：chenxr@mail.xjtu.edu.cn (请在邮件主题中注明 DPCS2006)
专委会联系人：南京大学计算机系 陈贵海，电话：025-58916715， 电子邮件：gchen@nju.edu.cn