ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.6, June 2006, pp.1360−1371
DOI: 10.1360/jos171360
© 2006 by *Journal of Software*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种面向动态软件体系结构的在线演化方法*

余 萍[1,2+], 马晓星[1,2], 吕 建[1,2], 陶先平[1,2]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)
[2](南京大学 计算机科学与技术系,江苏 南京 210093)

A Dynamic Software Architecture Oriented Approach to Online Evolution

YU Ping[1,2+], MA Xiao-Xing[1,2], LÜ Jian[1,2], TAO Xian-Ping[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)
[2](Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China)
+ Corresponding author: Phn: +86-25-83593694, Fax: +86-25-83593283, E-mail: yuping@ics.nju.edu.cn, http://www.nju.edu.cn

Yu P, Ma XX, Lü J, Tao XP. A dynamic software architecture oriented approach to online evolution. *Journal of Software*, 2006,17(6):1360−1371. http://www.jos.org.cn/1000-9825/17/1360.htm

Abstract: An increasingly important requirement for software systems is the capability to evolve dynamically according to the changes of computing environment and users' need during runtime. In this paper, a dynamic software architecture oriented approach is proposed to support online evolution. Notably, architecture information is reified as explicit and manipulable entities to organize a runtime architecture meta-model, which is causally connected to software implementation and specification. By using reflection, the evolved architecture meta-model modifies running system, and updates specification simultaneously. The well-defined architecture meta-model supervises all evolutionary behaviors to ensure system consistency, integrity, and evolution traceability. Based on this model, a visualized integrated development platform entitled Artemis-ARC is also successfully implemented. It supports component and service design, development and assembly, especially dynamic evolution. Furthermore, a simple application case is developed with Artemis-ARC to illustrate the effect of online evolution.
Key words: dynamic software architecture; online evolution; reflection; consistency; integrity; traceability

摘 要: 为适应计算环境和用户需求在系统运行期间的变化,满足软件系统进行动态演化的需求,提出面向动态软件体系结构的在线演化方法,设计并实现了一种运行时刻的软件体系结构元模型,将原先运行时刻不可见的体系结构设计信息具体化为显式的体系结构实体,并与系统实现及系统规约之间保持因果关联.元模型的演化可通过反射实现对运行系统的修改和对规约的更新,所有演化行为都在良定义的体系结构元模型的指导下规范地进行,保证了演化前后系统的一致性、完整性和演化的可追溯性.基于该方法开发了可视化支撑平台Artemis-ARC 系统,支持构件和服务的设计、开发、集成及动态演化,并通过简单的应用实例展示了在线演化的

* Supported by the National Natural Science Foundation of China under Grant Nos.60403014, 60233010 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2005AA113160, 2005AA119010, 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.2002CB312002 (国家重点基础研究发展规划(973))

Received 2006-01-09; Accepted 2006-03-13

© 中国科学院软件研究所 http://www.jos.org.cn

效果.

关键词: 动态软件体系结构;在线演化;反射;一致性;完整性;追溯性
中图法分类号: TP311 文献标识码: A

随着移动计算、普及计算和 Web Service 等技术的迅速发展,软件系统所面临的计算环境变得开放、多元和易变.为适应运行时刻计算环境中网络、设备、资源等的变化和用户需求的改变,在不中断系统运行的情况下进行“在线”演化的需求日益增长.然而,由于“在线”演化也容易破坏系统的完整性和一致性,这就造成了目前仍很少有应用系统实践“在线”演化的尴尬.为此,一些研究者开始从软件体系结构(SA)尤其是动态软件体系结构(DSA)的角度寻求对系统“在线”演化的支持[1−4].

SA 从全局的视角描述了系统的构件组成、构件之间的相互关系和连接方式所形成的拓扑结构,及其所遵循的模式和受到的约束等[5],DSA 则指那些在运行时刻会发生变化的体系结构[6].传统上使用体系结构描述语言(ADL)定义的 SA 规约仅仅停留在设计阶段,程序员根据 SA 规约实现系统的时候将体系结构信息都隐含在计算代码之中,系统实现以后便很难继续维护这些信息.尽管 SA 或 DSA 在系统动态演化中的作用已引起一定关注,但目前对软件体系结构动态特性的研究仍主要集中在 ADL 方面[2,7,8],而在软件运行时刻却普遍缺乏显式的动态体系结构视图,难以通过分析运行时刻的体系结构来指导软件的在线演化,这就导致了在线演化或者有可能脱离原有的设计,容易破坏系统的一致性和完整性,或者不得不被推迟到系统离线维护阶段重新设计.

本文详细分析了软件系统的动态特性,提出一种面向动态软件体系结构的在线演化方法.在该方法中,设计阶段的体系结构规约被具体化为内置于运行系统的可编程的体系结构元模型,在体系结构元模型的基础上自顶向下开发的实现系统与元模型维护着因果连接(causal connection)的关系.通过反射(reflection),运行时刻对体系结构元模型的修改一方面驱动了软件系统的演化,另一方面实时更新了与之关联的体系结构规约,保证了演化的一致性、完整性和可追溯性.我们将该方法应用到 CBSD(component based software development)和 SOA(service oriented architecture)领域中,开发了支撑平台 Artemis-ARC,支持构件/服务的设计、开发、集成和在线演化.

本文第 1 节分析软件系统的动态特性和面向 DSA 的软件设计、实现、演化架构.第 2 节介绍面向 DSA 的在线演化实现方法.第 3 节介绍该方法的实际应用.第 4 节讨论相关工作.最后总结全文.

1 动态软件体系结构

1.1 动态特性

SA 通常是对系统的静态描述,如果需要改变体系结构则必须重新设计新的 SA,这已不能适应现在越来越多的需要在运行时刻发生变化的系统的设计需求.DSA 则允许系统在执行过程中修改其体系结构,修改过程通常也被称为运行时刻的演化(即在线演化)或动态性[6].根据修改的内容不同,体系结构的动态变化可分为如下几个方面:

(1) 结构:软件系统为适应当前的计算环境往往需要调整自身的结构,比如增加或删除构件、连接子,这将导致 SA 的拓扑结构发生显式的变化;

(2) 行为:由于用户需求的变化或者系统自身 QoS 调节的需要,软件系统在运行过程中会改变其行为,比如由于安全级别的提高更换加密算法;将 http 协议改为 https 协议,行为的变化往往是由构件或连接子的替换和重配置引起的;

(3) 属性:已有的 ADL 大都支持对非功能属性(non functional properties)的规约和分析,比如对服务响应时间和吞吐量的要求等,在系统运行的过程中这些要求可能发生改变,而这些变化又会进一步触发软件系统结构或行为的调整.属性的变化是驱动系统演化的主要原因;

(4) 风格:软件体系结构风格代表了相似的软件系统的基本结构以及相关的构造方法[5].一般来说,演化前

© 中国科学院软件研究所 http://www.jos.org.cn

后软件的体系结构风格应该保持不变,如非要发生改变也只能是“受限”的演化,即只允许体系结构风格演化为其“衍生”风格.风格的“衍生”关系类似于面向对象中的继承关系.比如将原有的两层 C/S(client-server)结构调整为 3 层或多层的 B/S(browser-server)结构,将“1 对 1”的请求/响应结构改为“1 对 *N*”的结构,以实现负载平衡等.

在运行时刻实施这些变化的时候需要保证变化不会破坏系统体系结构的“一致性”和“完整性”[2,8],“一致性”和“完整性”是在线演化能否实施的必要条件.同时,为了便于软件演化后的维护,还需要进一步考虑演化的可“追溯性”.

“一致性”有 4 层含义:(1) 体系结构规约与系统实现的一致性,运行时刻的修改应及时地反映到规约中,以保证规约不会过时;(2) 系统内部状态的一致性,正在修改的部分不应被其他用户或模块更改;(3) 系统行为的一致性,若“管道-过滤器”风格的结构中增加一个过滤器,则需要保证该过滤器的输入和输出与相连的管道的要求一致;(4) 体系结构风格的一致性,演化前后体系结构或者保持风格不变,或者演化为当前风格的“衍生”风格.

“完整性”意味着系统的演化不能破坏 SA 规约中的约束,比如限制与某构件相连的构件数目为 1,若在演化过程中删除了与它相连的原有构件,或者为它增加了一个新的相连构件,都会导致系统出错;“完整性”还意味着演化前后系统的状态不会丢失,否则系统将变得不“安全”,甚至不能正确运行.

“追溯性”:传统的 ADL 采用逐步精化的方式将一个抽象层次很高的 ADL 规约逐步精化为具体的可直接实现的 ADL 规约,在精化的过程中通过形式化的验证保证每一步精化都符合要求,满足可追溯性[6].但对于动态系统而言,仅仅如此是不够的,追溯性需要被延伸到运行时刻,以保证系统的任何一次修改都会被验证,这样既有利于软件的维护,也为软件的进一步演化提供了可分析的依据.

已有的 ADL 及其相关工具或系统对体系结构动态性的支持较弱,大都是在规约语言层次上,通过形式化系统模拟系统的动态性,比如基于π演算的 Darwin[2]和基于 CSP 的 Wright[9],它们通常只能在体系结构设计阶段以声明式的规约定义受限的几种演化行为,系统实现后体系结构信息则被隐含在各个分散的模块及其交互中,无法在运行时刻维护显式的体系结构视图,演化行为的正确性不得不由程序员或借助外置的修改管理器保证,且很难继续和 SA 规约保持一致;也有从实现层次上,利用中间件来构建基于软件体系结构自适应的系统,例如 ArchStudio[3].ArchStuido 可以使用命令式的体系结构修改语言(AML)和约束语言(ACL)支持运行时刻修改系统结构,并能在一定程度上保证系统的“一致性”和“完整性”[8].ArchJava 则尝试把体系结构概念引入编程语言,通过 ArchJava 语言将规约和实现联系起来,可以保证通信的完整性,但由于软件的实现必须依赖于这种新型的语言,使得此方法的适用范围也较为有限[10].

1.2 面向DSA的架构

针对已有工作的不足,本文提出面向 DSA 的软件系统在线演化方法,尝试通过显式的运行时刻体系结构将体系结构规约的影响从系统设计阶段延续到运行和演化阶段,即动态软件系统的设计、实现和演化都将围绕体系结构展开.体系结构不再只是设计阶段的“静态”规约,而是被具体化为运行时刻显式存在于系统中的可操纵的“动态”实体,该实体从全局的视角描述了软件系统自身的体系结构组成,被称为体系结构元模型.它将软件的设计,实现和演化联系起来,形成以体系结构为中心的软件生命周期的循环过程(如图 1 所示).

图 2 给出了体系结构元模型与 SA 规约和系统实现,以及外部环境(情境信息、用户需求、QoS 等)之间的关系,它在规约与实现之间建立起因果关联,为系统的在线演化提供了一个面向 DSA 的“反射”架构.反射(reflect)是指计算系统通过与自身状态和行为具有因果互联的系统自述,以描述、推理和操纵自身的能力[11].面向 DSA 的架构利用反射计算的原理和技术,将软件系统的体系结构具体化为运行时刻可操纵的体系结构元模型,可对元模型进行编程、定义接口、生成或绑定构件和实现代码,通过对元模型的修改完成规约的更新和实现的演化.元模型内置于运行系统中,是系统必要的组成部分.

原则上并不限定描述体系结构所使用的 ADL(目前也还没有一个统一的语言),但鉴于 ACME 是一种通用的体系结构变换语言,它提供了在不同 ADL 之间实现变换的机制,并可以被扩展[7,12],我们采用它作为体系结构元模型的语义基础,并对它进行扩展(增加了 Behavior 元素等)以引入动态特性.ACME 属于规约语言,为了将它具体化为运行时刻可操纵的实体,需要建立一套相应的程序级表达机制和设施.我们借助于面向对象技术对体

© 中国科学院软件研究所 http://www.jos.org.cn

系结构元模型建模,将其中主要元素的映射关系进行归纳,见表 1.

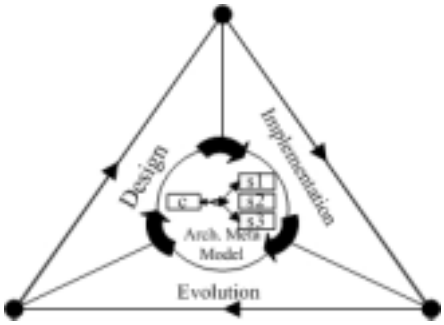


Fig.1 Architecture centric software life cycle

图 1 体系结构为中心的软件生命周期

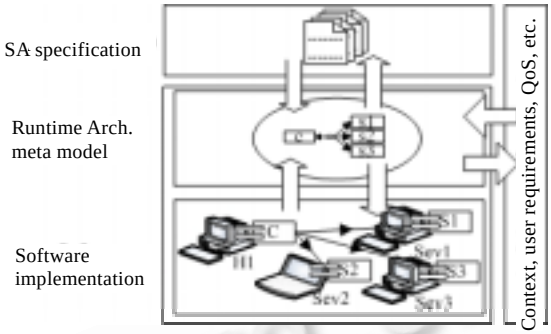


Fig.2 DSA oriented framework

图 2 面向 DSA 的架构

Table 1 Element mappings between ACME and architectural meta model

表 **1** ACME 和体系结构元模型的元素映射

ACME based SA spec.	Object oriented arch. meta model
SA style (family)/system	Architecture class/architecture object
Component type/component	Component class/component object
Connector type/connector	Connector class/connector object
Port type/port	Port class/port object
Role type/role	Role class/role object
Properties	Properties object
Invariant	validate() method
Heuristic	Property listener
Behavior* (extended for dynamism)	Modification interfaces

利用程序级的表达机制,体系结构具有了可编程能力,可以通过对体系结构元模型的编程将演化逻辑从实现系统的程序中分离出来.体系结构元模型本质上亦属于运行系统组成的一部分,从而可以在运行时刻充分利用体系结构设计阶段的知识(通常是专家知识)来指导系统的演化.

2 面向 DSA 的在线演化

2.1 体系结构元模型

可编程的体系结构元模型是抽象的体系结构信息在运行时刻的具体表达,它一方面需要准确地刻画“静态”的体系结构,另一方面需要提供“动态”修改体系结构的接口.结合扩展后的 ACME,并按照从“一般”到“特殊”、从“静态”到“动态”的顺序,我们认为体系结构元模型需要包含以下要素:

(1) 体系结构的“静态”结构:体系结构视图中的构件、连接子及其连接方式需要在体系结构实体中显式地表达.图 3 采用树型结构描述了“静态”体系结构模型中各个元素之间的层次关系,图 4 则给出了一个最简单的 Client/Server 风格(simpleCS)的体系结构图形化定义,及具体化生成的体系结构实体的内部状态(即体系结构上下文)(该例将贯穿本文).

(2) 元素属性:对于图 3 中后面加*的元素,可以定义其属性列表.根据属性是否可修改,将其分为两类:静态属性和动态属性,因此属性被定义为三元组 *Property*=(*PropertyName*, *PropertyValue*, “*static*”|“*dynamic*”).静态属性一般是元素的内部特性,比如图 4 中的 RPC 连接子,若规定其通信方式为同步的(“synchronous”,“true”,“static”),则表示不可以在运行时刻修改它;动态属性通常是一些非功能属性,例如图 4 中 C 构件期望 *S*1 构件的响应时间为 1000ms(“expResponseTime”,“1000”,“dynamic”),以及要在运行阶段实时获取的属性,比如当前响应时间(“responseTime”,“900”,“dynamic”).这些属性的值可能随着用户需求的变化或运行状况的不同而发生改变,是系统进行演化的动机之一.为此,体系结构元模型提供了监听动态属性值变化的基础设施 *ChangeListener*,比

© 中国科学院软件研究所 http://www.jos.org.cn

如增加监听“expResponseTime”属性值变化的监听器:“*addChangeListener*(*new PropertyListener*(“*expResponseTime*”, “1000”))”,用户若改变“expResponseTime”属性的值,监听器将及时捕获到该事件并通知体系结构元模型.

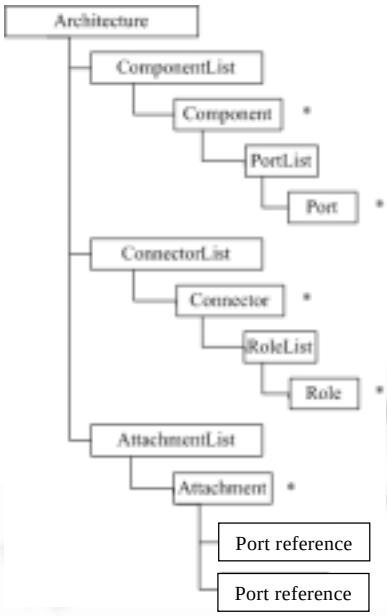


Fig.3 Elements of SA static model

图 3 静态结构模型组成元素及其关系

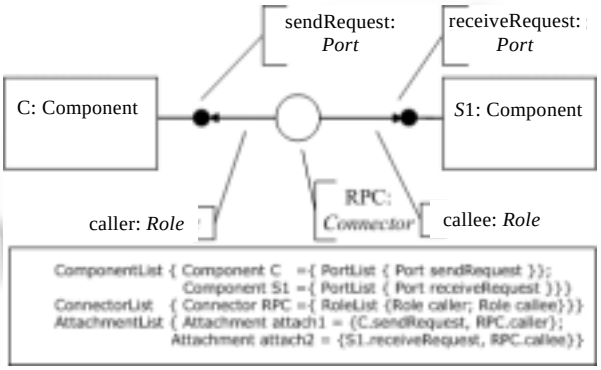


Fig.4 A simplified SA spec

图 4 一个简单的 SA 规约视图

(3) 体系结构风格和约束:在体系结构元模型中,体系结构的风格被映射为体系结构实体的类,比如,图 4 展示的 SimpleCS 风格被具体化为 SimpleCS 类型的体系结构类.风格的衍生关系在体系结构元模型中则体现为类的继承,规定所有的体系结构类都直接或间接地扩展自全局体系结构类 *RuntimeArchitecture*.体系结构风格一般通过定义约束(或不变式)限制结构的构造方法,不同的 ADL 对体系结构风格中的约束有不同的表述方法,典型的如 ACME 中基于一阶谓词逻辑的约束描述语言[12],以及属性图文法中对图的特性的推导[13],通常可以在 SA 设计阶段使用工具对结构进行约束检验.但是当系统实现并投入运行以后,除非有显式的体系结构模型,否则难以使用这些工具对运行系统的结构进行检查,内置于运行系统中的体系结构元模型可以满足这个条件.更进一步地,体系结构类中允许定义“自检验”(*public boolean validate*())的方法支持体系结构元模型对自身的状态进行约束检查(见第 2.2.3 节).

(4) 修改体系结构的元接口:需要提供支持修改体系结构的接口以改变运行时刻体系结构元模型的状态.本文采用了类似于 C2 的命令式方式定义修改结构的原语,即在 RuntimeArchitecture 类中预定义了一组基本原子操作集,包括 addComponent,addPort,addConnector,addRole,addAttachment,removeComponent,removePort,removeConnector,removeRole,removeAttachment 等,修改体系结构的元接口(即演化行为)可以由一个原子操作或一组原子操作完成.体系结构设计者可以在设计体系结构类型的时候自定义演化行为:包括定义行为名称、参数以及所涉及的原子操作的组合.比如,在 SimpleCS 类型中定义替换服务器构件的演化行为如下:

```
public void replaceServer (Component S2){
        addComponent (S2);
        removeAttachment (S1.receiverRequest, RPC.callee);
        addAttachment(S2.receiveRequest, RPC.callee);
        removeComponent (S1);
}
```

© 中国科学院软件研究所 http://www.jos.org.cn

为支持体系结构风格的演化,在 *RuntimeArchitecture* 类中定义了“*public void upgrade*(*String subType*)”操作(见第 2.2.2 节),该操作和修改结构的原语一样,亦属于原子操作;

(5) 自适应规则:为了支持系统在无人干预情况下的自主演化,可以为定义了演化行为和属性监听器的体系结构元模型添加自适应规则,规则的形式为“Do action when condition”.以图 4 为例,可定义自适应规则:“*Do replaceServer when responseTime*>1000”,表示响应时间超过 1000ms 时将抛出一个事件触发体系结构元模型执行“replaceServer”操作.体系结构元模型自适应的框架如图 5 所示.

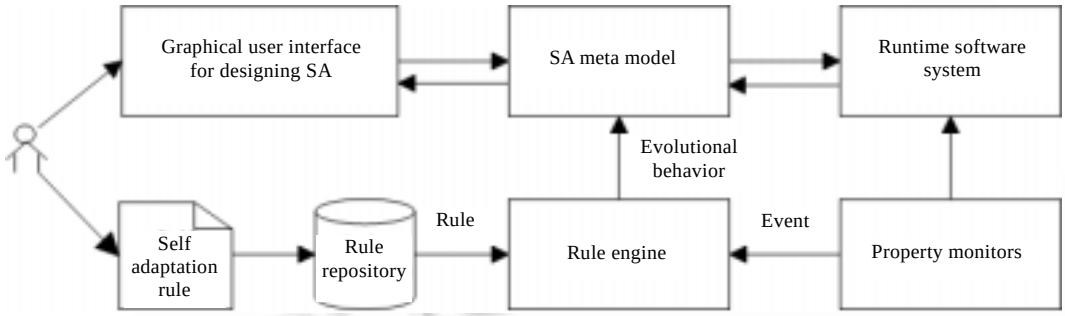


Fig.5 DSA oriented self-adaptation framework

图 5 面向 DSA 的自适应框架

2.2 在线演化的实施

上述体系结构元模型如何在运行时刻影响系统的实际结构呢?下面就预设和非预设两类演化行为分别阐述其基本原理和实现方法.其中,在设计体系结构时考虑到的演化行为称为预设的演化,否则称为非预设的演化.

2.2.1 预设的演化

图 6 以 SimpleCS 为例,每一个基层(base level)的组件在元层(meta level)都有一个元对象与之对应(比如,与 Client 构件对应的是 Component 对象 C),元对象中包含了基层组件的自描述信息.这些元对象按照体系结构规约中定义的拓扑结构建立起关联关系,生成与其体系结构风格对应的体系结构类的对象(图中简称为 ArchAgent,其类型为 SimpleCS),该对象通过反射和动态代理机制将基层构件的交互解耦[14].举例来讲,Client 构件在运行时刻无须知道为它的请求接口提供服务的实际构件,而是由运行系统将其请求转交给体系结构对象,再由体系结构对象负责从当前体系结构上下文中查询与它绑定的 Server 构件.当体系结构对象发现 Client 构件的元对象 C 连接的是 *S*1 时,它将根据 *S*1 找到对应的基层构件 *Server*1,继而将请求转发给 *Server*1,从而实现了 Client 和 Server 之间面向体系结构的松耦合.当体系结构对象在运行时刻被预设的演化行为“replaceServer(*S*2)”修改(图中与 C 相连的构件被换成 *S*2),在新的体系结构上下文的重新解释下,Client 和 *Server*1/*Server*2 之间的调用关系被改变(即当 Client 再次发出请求时将得到 *Server*2 的响应),从而完成了一次在线演化.这里并不限制由谁对体系结构对象实施演化行为,可以是系统管理员,也可能是系统自身,后者即是我们前面所讨论的自适应演化.

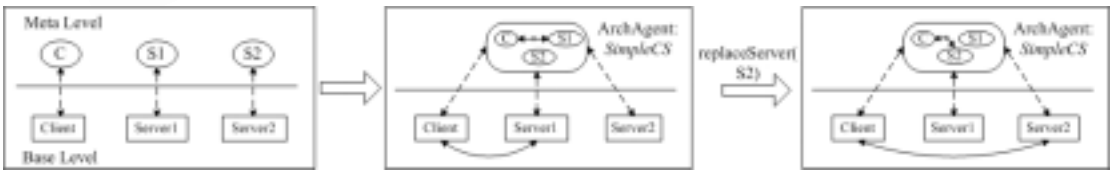


Fig.6 Anticipated evolution

图 6 预设的演化

© 中国科学院软件研究所 http://www.jos.org.cn

2.2.2 非预设的演化

系统在运行过程中可能出现体系结构设计初始未曾预料的情况,这时,当前体系结构类型中预设的演化行为已不能满足系统演化的需求,若要实现非预设的演化则需要增加新的演化行为.使用面向对象方法实现的体系结构元模型可以利用“继承”和“多态”完成体系结构类型的扩展.具体来讲,原体系结构类可以衍生出新的子类,在子类中定义新的演化行为.为了能够执行新定义的演化行为,需要用当前 ArchAgent 的状态初始化新类型的 ArchAgent,新的 ArchAgent 可以执行新定义的演化动作,此过程被称为“升级”(即体系结构风格的衍生).例如,SimpleCS 风格规定 Client 构件仅能和一个 Server 构件相连,并且只预设了一种演化行为,即“replaceServer”.由于负载平衡的需要,希望能实现 Client 构件和多个 Server 构件相连,因此,要对 SimpleCS 类型进行扩展,以增加新的演化行为“addServer”和“deleteServer”,子类型 ComplexCS 定义如下:

```
public class ComplexCS extends SimpleCS{
    …
    public void addServer(Component server){
        …
        addComponent(server);
        Role newRole=new Role(Role.Callee);
        addRole(conn, newRole);
        addAttachment(server.port, newRole);
    };
    public void deleteServer(Component server) {…};
}
```

图 7 给出了该场景下的非预设演化过程.当前的体系结构类型不能满足演化需求时 ArchAgent 执行“upgrade”操作,生成一个新的具有相同状态的 ComplexCS 类型的 ArchAgent,随后与预设的演化过程一样可以对该 ArchAgent 施加演化行为“addServer(*S*2)”.这样,C 构件就与 *S*1 和 *S*2 同时建立了连接,从而使得 Client 能够同时得到 *Server*1 和 *Server*2 提供的服务.由于属于非预设的演化,这个过程通常需要管理员的干预.从技术层面来讲,非预设的演化只是比预设的演化增加了一次或多次 upgrade 操作;从体系结构层面来看,预设的演化不涉及体系结构风格的修改,而非预设的演化则需要调整体系结构的风格.

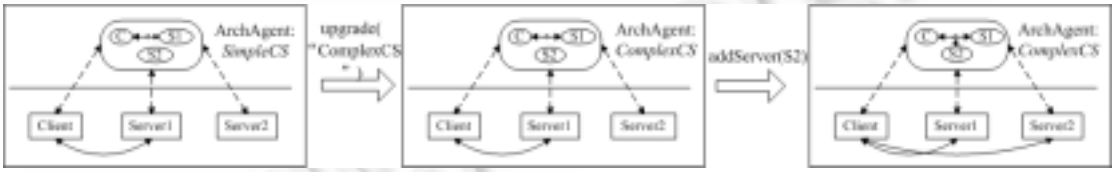


Fig.7 Unanticipated evolution

图 7 非预设的演化

2.2.3 演化的验证

已有工作对体系结构动态修改的验证大都停留在规约层,难以应用到实际系统在线演化中.在面向 DSA 的演化方法中,演化行为不直接针对系统实现,而是通过体系结构元模型的演化来改变运行系统的结构,这样,问题就简化为如何保证体系结构元模型演化前后的“一致性”、“完整性”和“追溯性”.

- 一致性

(a) 类型系统:由于约定只允许体系结构风格沿着衍生路径演化,因此在体系结构对象执行“升级”动作之前,其类型系统将首先检查体系结构对象当前的类型和即将“升级”到的类型,只有在后者是前者的子类时才允许“升级”.类型系统还被用来检查行为的一致性.像前面所提及的“管道-过滤器”风格的结构,如果在运行时刻动态添加了一个过滤器,那么该过滤器的输入参数类型和与它相连的管道的输出结果类型必须保证兼容.若体系

© 中国科学院软件研究所 http://www.jos.org.cn

结构元模型检查发现接口类型不兼容,则会阻止演化,这样,底层运行系统的行为依然能够保证是一致的;

(b) 事务处理:在线演化的一个关键问题是如何保证系统内部状态的一致性以及演化前后系统的状态不会丢失.我们将修改体系结构元模型的原子操作或操作集都当作事务处理单元,由事务管理子系统严格监控体系结构元模型执行演化操作,一旦演化失败便取消所有修改实现状态的回滚.此外,事务处理还可以确保正在被修改的体系结构对象不会同时被其他操作修改.对于分布式应用,由于各个节点上均需要维护一个体系结构元模型,当需要修改时,则通过二阶段事务提交协议[15]保证体系结构元模型一致.

```
BEGIN TRANSACTION;
replaceServer(S2);
if (validate()){
    COMMIT TRANSACTION;
}else ABORT TRANSACTION;
```

(c) 反射架构:面向 DSA 的方法所采用的反射架构一方面保证了运行系统的任何演化都在体系结构元模型的“指导”下进行,即由体系结构元层完成对基层实现系统的修改;另一方面,体系结构元模型与规约之间的因果关联又使得每次演化都被反射回规约,即通过体系结构元模型这个中间桥梁可以保持规约与实现的一致.

- 完整性

体系结构在设计阶段可以通过模型检验工具检查其是否符合体系结构风格中规定的约束.若运行时刻的演化是由管理员修改规约引起的,则仍然可以使用这些工具对体系结构进行约束检查,只有合法的修改才会真正被作用于体系结构元模型,破坏了约束的修改将报错并被阻止.若演化是由运行系统根据自适应规则决定的,则需要体系结构元模型在实施演化行为时进行“自检”.具体地讲,在体系结构类中需要定义检查当前结构是否满足约束的 validate()方法,比如 SimpleCS 风格约束有且仅有一个 Server 构件和 Client 构件相连,其基于一阶谓词表达式的不变式定义如下

$$\exists s \in Component, connected(client, s) \wedge (Connected\ Component(client)).size() = 1.$$

相应的 SimpleCS 体系结构类则将该不变式具体化为 validate()方法,若符合约束,则返回 true;否则,返回 false.目前,该方法由体系结构类的设计者进行定义,我们在进一步的工作中将考虑提供代码模板及根据不变式自动生成 validate()方法体的工具.

```
public boolean validate(){
    Vector connectedComponents=client.ConnectedComponents();
    if (connectedComponents==null||connectedComponents.size()!=1){
        return false;
    }else return true;
}
```

通过检验的演化行为何时执行才能保证体系结构元模型是“安全”的呢?当基层实现系统和体系结构元模型之间存在交互或消息传递时,被认为是“不安全”的状态(比如,当图 6 中 Client 构件发送消息给 ArchAgent 时),此时不能对体系结构元模型执行修改操作,修改操作将被阻塞,直到此次交互结束为止.相应地,当体系结构元模型正在演化时,基层构件发送给它的消息也会被阻塞,直到演化完成.

- 追溯性

软件系统是在体系结构元模型基础上进行的自顶向下的开发,而系统的在线演化行为又都离不开体系结构元模型的指导,即每次演化都必须通过体系结构元模型进行验证.此外,体系结构元模型在运行时刻的每次演化操作均可以被记录下来,以便维护人员对软件进一步分析.以体系结构为中心的软件生命周期保证了软件的体系结构从设计、实现到运行阶段均能够被追溯.

© 中国科学院软件研究所 http://www.jos.org.cn

3 支撑平台和应用实例

3.1 支撑平台Artemis-ARC

本文提出的面向DSA的在线演化方法已经被应用在以构件和Web服务为主要计算单元,以RPC,RMI/IIOP和SOAP为基本通信协议的应用系统集成领域,支持面向DSA的构件/服务组装和演化[14,16],并相应开发了支撑系统Artemis-ARC.该系统基于工业界广泛采用的Apache Axis和JBoss平台,支持Web Service, EJB等的设计、开发和组装,并提供了包括图形化界面开发、基于属性图文法的体系结构检查、代码生成(目前仅限于提供框架性的代码,比如体系结构图中构件的Port对应于实现代码中的方法接口,我们将按照体系结构设计者对Port的属性定义对该接口进行造型)等工具集以及设计运行时刻体系结构类的APIs.系统的设计框架如图8所示.

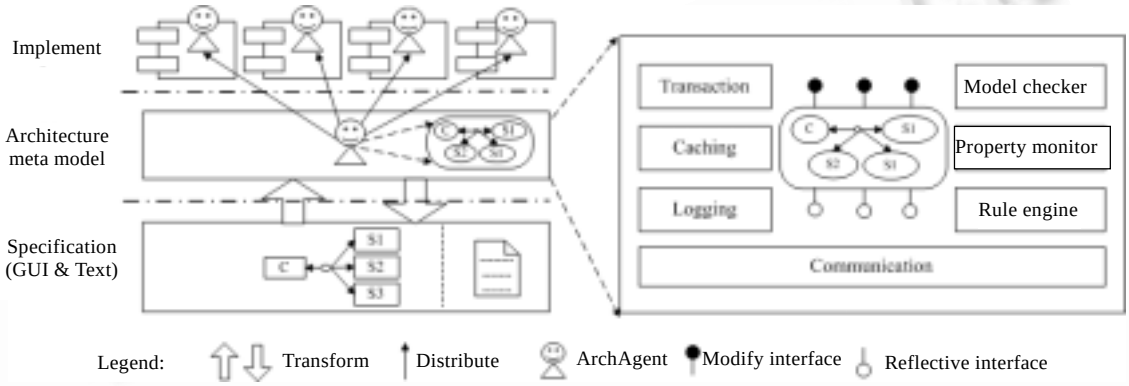


Fig.8 Artemis-ARC system design framework

图8 Artemis-ARC系统设计框架

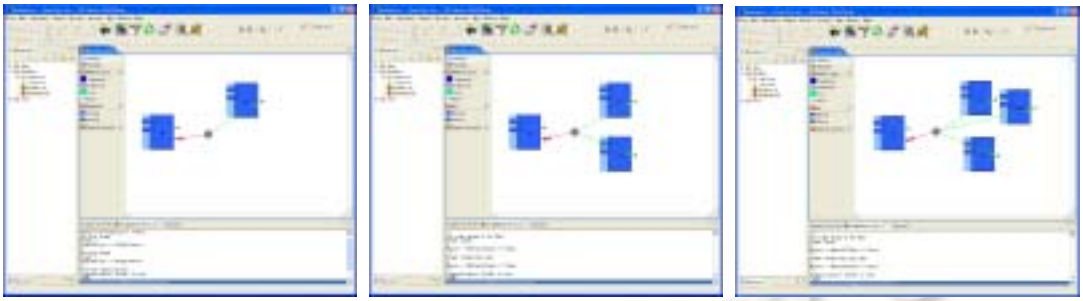
图8左部的两条水平虚线将系统框架分为3层,其中:底层是基于GUI和文本描述的体系结构规约层,中间是体系结构元模型层,两者之间可以进行相互转换;上层是基于构件/服务的实现层,构件之间通过体系结构实体(ArchAgent)建立松耦合式的交互.体系结构元模型层中集成了通信、事务处理、缓存、日志、模型检验、属性监视、规则引擎等模块,核心部分“体系结构元模型”包含了修改体系结构的接口和对体系结构进行查询等操作的反射接口.面向DSA的构件组装/服务集成与以往基于体系结构的开发方法不同之处在于,体系结构规约在这里被具体化为运行时刻可操纵的体系结构实体,构件/服务之间通过体系结构视图间接耦合,可实现集成应用面向DSA的在线演化.对于分布式应用,体系结构对象被分发到构件所在的服务器上组成一个分布式的体系结构元模型空间,共同维护系统运行时刻的体系结构视图.Artemis-ARC目前只支持对体系结构的追溯、验证以及对框架代码的检查,对单个构件内部代码的验证则超出了本文的工作范围.

3.2 应用实例

为了展示Artemis-ARC系统对在线演化的支持,我们开发了一个简单的应用实例.该实例的主要功能是进行连续的分形计算.我们首先采用了SimpleCS类型,其SA设计图如图9(a)所示.在这个应用中,Client构件只负责接受对一个给定大小的图进行分形计算的任务,它需要请求Server构件提供分形计算的服务.当Client接收到的计算请求不断增长时,Server构件的计算量将急剧增长,为了提高服务质量需要在不影响正在进行的计算的情况下及时调整应用的结构实现负载平衡.在该例中可以采用预设的演化动作“replaceServer”重绑定到高性能的备用服务器或者采用“升级”体系结构类型的非预设演化方法,比如将SimpleCS类型升级为ComplexCS类型,这样可以通过对Client任务的分解使多个Server能同时执行计算.通过验证的演化动作提交后,体系结构对象将类型从SimpleCS升级为ComplexCS,升级成功后可以执行新类型中的预设演化动作“addServer”,如图9(b)所示.从分形图的演示效果可以证明体系结构的修改过程没有影响应用的正确执行.为了验证在线自适应演化的功能,我们还定义了自适应规则“**Do** *addServer* **when** *responseTime*>12000”,由体系结构元模型的属性监测器

© 中国科学院软件研究所 http://www.jos.org.cn

获取响应时间,在响应时间超过 12 秒时增加 Server 构件,其中任何一次演化动作都会反射到体系结构规约的 GUI 视图中,图 9(c)显示了体系结构视图和运行系统的一致性.



(a) SimpleCS (b) ComplexCS (c) After self-adaptation

(a) SimpleCS 规约图 (b) ComplexCS 规约图 (c) 自适应后的规约图

Fig.9

图 9

4 相关工作

文献[17,18]利用 UML 类图对体系结构风格进行建模,利用图转换规则对动态行为进行分析和验证,这与本文所采用的 ACME 和属性图文法相结合的方法较为类似.但是,他们的工作依然集中在软件系统的设计阶段,当软件投入运行后体系结构信息已被隐藏和分散在各个构件之中,无法再显式地利用体系结构信息进行模型检验,在线演化行为难以控制.因此,本文提出把体系结构信息具体化为运行时刻可见的体系结构元模型,将体系结构贯穿于软件的整个生命周期.

在基于体系结构信息实现系统的自适应演化方面,具有代表性的工作包括 CMU 的 Rainbow[7]和 UCI 的 ArchStudio[8],其中:Rainbow 使用外置的体系结构管理器,对体系结构的修改需要借助特定于应用的模型管理机制和属性探测机制,以使真实系统和体系结构之间保持一致;ArchStudio 则基于多层消息总线式的体系结构风格,利用一个外部的体系结构演化管理器实现对体系结构修改行为脚本的解释和执行.与它们不同的是,本文提出的体系结构元模型内置于运行系统中,能够充分利用面向对象语言中的类型、继承、多态等机制,为系统的在线演化提供一个通用的可复用的框架,因此我们称这种方法为“面向动态软件体系结构”的演化以示区别.

近年来亦有研究者开始尝试将反射、AOSD (aspect-oriented software development)等方法应用到软件演化中.Cazzola 等人认为,反射为软件设计和开发所提供的透明性、关注分离、可扩展性等有助于实现一个更清晰的演化架构,反射的结构使得运行系统能够对非预期的行为做出调整,从而可以支持持续的演化[19];北京大学开发的 PKUAS 系统是一个基于体系结构的反射式 J2EE 应用服务器,主要侧重于中间件系统的动态调整[4,20].AOSD 允许开发者使用模块化的方式描述系统的不同关注点及其关系,以往分散在代码中的贯穿特性被独立封装成 Aspect,这种松耦合的方式在提高复用性的同时亦有利于软件的演化[21].从 Aspect 的角度来看,本文将体系结构信息作为软件的一种特殊的贯穿特性,通过封装了体系结构信息的体系结构对象将构件之间的交互分离开来,实现了一种灵活的基于运行时刻体系结构上下文的构件松耦合配置方式,为系统提供了可在线演化的前提.从反射的角度来看,体系结构元模型正是传统反射架构中的元层,具有表述和推理基层、修改基层结构和行为的能力,其中与规约层的因果关联保证了规约始终和运行系统保持一致,并可实现演化的追溯,与系统实现层的因果关联则完成了面向 DSA 的在线演化.

5 结束语

计算环境和用户需求的改变,使得人们在设计软件的体系结构时开始关注软件的动态特性.如何在体系结构的描述中刻画这种动态特性目前已有不少研究成果,但对于如何真正在系统运行时刻进行演化,如何保证演

© 中国科学院软件研究所 http://www.jos.org.cn

化行为的实施不破坏系统的完整性和一致性,如何实现动态体系结构的追溯性,目前还没有一个完善的解决方法.为此,本文从软件体系结构的角度给出了一个面向 DSA 的系统在线演化架构,提出以体系结构为中心的软件设计、开发、演化方法;以 ACME 语义为基础设计了运行时刻体系结构元模型,通过可编程的体系结构元模型将系统中的构件的交互解耦,实现基于体系结构上下文的松耦合;利用反射技术和编程模型中的设施使所有演化都在运行时刻体系结构的指导下进行可保证系统演化前后的一致性、完整性和追溯性.

为在真实的软件系统中应用上述技术,开发了面向 DSA 的构件组装/服务集成和演化支撑系统 Artemis-ARC.该系统支持从基于构件和服务的集成应用的体系结构设计到实现、部署、运行、演化等一系列流程,由体系结构元模型在系统运行时刻提供的动态体系结构视图指导应用的在线演化.

下一步我们将对体系结构元模型的语义给出一个形式化的规约,并完善 Artemis-ARC 系统开发环境,使其能够适用于更通用的软件开发.

References:

[1] Garlan D. Software architecture: A roadmap. In: Finkelstein A, ed. Proc. of the Future of Software Engineering. New York: ACM Press, 2000. 91−101

[2] Magee J, Kramer J. Dynamic structures in software architectures. Software Engineering Notes, 1996,21(6):3−14.

[3] Medvidovic N, Rosenblum DS, Taylor RN. A language and environment for architecture-based software development and evolution. In: Proc. of the 1999 Int'l Conf. of Software Engineering. Los Angeles: ACM, 1999. 44−53.

[4] Wang QX, Huang G, Shen JR, Mei H, Yang FQ. Runtime software architecture based online evolution. In: Alexander R. ed. Proc. of the COMPSAC 2003. Dallas: IEEE Computer Society, 2003. 230−235.

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

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

[7] Garlan D, Schmerl B. Using architectural models at runtime: research challenges. In: Oquendo F, Warboys B, Morrison R, eds. Proc. of the 1st European Workshop on Software Architectures. LNCS 3047, St. Andrews: Springer-Verlag, 2004. 200−205.

[8] Oreizy P, Gorlick MM, Taylor RN, *et al*. An architecture-based approach to self-adaptive software. IEEE Intelligent System, 1999, 14(3):54−62.

[9] Allan R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures. In: Astesiano E, ed. Proc. of the Fundamental Approaches to Software Engineering. LNCS 1382, Lisbon: Springer-Verlag, 1998. 21−37.

[10] Aldrich J, Chambers C, Notkin D. ArchJava: Connecting software architecture to implementation. In: Proc. of the ICSE 2002. Orlando: ACM, 2002. 187−197.

[11] Maes P. Concepts and experiments in computational reflection. In: Meyrowitz NK, ed. Proc. of the OOPSLA'87. Orlando: ACM, 1987. 147−155.

[12] Garlan D, Monroe RT, Wile D. ACME: An architecture description interchange language. In: Johnson JH, ed. Proc. of the CASCON'97. 1997. 169−183. http://www.cs.cmu.edu/~able/publications/acme-cascon97/

[13] Ma XX, Zhang XL, Lu J. Description and implementation of dynamic software architectures: A reflective approach. Journal of Nanjing University (Natural Science), 2004,40(2):146−155 (in Chinese with English abstract).

[14] Ma XX, Yu P, Tao XP, Lu J. A service-oriented dynamic coordination architecture and its supporting system. Chinese Journal of Computers, 2005,4(28):467−477 (in Chinese with English abstract).

[15] Tanenbaum AS, Steen M. Distributed Systems Principles and Paradigms. New York: Prentice Hall, 2002.

[16] Yu P, Ma XX, Lu J. Dynamic software architecture oriented service composition and evolution. In: Gu N, Wei DM, Xie ZP, eds. Proc. of the CIT 2005. Shanghai: IEEE Computer Society, 2005. 1123−1129.

[17] Baresi L, Heckel R, Thone S, Varro D. Modeling and validation of service-oriented architectures: Application vs. style. In: Proc. of the ESEC/FSE 2003. Helsinki: ACM, 2003. 68−77.

© 中国科学院软件研究所 http://www.jos.org.cn

[18] Kuske S, Gogolla M, Kollmann R, Kreowski HJ. An integrated semantics for UML class, object and state diagrams based on graph transformation. In: Butler MJ, Petre L, Sere K, eds. Proc. of the 3rd Int'l Conf. on Integrated Formal Methods. LNCS 2335, Turku: Spinger-Verlag, 2002. 11–28.

[19] Cazzola W, Chiba S, Saake G. Software evolution: A trip through reflective, aspect, meta-data oriented techniques. In: Malenfant J, Østvold BM, eds. Proc. of the ECOOP 2004 Workshop Reader. LNCS 3344, Oslo: Springer-Verlag, 2005. 118–132.

[20] Huang G, Mei H, Yang FQ. Runtime software architecture based on reflective middleware. Science in China (Series E), 2004,34(2):121–138 (in Chinese with English abstract).

[21] Falcarin P, Alonso G. Software architecture evolution through dynamic AOP. In: Oquendo F, Warboys B, Morrison R, eds. Proc. of the 1st European Workshop on Software Architectures. LNCS 3047, St. Andrews: Springer-Verlag, 2004. 57–73.

附中文参考文献:

[13] 马晓星,张小蕾,吕建.自省的动态软件体系结构描述与实现.南京大学学报(自然科学),2004,40(2):146–155.

[14] 马晓星,余萍,陶先平,吕建.一种面向服务的动态协同架构及其支撑平台.计算机学报,2005,4(28):467–477.

[20] 黄罡,梅宏,杨芙请.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),2004,34(2):121–138.

余萍(1979－),女,江苏泰州人,博士生,主要研究领域为软件体系结构,移动 Agent 技术,Internet 软件技术.

吕建(1960－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件形式化和自动化,软件 Agent 技术及应用.

马晓星(1975－),男,博士,副教授,主要研究领域为 Internet 软件技术,软件体系结构.

陶先平(1970－),男,博士,教授,CCF 高级会员,主要研究领域为 Internet 软件技术,移动 Agent 技术.

© 中国科学院软件研究所 http://www.jos.org.cn