ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.8, August 2008, pp.1881−1892
DOI: 10.3724/SP.J.1001.2008.01881

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

基于图文法的动态软件体系结构支撑环境*

马晓星[1,2+], 曹 春[1,2], 余 萍[1,2], 周 宇[1,2]

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

[2](南京大学 计算机软件研究所,江苏 南京 210093)

A Supporting Environment Based on Graph Grammar for Dynamic Software Architectures

MA Xiao-Xing[1,2+], CAO Chun[1,2], YU Ping[1,2], ZHOU Yu[1,2]

[1](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

[2](Institutue of Computer Software, Nanjing University, Nanjing 210093, China)

+ Corresponding author: E-mail: xxm@nju.edu.cn, http://moon.nju.edu.cn/~xxm

Ma XX, Cao C, Yu P, Zhou Y. A supporting environment based on graph grammar for dynamic software architectures. *Journal of Software*, 2008,19(8):1881−1892. http://www.jos.org.cn/1000-9825/19/1881.htm

Abstract: In this paper, software architectures and architecture styles are modeled with attributed typed graphs and graph grammars respectively. Accordingly, dynamic reconfigurations of software architectures are modeled with graph transformations. Based on such a modeling, a supporting environment is constructed twofold. Firstly, the visual manipulation of the graphical representation of software architectures is supported with a graph grammar-enabled editor. Secondly, the graphical architecture model is reified as a runtime software architecture object built into the physical running system, through which graph transformations of the architecture model is then naturally reflected as dynamic reconfigurations of the running system.

Key words: dynamic software architecture; visualization; graph grammar; development environment

摘 要: 使用类型化的属性图及其图文法来直观而形式地刻画软件体系结构和体系结构风格,用图转换来刻画动态体系结构的重配置行为.基于这种刻画,构建了一个动态软件体系结构支撑环境.该环境一方面,通过一个基于图文法的编辑器来支持体系结构图模型的可视化构造和操纵;另一方面,基于内置运行时体系结构技术实现了体系结构图模型在具体系统中的物理实施,并使得图模型上的图转换操作可以自动映射到实际系统的动态重配置上.再加上一系列的辅助设施,形成了一个较为完整的基于图文法的动态软件体系结构支撑环境.

关键词: 动态软件体系结构;可视化;图文法;开发环境

中图法分类号: TP311 **文献标识码**: A

软件体系结构刻画了一个软件系统的构件组成、构件之间相互连接的方式、所形成的系统结构配置及其

* Supported by the National Natural Science Foundation of China under Grant No.60736015 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2006AA01Z159, 2007AA01Z178 (国家高技术研究发展计划(863)); the Fund for Creative Research Groups of NSFC under Grant No.60721002 (国家基金委创新研究群体项目); the Program for New Century Excellent Talents in University of China under Grant No.NCET-07-0419 (新世纪优秀人才支持计划)

Received 2008-01-09; Accepted 2008-04-18

动机、约束和模式等[1].作为系统开发过程早期的一个核心制品,软件体系结构为在一个较高的抽象层次上分析验证系统性质,以指导后续的系统开发提供了依据.经过十五六年的发展,软件体系结构的相关观念和技术已经逐渐为广大开发者所接受[2].此时,相关支撑工具,特别是图形化、集成化的体系结构支撑工具的研究成为一个重要问题.

同时,随着越来越多的软件系统需要面对开放的网络环境和变化的用户需求,它们需要在运行时刻动态调整其体系结构配置,以满足应用需求.这种动态软件体系结构[3]对支撑工具提出了更高的要求.首先,支撑工具不仅要能表达特定的体系结构配置,还要表达体系结构的动态重配置行为.其次,支撑工具需要保证其所维护的体系结构模型和真正运行中的系统的体系结构之间的一致性.换言之,该支撑工具应能将体系结构模型上的修改忠实地反映到运行中的系统中去.

一个较为理想的动态软件体系结构支撑工具应满足下列要求:首先,对软件体系结构模型的视觉表达比较直观,否则难以广为接受;其次,有严格的语义以支持抽象的分析,最好是基于较为成熟的形式化工具;第三,能够支持规则驱动的可视化构造和操纵;第四,有自动化的工具支持;最后,要有利于体系结构的实现,尤其是有利于其动态重配置的实施.当前虽然已有不少软件体系结构的支撑工具,如 ACME Studio[4],ArchStudio[5],Archware[6]等,它们均支持图形化的表示,也有较为严格的形式语义基础,但它们多着重于体系结构规约构造与分析阶段的支持和部分代码的自动生成,而对动态重配置实现的支持不足,或者局限于特定的体系结构风格.此外,它们在对用户定义的体系结构风格下的规则驱动的体系结构可视化构造和操纵方面尚不能提供令人满意的支持.

本文中我们使用一种基于图文法(graph grammar)和图转换(graph transformation)[7]的途径来构造动态软件体系结构支撑系统.实际上,十余年前 Le Metayer[8]就提出使用图文法来刻画软件体系结构风格,但缺乏相应的能够满足上述考虑的基于图文法的动态软件体系结构支撑系统.图文法作为一种较为成熟的形式化工具,有较为完善的理论支持,可满足体系结构建模和分析的需要,特别是,图文法直接以图为主要处理对象,与其他工相比,具具有更好的直观性.同时,图文法是一种基于规则的系统,可以方便地刻画动态体系结构的重配置操作行为.作为一种元级语言,图文法直接支持用户定义的图语言,从而直接支持定制的体系结构风格和该风格特定语境下的体系结构构造和操纵.具体而言,我们首先使用类型化的属性图及其图文法来直观而形式地刻画软件体系结构和体系结构风格,用图转换来刻画动态体系结构的重配置行为.而后,一方面,基于属性图文法系统 AGG(attributed graph grammar)[9]和 Eclipse 图形编辑框架 GEF(graphical editing framework)等既有软件包实现了体系结构图模型的可视化编辑构造和操控;另一方面,基于内置运行时体系结构技术实现了体系结构图模型在具体系统中的物理实施,并使得图模型上的图转换操作可以自动映射到实际系统的动态重配置上.再加上一系列的辅助设施,形成了一个较为完整的基于图文法的动态软件体系结构支撑环境.

本文第 1 节讨论如何使用类型化属性图及图文法来描述软件体系结构和体系结构风格.第 2 节讨论基于图转换的体系结构重配置规约和实施.第 3 节给出支撑环境的设计原理和若干技术要点.第 4 节介绍我们实现的一个集成支撑环境原型系统.在给出全文总结之前,第 5 节讨论一些相关工作.

1 基于图文法的软件体系结构描述

首先来讨论如何将软件体系结构的相关概念用图文法这一工具直观而严格地进行表达.本文的体系结构相关术语和概念遵从 ACME[4]这一较为通用的体系结构描述语言的约定.对软件体系结构的刻画包括体系结构实例、体系结构风格及其属性约束等方面的刻画.我们将使用类型化的属性图及相应图文法[10,11]来刻画这些概念.需要说明的是,属性图文法是一个严格的形式化工具,使用它来刻画软件体系结构,其目的在于利用既有的图文法工具构造动态软件体系结构支撑环境.出于可读性和篇幅上的考虑,对于某些图文法相关概念,本文偏重于直观的解释,严格形式化的定义和讨论可参见 Ehrig 和 Taentzer 等人的工作[7,10,11].

1.1 体系结构实例的刻画

软件体系结构实例是由一组构件、连接子以及构件的端口和连接子的角色之间的绑定而构成的一个拓扑

配置,而这些体系结构元素可有属性.因此,体系结构实例直观地描述为一个属性图.由于一个构件或连接子可能有多个属性,故首先采用一种称为 E-Graph[11]的结构来表示体系结构的拓扑配置: $G=(V_G,E_G,V_D,E_{NA},E_{EA},(source_j,target_j)_{j\in\{G,NA,EA\}})$,其中:

- V_G 是图中节点的集合,用来代表体系结构实例中的构件和连接子;
- E_G 是图中边的集合,用来代表体系结构实例中的构件端口和连接子角色之间的绑定;
- V_D 是"数据节点",也就是属性的集合,用来代表体系结构实例中涉及到的各属性值;
- E_{NA} 是图中的节点到数据节点的"属性边"的集合,代表体系结构实例中各构件和链接子的属性;
- E_{EA} 是图中的边到数据节点的属性边的集合,代表体系结构实例中各端口、角色的属性;
- $source_G,target_G:E_G\to V_G$ 是边的起点和终点函数;
- $source_{NA}:E_{NA}\to V_G,target_{NA}:E_{NA}\to V_D$ 是节点的属性边的起点和终点函数;
- $source_{EA}:E_{EA}\to E_G,target_{EA}:E_{EA}\to V_D$ 是边的属性边的起点和终点函数.

两个 E-Graph G 和 H 之间可能存在一种态射(morphism)关系,即可将 G 的节点(包括数据节点)映射到 H 的节点,G 的边(包括属性边)映射到 H 的边;且这种映射能够保持边的起点和终点关系,即 G 中某边 e 的起点(终点)v 映射到 H 中为 v',则 v'必然是 e 映射到 H 中的边 e'的起点(终点).

属性图(attributed graph)$AG=(G,D)$在上述 E-Graph G 的基础上还引入了一个代数(algebra)D[10].这个代数基于一个数据基调(data signature),提供了一种数据抽象设施,从而使我们可以刻画属性间的结构(或称有结构的属性)以及其上的操作.这意味着在实现级可使用自定义的对象作为属性,这对于运行时体系结构的刻画是十分重要的.属性图之间也可以有态射关系,该关系除了要求相应的 E-Graph 之间有态射关系以外,还要求相应的代数之间有能够保持相应特性的代数同态关系.

1.2 体系结构风格的刻画

直接刻画软件体系结构实例并不方便,也不利于复用.如果可能,通过体系结构风格来定义具体的体系结构实例是更好的方式.体系结构风格刻画了一类体系结构,如管道/流水线、黑板系统、主从结构等.体系结构风格定义一组构件类型和连接子类型,并给出组合这些构件和连接子的规则[1].为了刻画体系结构风格,我们首先通过属性图的类型化来刻画一个体系结构实例中的构件和连接子的类型化,再通过图文法来进一步刻画体系结构的拓扑配置约束.

体系结构中构件和连接子的类型实际上是对其属性和所属拓扑结构两方面的约束.例如,在一个 Master/Slave 风格中的 Master 构件将有一个表示工作负载的 workload 属性;而它将通过连接子与 1 个或多个 Slave 连接.这种约束可以通过属性图的类型化来表达.图的类 v'型化在直观上是赋予图的每个节点和边一个类型,而这些类型之间存在着预先设定的一些关系,于是,图中的节点和边也必须满足这些关系.理论上,这些类型与类型之间的关系也是通过一个图 TG 来表达的,这个图称为"类型图(type graph)".一个图 G 的"类型化(typing)"也就是一个从 G 到 TG 的态射 $type:G\to TG$.图 1 所示为一个简单的 Master/Slave 体系结构的类型化属性图的表示.左边的图是 E-Graph 形式的类型图,表示 Master,Slave 这两个构件类型、Mux 这个连接子类型之间的关系,以及它们的属性名和类型.中间的图是该类型图的一种紧凑表示,各元素的属性名和类型直接标注在元素图元中.右边的图是一个 Master/Slave 体系结构实例.

上述类型化的手段仅能表达对元素属性及元素间局部关系的约束,无法刻画体系结构风格的拓扑结构的约束.例如,在 Master/Slave 风格下,我们要求一个系统有且仅有 1 个 Master 构件和 1 个 Mux 连接子,有 1 个或者多个 Slave 构件,且各 Slave 构件必须通过 Mux 连接到 Master.此时,我们需要使用图文法来刻画这些约束.简单地说,一个图文法 $GG=(S,P)$由一个开始图 S 和一组产生式规则 P 构成.P 中的一个产生式规则写作 $p=LHS\to RHS$,LHS 和 RHS 都是图.所谓图转换(graph transformation)或图重写(graph rewriting) $G\xrightarrow{p,m}H$ 是指在一个图 G 上,应用规则 p 得到新的图 H,其直观过程是首先在 G 中找到一个与 LHS 匹配的子图 m,将其替换为 RHS,从而得到 H.当然,对于类型化的图文法而言,所有这里涉及到的图均应可用一个类型图来类型化(即可态射到该类型图).有了图文法的概念,我们可以将一个体系结构的所有合法实例集合刻画为该文法生成的一个语言 $L(GG)=$

$\{G \mid S \xrightarrow{*} G\}$,即任何一个体系结构实例要么是开始图,要么可以从开始图应用 P 中的产生式逐步推导出来.

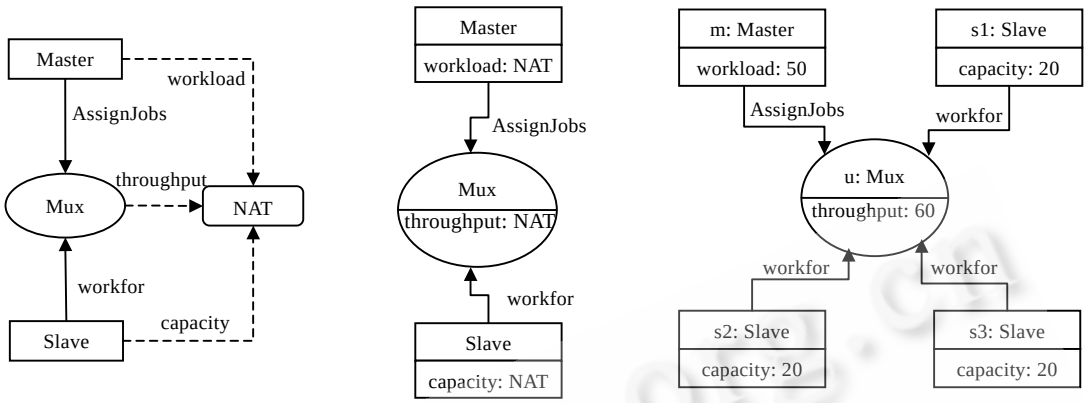

Fig.1 Typed attributed graph for a Master/Slave architecture

图 1 Master/Slave 体系结构的类型化属性图

图 2 所示为 Master/Slave 风格的一种可能的图文法规则,其开始图为空图.规则 p_1 无中生有地生成一个 Master 构件、一个 Slave 构件和一个 Mux 连接子,并恰当地连接起来.规则 p_2 用以生成更多的 Slave 构件.注意,我们期望 p_1 只会被使用 1 次,但该规则的 *LHS* 为空,可以被任意匹配;为了避免创建出多个 Master 和 Mux,引入规则应用条件 *AC*.*AC* 也是一个图,它成立与否取决于当前图是否包含与该应用条件图匹配的子图.应用条件有肯定和否定两种形式.这里,p_1 使用的是否定形式 *NAC*,当原图存在一个子图与该 NAC 匹配时,该规则不得应用.由于 p_1 应用之后必然会存在与该 *NAC* 匹配的 Master 构件节点,故 p_1 将不会再被应用.

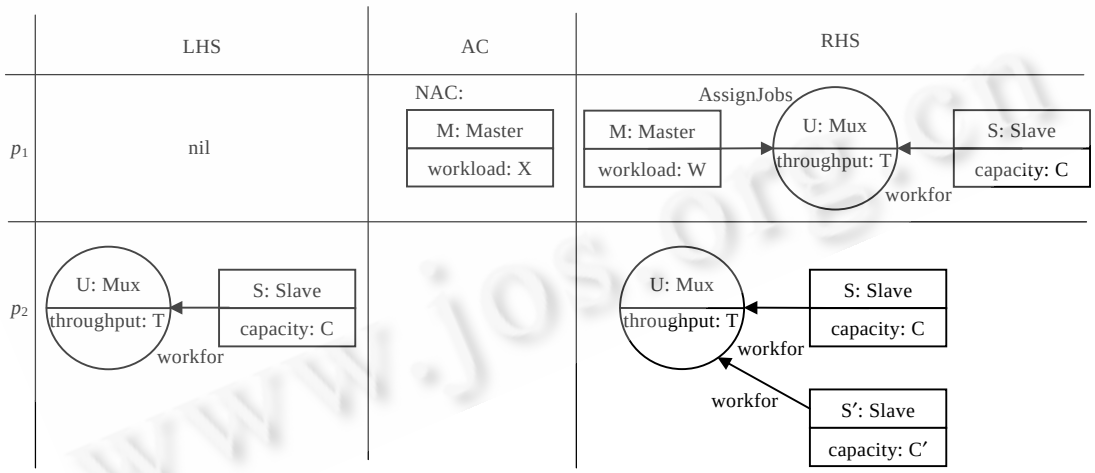

Fig.2 Graph grammar rules for Master/Slave architecture style

图 2 Master/Slave 体系结构风格的图文法规则

通过这种直观而严格的手段,我们可以刻画体系结构风格的拓扑结构约束.这在实际应用中可以有两种使用方式:一种是从开始图出发,严格使用这些规则来构造体系结构实例,如图 1 中右边的体系结构实例可从开始图(空图)出发,应用规则 p_1 一次,再应用规则 p_2 三次得到.这种方式称为语法制导的构造.另一种是反过来,给定一个体系结构实例,我们可以判定它是否满足某体系结构风格.这实际上是一个图的识别(parse)过程,即判定一个图 G 是否属于某图文法 GG 的语言 $L(GG)$.这种方式是自由构造,事后验证.

1.3 体系结构风格属性约束的刻画

体系结构风格还定义了一些属性上的约束,特别是全局性的约束.例如,在上述 Master/Slave 风格中,一个系统总的负载不应超过所有 Slave 的处理能力之和.在图文法的框架下,无法直接判断和验证这种全局约束,因而首先要将这些全局性的约束转化为局部于各条图文法规则的约束.例如,上述约束可以表示为下列两个分别局部于产生式规则 p_1 和 p_2 的不变式:

INV_1: $(Master)M.workload \le (Mux)U.throughput$,即系统负载不超过总的处理能力;

INV_2: $(Mux)U.throughput = \sum_{S \in Slave} S.capacity$,即总处理能力等于各 Slave 能力之和.

进而,需将这些不变式变成附着在产生式规则上的属性操作,以便在通过图文法产生体系结构实例的同时,求出这些约束是否满足的布尔值.

对于 INV_1,处理看起来很简单,p_1 应用时立即可以判断;对于 INV_2,在 p_2 应用时附以属性操作 $U.throughput:=\textbf{old}\ U.throughput+S'.capacity$ 即可.但这里存在一个问题:由于 p_2 是在 p_1 之后应用的,而 p_2 会修改 Mux 的 throughput 属性,此时已无法回头依据 p_1 判断 INV_1.为此,可在 Mux 上也引入一个属性 workload,并在应用 p_1 时进行属性操作 $(Mux)U.workload:=(Master)M.workload$,在每次应用 p_2 时判断 $(Mux)U.workload \le (Mux)U.throughput$ 即可.这可看作是 Knuth 的属性文法的一种应用.

2 基于图转换的体系结构重配置

以上讨论主要针对静态的软件体系结构,但许多系统需要在运行时刻改变其体系结构配置,以适应环境和需求的变化,维持满意的服务.这类动态软件体系结构[12]对软件体系结构的规约和实现技术提出了新的要求.动态体系结构的重配置行为可依据是否在设计时刻已知并在实现中予以支持分为预设的重配置和非预设的重配置两类.从软件工程的角度看,动态软件体系结构的问题可从两个侧面进行讨论:一是如何抽象地刻画体系结构的重配置行为,二是如何实现这种重配置行为.图文法和图转换作为一个直观的具有操作性的形式模型,非常适于表达体系结构的重配置行为.而刻画体系结构重配置的图转换操作的物理意义也可以通过一种运行时体系结构技术直接实现.

2.1 体系结构重配置规约

我们以一个图转换规则 $r = LHS_r \to RHS_r$ 来表示某动态体系结构的一个重配置行为.如前所述,我们以体系结构风格来约束体系结构实例.作为预设的重配置行为应当不超出该体系结构风格(注意,这里的转换规则未必是体系结构相应文法的产生式规则).例如,对于一个 Master/Slave 系统来说,可以预计系统在运行时刻可能根据负载情况增加或减少 Slave 的数目,但重配置后的结果仍符合 Master/Slave 风格.于是,对于一个相应图文法为 GG 的体系结构风格,其任何一个预设的重配置行为相应的规则 r,有

$$\forall H, H'((H \in L(GG)) \wedge (H \xrightarrow{r} H') \to (H' \in L(GG))) \tag{1}$$

而对于非预设的重配置行为相应的规则 r',上述规则可能已不能满足,即生成的 $H' \notin L(GG)$. 例如,一个 Master/Slave 系统随着不断增加 Slave,最终负责分发任务的 Master 本身成为瓶颈;此时需要增加新的 Master.这突破了原有的体系结构风格.基于对新情况的新认识,可能需要定义新的体系结构风格及其相应的文法 GG'.对于该新文法,上述性质重新得到满足:

$$\forall H, H'((H \in L(GG')) \wedge (H \xrightarrow{r'} H') \to (H' \in L(GG'))) \tag{2}$$

实际的演化过程往往是先进行第 1 类的重配置,直到某个时刻未曾预料的事件发生了,此时我们才必须定义新的体系结构风格和图文法,并使当前的体系结构实例成为该新风格一个合法的实例,而后再依据新风格定义的重配置行为,从当前实例出发,获得所需的体系结构配置.刻画这个演化过程的图转换过程是

$$H_0 \xrightarrow{r_1} H_1 \ldots \xrightarrow{r_{j-1}} H_{j-1} \xrightarrow{r_j} H_j \xrightarrow{r_{j+1}} H_{j+1} \ldots \xrightarrow{r_n} H_n.$$

其中 $H_0 \ldots H_j \in L(GG); H_j \ldots H_n \in L(GG'); r_1 \ldots r_j$ 满足上述性质(1);$r_{j+1} \ldots r_n$ 满足上述性质(2).

2.2 体系结构重配置操作实现

与其他动态体系结构的规约方法[3,12]相比,基于图转换的重配置操作具有鲜明的物理意义.例如,对于像图 2 中的第 2 条产生式那样的转换规则,它直接表明了该重配置操作是在系统中增加一个 Slave,而不是像其他体系结构修改语言那样将其分解为一组基本元素的增删动作或一系列难以理解的进程交互的效果.这就提示我们,体系结构的重配置行为应予以某种形式的集中地、显式地实现,而不应分散于系统部件之中.

这一思想导致我们提出了一种所谓内置式运行时体系结构的技术[13−15].使用该技术,我们的体系结构风格实现为一个类,相应的重配置动作实现为该类的方法.该风格的一个具体实例就是该类的一个对象,称为体系结构对象.进而通过用体系结构对象来重新解释构件对象间的引用方式,使得该体系结构对象确实掌控着系统的体系结构.于是,在该对象上调用相应的重配置方法就直接对应于在相应的图上作用以重配置操作.相应地,对于非预设的动态重配置,需要定义新的体系结构风格类(相应于新的图文法 GG'),并以当前的体系结构对象为基础来生成新的体系结构对象(相应于上述 H_j 既属于 $L(GG)$ 又属于 $L(GG')$),而后再在其上应用非预设的动态重配置行为.

这种技术又提示我们可以在某种程度上将动态体系结构的支撑系统看作/实现为图文法和图转换规则的一种解释器,从而直接将图转换作为动态体系结构的操作手段.进一步地,体系结构的动态重配置行为是为了适应某种情况的变化.适应逻辑常可表达为“事件-条件-动作”(ECA)的形式.图转换规则加上它的应用条件恰好可以构成适应逻辑的主体部分.当然,有关自动适应逻辑的设计还会涉及到环境建模、应用领域知识建模以及它们与体系结构模型的互操作等问题,这超出了本文讨论的范围.

3 基于图文法的支撑环境设计

使用图文法和图转换来刻画动态软件体系结构不仅可以提供直观而严谨的规约,还便于我们利用既有的图文法工具包来为动态软件体系结构系统提供有力的支撑.一个动态软件体系结构的支撑环境可能需要对系统的开发、运行、监控、动态重配置提供多方面的支持.图 3 所示为动态体系结构应用系统的开发和演化的宏观过程以及图文法和图转换模型在其中可发挥的作用.

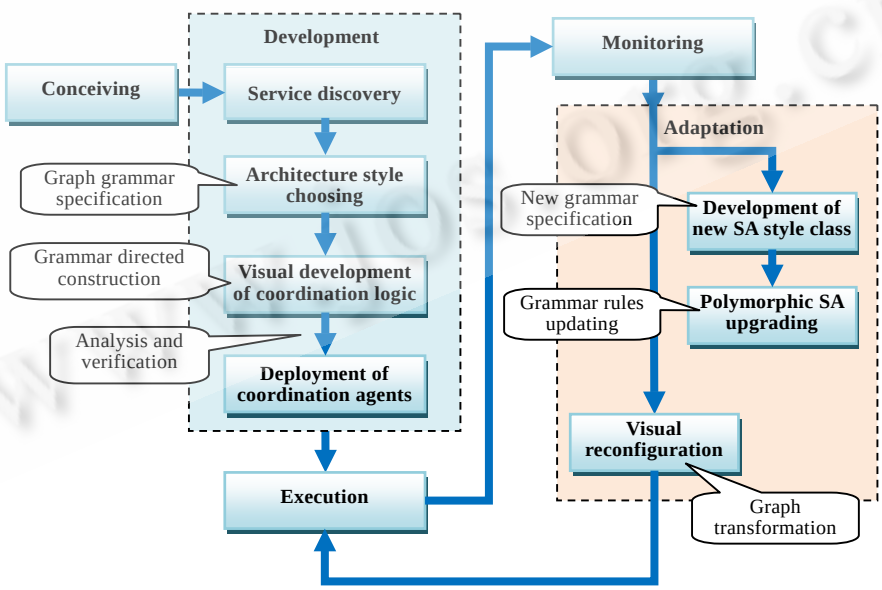

Fig.3 Development and adaptation of systems with dynamic architecture

图 3 动态体系结构应用系统的开发与演化

3.1 设计原理

图文法本身是一种元级语言,它用以定义具体的图语言.因而从图文法模型的角度看,相关支撑工具可视为某种类似于 Lex/YACC 的元语言解释器或翻译器.但与 VisPro[16]等基于图文法的可视语言生成器不同,这里需要一种双重解释机制,如图 4 所示.一方面,用户通过一个可视化编辑器来操纵这个基于图的体系结构模型,这是对图文法和图转换的直接解释;另一方面,该图模型又被 Artemis 系统解释为实际的协同行为,通过协同环境中的构件、服务来达成应用目标,同时,通过可视化编辑器作出的图模型的转换行为也被解释为对实际协同体系结构的在线调整.

为了实现上述双重解释,整个支撑系统采用如图 5 所示的层次结构.该系统的核心为运行时刻体系结构对象.该对象实际上同时用 3 种手段刻画同一个系统当前体系结构配置的概念.这 3 种手段的抽象层次和角度各不相同:基于 ACME[4]的表示代表的是一般的体系结构描述抽象,其主要目的是与其他软件体系结构支撑系统的互操作;基于前文所述类型化属性图的体系结构实例表示使我们可以应用图文法和图转换规则来指导体系结构实例的可视化生成和演化,并支持相关特性的分析;而基于对象程序设计模型的运行时体系结构对象实现则在抽象的体系结构规约和实际的系统之间建立了一种因果互联(causal connection)[14],从而形成一种自省计算(computational reflection)[17]的机制,进而实现系统的动态演化.

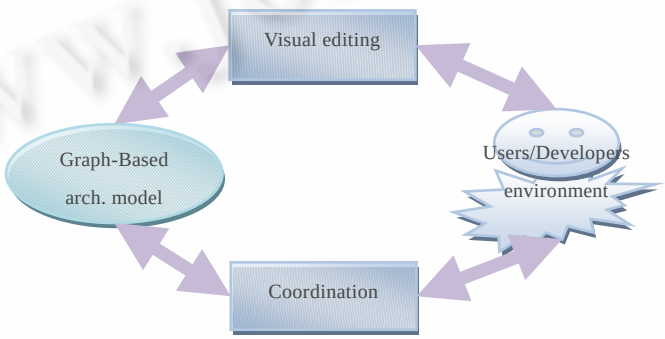

Fig.4 Dual interpretation of the graphical architecture model

图 4 基于图的体系结构模型的双重解释

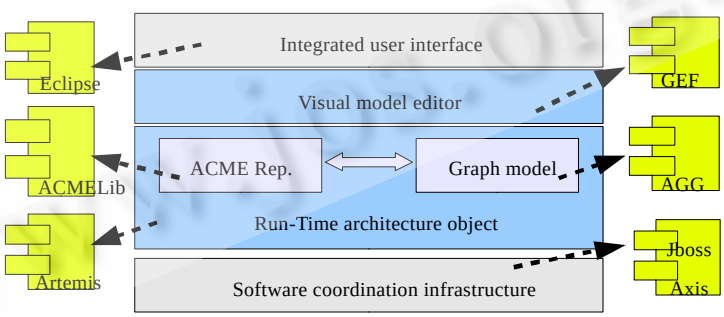

Fig.5 Layered structure of the supporting system

图 5 支撑系统层次结构

3.2 相关技术问题

在可视化的体系结构编辑方面,有两种编辑模式,分别称为自由编辑模式和语法制导的编辑模式.

在自由模式下,用户可以自由选取各种类型的构件和连接子,自由组合,形成任意配置.在编辑完成之后需要检查是否满足给定的体系结构风格约束,也就是判定当前的图是否属于相应图文法的图语言.解决这个问题

的基本思路是,将该文法的每一个规则 $p = LHS \to RHS$ 的左部与右部互换,得到 $p' = RHS \to LHS$,然后在当前图上应用这些规则,以穷举搜索的方式看是否能够找到一条路径,使得可以回到文法的开始图.为了使这个过程能够确定终止,甚至在多项式时间内确定终止,人们提出了种种办法,主要是对文法进行限制,也可以对文法规则的应用顺序和条件加以限制[7,11,16].我们在应用中发现常用的体系结构风格均可利用这些技术以获得满意的效果.

在语法制导的模式下,用户只能从文法给定的开始图出发,每一步可施行的编辑操作仅能在当前可应用的图转换规则中选取.这样可以保证用户编辑的体系结构配置一定满足当前风格.这样做的好处还在于,每一步编辑都有明确的体系结构语义,例如,在一个 Master/Slave 系统中增加了一个 Slave.这种模式要求编辑器能够依据不同体系结构风格相应的图文法调整自己的界面和行为,也就是说,这个编辑器实际上是可依据用户给定文法生成相应编辑器的"编辑器生成器"[16].文献[16]针对 RGG 文法类型给出了一种处理办法,而我们在 AGG[9]和 GEF 的基础上也进行了初步的尝试,结果证明也是基本可行的.

在体系结构的图模型与系统的因果互联方面,一种办法是在真正运行的系统之外引入一种"全能"的配置管理器[18],它能够从实际运行的系统中不断提取最新的软件体系结构信息,同时也能在实际系统上执行在软件体系结构规约层面表述的动态调整命令.然而要找到一种较为通用的配置管理器难度很大.因此,已有的工作要么针对特殊的体系结构风格,要么需为特定应用作特定的、难以复用的开发.我们需要一种更为本质的、深入计算模型层面的机制来实现这种因果互联.

在实现级,当前的各种主流构件框架的协同机制仍未脱离(分布)对象计算模型.在这一模型下,(分布)对象之间的(远程)方法调用确定了系统的结构.面向开放、动态、难控的环境,这种对象方法调用抽象(表述为"*O.m*()")显现出以下不足:首先,协同机制固化,难以编程调节;其次,用户协同逻辑隐藏在计算代码之中,而且分散在系统各处;最后,这种机制丢失了决定系统结构之所以如此的语义信息,从而在需要进行体系结构调整时难以着手.为此,我们对该机制进行了重新解释,对象引用"*O*"不再指向某个具体的对象,而是当前体系结构下的某个需要动态决定的角色;方法"*m*"也不再意味着简单的方法调用,而是可以根据需要动态调整的协同模式.

我们以体系结构图模型为核心来实现这种重解释.实现上,首先将体系结构风格实现为一个面向对象语言的类,该风格所支持的动态重配置行为实现为该类的修改方法.于是,在体系结构图上应用图转换规则也就同时在相应体系结构对象上调用相应的方法.也就是说,我们不仅在可视化编辑器中依据文法解释执行相应的操作,我们还在这个对象上解释执行相应的操作.当然,这个对象的类的具体行为语义需要定义体系结构风格的用户在给定相应图文法时指定.为了确保这个对象能够真正体现系统当前的体系结构,我们让该对象来动态地解释构件之间的对象引用和交互行为.这样,在这个对象上的结构配置修改行为就直接反映到运行的系统上.这个体系结构对象逻辑上是集中的,但物理上可实现为一个分布共享对象.有关这种内置运行时体系结构对象技术的详细讨论不是本文的重点,可参见文献[13–15].此外,有关如何感知环境的变化,并依据环境的变化来驱动系统的动态演化的问题,我们将另文加以讨论.

4 ARTEMIS支撑系统原型

我们在面向服务的动态协同环境 ARTEMIS-ARC[13]中使用了前述基于图文法和图转换的软件体系结构刻画技术,设计并原型实现了相应的基于图文法和图转换的支撑系统 ARTEMIS-AGG.该环境的可视化集成开发界面如图 6 所示.它支持以图形化的方式定义集成系统的软件体系结构,以此作为系统集成或协同的逻辑,并据此生成内置于应用系统之中的运行时刻体系结构对象,作为应用系统运行、演化及自适应的基础.并且它还集成了服务搜索、基于图文法的体系结构约束检查,运行时监控、QoS 建模以及自适应规则定义等诸多功能,为服务集成系统的开发、运行、监控和动态演化提供了一个一致、易用、易扩展的支撑平台系统原型.

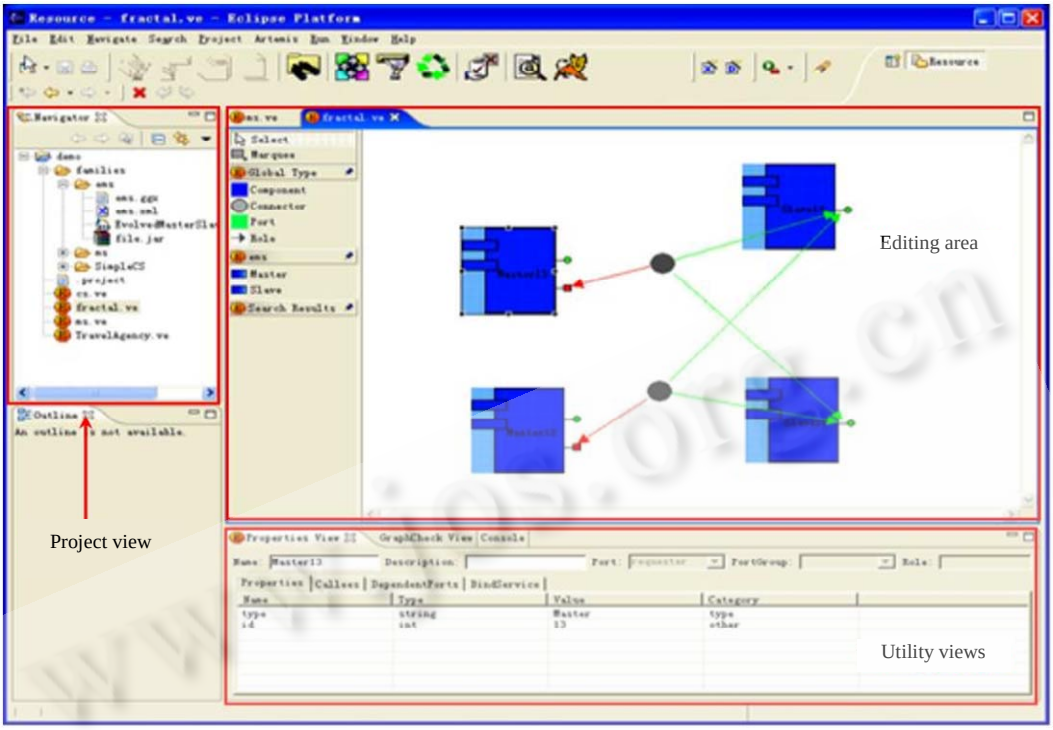

Fig.6 Visual integrated development environment of ARTEMIS

图 6 ARTEMIS 可视化集成开发环境

在该系统的实现过程中,我们尽量复用既有的标准软件包(如图 5 所示).首先,我们将其所有功能以插件形式整合到 Eclipse 环境当中,这样,可以利用这一优秀平台提供的通用功能;其灵活、开放的架构也便于今后对系统功能的扩展和与其他系统的互操作.在软件体系结构描述方面,使用来自美国 CMU 的 ACME Lib 软件包.在图文法引擎方面,使用了德国柏林技术大学的 AGG 软件包.可视化的编辑工具实现乃是基于 GEF 软件包,可与 Eclipse 环境紧密集成.最后,为了能够利用网络环境中的 EJB 构件和 Web Services,将我们的协同系统构建在 JBoss 和 Axis 系统之上.

在使用 ARTEMIS 支撑环境进行应用系统的开发时,用户首先载入既定义的软件体系结构风格模板,显示于图 6 中的项目视图区中.该模板包含该风格所对应的图文法规则、相应的体系结构类和该风格下的各种构件和链接子定义.代表构件和链接子的图元将出现在编辑区左侧的工具条中.用户可以使用这些图元在编辑区中创建一个体系结构配置.同时针对各个图元代表的构件和链接子,可以在视图区指定、编辑其相关属性.然后,可通过服务/构件搜索发现工具来获得可用的物理服务/构件,并将其绑定到当前体系结构的构件图元上.一个体系结构配置实例创建完成以后,系统将依据给定的图文法规则来检验它是否符合给定的风格.检验通过以后,就可以将整个应用系统提交部署.由于所使用的物理服务/构件是既已存在的,这里部署的实际上是应用系统的协同逻辑.ARTEMIS 支撑环境将依据体系结构风格类自动生成体系结构对象.该对象是一个分布共享对象,它基于当前体系结构配置,为各个物理服务/构件的运行提供一个协同上下文.

当发现应用系统需要进行在线的重配置时,如果这种重配置是该体系结构风格本身所支持的,如 Master/Slave 风格下增加一个 Slave,则可以直接在编辑区上施行相应的操作.编辑器的图文法检查系统将确保这种操作是该风格的图文法中定义的合法规则.同时,在当前体系结构对象上调用定义在体系结构类中的相应演化方法.如果当前体系结构风格不能支持所需的演化,则需要从当前体系结构风格出发,扩展出新的体系结构

风格,给出相应的图转换规则,定义相应的体系结构风格类,以支持相应的演化行为.而后,ARTEMIS 系统可动态载入这个风格,生成新的体系结构对象,并用当前对象加以初始化.在这个新的风格下,再使用前述办法来进行在线的重配置.

在这个系统上,我们开发了若干演示验证应用,包括一个在线综合交通票务系统和一个水利水情会商模拟系统,它们均使用了上述在线演化技术以应对需求和环境的变化.这些实验初步表明,本文所述的基于图文法的动态体系结构支撑技术是可行和有效的[19].

当前我们正在进行新一代的 ARTEMIS-NG 系统的开发.在现有 ARTEMIS-AGG 所使用的自由编辑、事后图文法检查的体系结构构造模式的基础上,ARTEMIS-NG 将提供语法制导的编辑模式.而通过这种语法制导的编辑模式,我们意图结合 van Lamsweerde 等学者在从需求到体系结构转换方面的工作[20],给出一种启发式的体系结构构造支持——实际上,文献[20]中最后给出的体系结构精化模式几乎可直接以图转换规则来表达.

5 相关工作

在基于文法规则的软件工程支撑工具方面,相关工作可以追溯到 20 世纪 70 年代末的康奈尔程序合成器[21].实践表明,在程序源代码一级,对于新手程序员,语法制导的编辑有助于避免无谓的语法错误;对于较熟练的程序员,虽然主要使用自由编辑模式,但高亮提示、格式调整和源码重构等语法制导工具对提高日常工作效率作用明显,已成为现代程序开发环境(如 Eclipse JDE)中不可或缺的部分.在分析和设计阶段,图形化的可视语言(如 UML)成为主流,此时自由编辑更容易导致制品内部的各种不一致和不完整问题.由于缺乏成熟而通用的基本理论工具,基于图形的软件开发过程制品的一致性和完整性检查要比基于文本的更为困难.当前已经出现了一些利用图文法来解决这个问题的工作,如,像 VisPro 这样的可依据图文法生成各种特定可视化编辑器的元语言工具[16]以及基于图文法的 UML 编辑和分析工具[22]等.而像 AGG[9],PROGRESS[23]这样的通用图文法语言和引擎为这类工作提供了有效支持.

在基于图文法的体系结构建模和分析方面,自从 Le Metayer 提出可利用图文法来描述软件体系结构风格[8]以来,Hirsch[24],Kong 和 Zhang 等人[25]也使用不同的图文法模型来描述体系结构风格及其约束,并利用自动化的工具来检测约束是否满足.Wermelinger 等人[26]也使用图转换来规约软件体系结构的动态重配置.这些工作表明,图文法,尤其是上下文相关图文法有足够的描述能力来刻画软件体系结构和体系结构风格,但对于普通软件开发人员来说,使用图文法来精确定义软件体系结构并进行严格分析并不容易.因此,我们需要将图文法规约的工作交给专家,而普通软件架构师可以通过类似于本文所构造的工具直观地构造和调整系统架构.同时,这些工作主要侧重于体系结构的形式模型和属性约束验证方面,对体系结构及其动态重配置的物理实现支持相对较弱.而我们则试图通过对图文法模型的双重解释来弥补这一不足.

在可视化软件体系结构开发环境方面,许多体系结构描述语言均配备有相应的支撑工具,如 ACME Studio[4],ArchStudio[5]以及 Archware[6]等.它们均支持图形化的表示,符合软件开发人员的习惯,也有较为严格的形式语法和语义.但其体系结构描述的形式语法和语义与图形化的表示之间不能直接对应,有损其直观性.同时,它们一般虽能支持动态体系结构的规约,但对动态重配置实现的支持也不够理想,或者局限于特定的体系结构风格.本文的工作实际上是在这种类型的系统的基础上引入了图文法和图转换技术来弥补其不足.

6 结束语

软件工程发展的历史表明,图形化的工具支持是一项软件技术获得广泛应用的重要促进因素.本文讨论了一种基于图文法来构建动态软件体系结构可视化支撑环境的技术途径,给出了其原理设计和一个原型实现.进一步的工作主要包括实践和理论两个方面.在实践方面,要进一步完善相关支撑系统,更重要的是要在其上开发一些更加真实和典型的具体应用以进一步检验这种技术的可行性和有效性;在理论方面,虽然图文法和图转换可以较好地处理软件体系结构的结构建模和结构重配置,但软件体系结构除了结构的侧面以外还有行为的侧面.如何自然地结合图文法模型和某种更善于刻画行为的其他模型(例如进程代数),为动态软件体系结构的建

模、分析、实现和演化提供全面的支撑,是一个需要深入研究的问题.

致谢 感谢罗斌、石兵、冉平、潘健、邢阳和谢德平等人在 ARTEMIS 系统开发中所做出的贡献.

References:

[1] Shaw M, Garlan D. Software Architecture: Perspective on an Emerging Discipline. Upper Saddle River: Prentice Hall, 1996.

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

[3] Allen R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures. In: Astesiano E, ed. Fundamental Approaches to Software Engineering (FASE'98). Berlin: Springer-Verlag, 1998. 21−37.

[4] Garlan D, Monroe RT, Wile D. Acme: Architectural description of component-based systems In: Sitaraman M, ed. Foundations of Component-Based Systems. Cambridge: Cambridge University Press, 2000. 47−68.

[5] Oreizy P, Gorlick MM, Taylor RN, Heimbigner D, Johnson G, Medvidovic N, Quilici A, Rosenblum DS, Wolf AL. An architecture-based approach to self-adaptive software. IEEE Intelligent Systems, 1999,14(3):54−62.

[6] Oquendo F, Warboys B, Morrison R, Dindeleux R, Gallo F, Garavel H, Occhipinti C. Archware: Architecting evolvable software. In: Oquendo F, ed. European Workshop on Software Architecture. Berlin: Springer-Verlag, 2004. 257−271.

[7] Rozenberg G. Handbook of Graph Grammars and Computing by Graph Transformation, Vol.1. River Edge: World Scientific, 1997.

[8] Metayer DL. Describing software architecture styles using graph grammars. IEEE Trans. on Software Engineering, 1998,24(7): 521−533.

[9] Taentzer G. AGG: A graph transformation environment for modeling and validation of software. In: Pfaltz JL, Nagl M, Bohlen B, eds. Applications of Graph Transformations with Industrial Relevance. LNCS 3062, Berlin: Springer-Verlag, 2004. 446−453.

[10] de Lara J, Bardohl R, Ehrig H, Ehrig K, Prange U, Taentzer G. Attributed graph transformation with node type inheritance. Theoretical Computer Science, 2007,376(2):139−163.

[11] Ehrig H, Prange U, Taentzer G. Fundamental theory for typed attributed graph transformation. In: Ehrig H, *et al*., eds. Proc. of the ICGT 2004. LNCS 3256, Berlin: Springer-Verlag, 2004. 161−177.

[12] Kramer J, Magee J. Analysing dynamic change in distributed software architectures. IEE Proc.-Software, 1998,145(5):146−154.

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

[14] Ma XX, Zhou Y, Pan J, Yu P, Lü J. Constructing self-adaptive systems with polymorphic software architecture In: Proc. of the 19th Int'l Conf. on Software Engineering and Knowledge Engineering. Skokie: Knowledge Systems Institute Graduate School, 2007. 2−8.

[15] Lü J, Tao XP, Ma XX, Hu H, Xu F, Cao C. On agent-based Internetware model. Science in China (Series E), 2005,35(12): 1233−1253 (in Chinese with English abstract).

[16] Zhang K, Zhang DQ, Cao J. Design, construction, and application of a generic visual language generation environment. IEEE Trans. on Software Engineering, 2001,27(4):289−307.

[17] Maes P. Concepts and experiments in computational reflection. ACM SIGPLAN Notices, 1987,22(12):147−155.

[18] Garlan D, Cheng SW, Huang AC. Rainbow: Architecture-Based self-adaptation with reusable infrastructure. IEEE Computer, 2004, 37(10):46−54.

[19] Zhou Y, Pan J, Ma X, Luo B, Tao X, Lü J. Applying ontology in architecture-based self-management applications. In: Cho P, Wainwright RL, Haddad HM, Shin SY, Koo YW, eds. Proc. of the 2007 ACM Symp. on Applied Computing. New York: ACM Press, 2007. 97−103.

[20] van Lamsweerde A. From system goals to software architecture. In: Bernardo M, Inverardi P, eds. Formal Methods for Software Architectures. Berlin: Springer-Verlag, 2003. 25−43.

[21] Teitelbaum T, Reps T. The cornell program synthesizer: A syntax-directed programming environment. Communications of the ACM, 1981,24(9):563−573.

[22] Tsiolakis A. Consistency analysis of UML class and sequence diagrams based on attributed typed graphs and their transformation. Berlin: Department of Computer Science, Technische Universitt Berlin, 2000.

[23] Schürr A. Introduction to PROGRESS, an attribute graph grammar based specification language. In: Nagl M, ed. Proc. of the 15th Int'l Workshop on Graph-Theoretic Concepts in Computer. Berlin: Springer-Verlag, 1990. 151−165.

[24] Hirsch D, Inverardi P, Montanari U. Graph grammars and constraint solving for software architecture styles. In: Magee JN, Perry DE, eds. Proc. of the 3rd Int'l Workshop on Software Architecture. New York: ACM Press, 1998. 69−72.

[25] Kong J, Zhang K, Dong J, Song G. A generative style-driven framework for software architecture design. In: Proc. of the 29th Annual IEEE/NASA Software Engineering Workshop. Washington: IEEE Computer Society Press, 2005. 173−182.

[26] Wermelinger M, Fiadeiro JL. A graph transformation approach to software architecture reconfiguration. Science of Computer Programming, 2002,44(2):133−155.

附中文参考文献:

[13] 马晓星,余萍,陶先平,吕建.一种面向服务的动态协同架构机器支撑平台.计算机学报,2005,28(4):467−477.

[15] 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.

马晓星(1975－),男,江苏南通人,博士,副教授,CCF 会员,主要研究领域为软件方法学,软件体系结构,中间件.

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

曹春(1978－),男,博士,CCF 会员,主要研究领域为软件中间件,软件 Agent.

周宇(1981－),男,博士生,主要研究领域为软件体系结构,普适计算.