基于 Delta-Grammar 的软件体系结构演化的描述

程晓瑜 曾国荪 徐洪珍

(同济大学计算机科学与技术系 上海 201804) (嵌入式系统与服务计算教育部重点实验室 上海 201804)

摘 要 软件需要不断演化以适应复杂多变的环境和需求。为了分析软件演化的过程和规则,提出了一种特殊的图文法 delta-grammar 用以描述软件体系结构的演化,具体给出了增加、删除、替换、重组、拆分、并发等演化的产生式规则,从而能够方便、直观、图示化地刻画演化过程。并且,以电子商务信息系统为例,展示了应用 delta-grammar 进行软件体系结构演化描述的过程和作用。

关键词 软件演化,Delta-grammar,产生式规则

中图法分类号 TP311 **文献标识码** A

Description of Software Architecture Evolution Based on Delta-Grammar

CHENG Xiao-yu ZENG Guo-sun XU Hong-zhen

(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

(Embedded System and Service Computing Key Lab of Ministry of Education, Shanghai 201804, China)

Abstract Software requires continuing evolution to adapt complex environment and meet variable requirements. In order to analyze the process and rules of software evolution, we proposed a special graph grammar, delta-grammar to describe the evolution of software architecture(SA). In particular, we provided the production rules of insertion, removal, replacement, recombination, split and concurrency for depicting the evolution process more conveniently, intuitively and graphically. Finally, we showed the process and effect of applying delta-grammar to describe software architecture evolution by taking e-commerce information system for example.

Keywords Software evolution, Delta-grammar, Production rule

1 引言

变化是自然界的客观规律,软件作为客观事物也在进行持续的变化活动,例如环境的变化、管理制度的改变、硬件设备的更新、新功能的增加、新技术的出现等,都迫切需要软件进行不断地变化。尤其在 Internet 成为主流软件运行环境之后,动态、开放、分布的网络特性使得客户需求和计算环境不断改变。为了适应这种需求和环境,在软件系统的生命周期内不断进行软件维护和软件更新,使软件演化控制成为必然[1]。

软件演化伴随软件生命周期中的每一个阶段,由一系列复杂的变化活动组成,这种复杂性导致目前对软件演化的研究只能从宏观层面入手[2]。软件体系结构(Software Architecture)作为软件的高层描述,摒弃了繁琐的细节,为人们宏观把握软件的整体结构和软件演化提供了一条有效的途径[3]。面向体系结构层次的软件演化是近年来该领域的研究热点,其主要工作集中在软件演化描述和支持演化的执行工具上,但是仍缺乏合理有效的描述机制,很难进行全面深刻的演化分析。

通常,描述软件体系结构演化一般采用体系结构描述语言 ADL,或统一建模语言 UML。ADL(Architecture Description Languages)作为一种语言,有严格的语法和语义,支持精确刻画体系结构的演化,并对演化进行分析。ADL 有多个版本,如 Darwin, Rapide, Wright 等。其中 Darwin[4] 运用 π 演算给出体系结构的语义,支持演化的建模;Rapide[5] 基于偏序事件集对构件的计算行为和交互行为进行建模;Wright[6] 以 CSP 形式化语义为基础,提供体系结构的演化建模。以上 ADL 的描述严密,有较强的表达能力,但缺乏直观的可视化效果,不易理解和沟通。另外,UML(Unified Modeling Language)作为一种图形化的工具,有利于描述静态的软件体系结构,但缺乏形式化的验证机制[7],对于体系结构的演化建模也没有统一的标准,不能很好地支持演化。图文法(Graph Grammar)作为一种较成熟的描述方法,直接以图为主要处理对象,同时又是一种基于规则的系统,可以方便直观地刻画体系结构及其重配置操作[8]。Daniel Le Metayer[9], Jun Kong, Kang Zhang[10] 等使用图文法描述体系结构及其风格,并用图

到稿日期:2009-10-30 返修日期:2010-01-29 本文受 863 项目(2007AA01Z425, 2009AA012201), 973 课题(2007CB316502), 国家自然基金项目(90718015), NSFC-微软亚洲研究院联合资助项目(60970155), 教育部博士点基金项目(20090072110035), 上海市优秀学科带头人计划项目(10XD1404400), 高效能服务器和存储技术国家重点实验室开放基金项目(2009HSSA06)资助。

程晓瑜(1986—),女,硕士生,主要研究方向为软件演化、可信软件,E-mail:fish.c0616@gmail.com;曾国荪(1964—),男,博士,教授,博士生导师,主要研究方向为并行分布处理、可信计算;徐洪珍(1976—),男,博士生,副教授,主要研究方向为软件演化、可信软件。

转换的方法实现其演化过程。它的优点在于直观、易于理解，可视化效果好，并且文法规则在传达系统的结构信息上有很强的能力[8]。本文采用一种特殊的图文法，即 delta-grammar，定义了一系列演化操作，用以描述体系结构演化及演化约束，并以电子商务系统的体系结构为例进行演化过程描述以及演化正确性和完整性的分析。应用举例表明，delta-grammar 能很好地支持演化并保证演化系统的一致性。

2 软件体系结构演化的相关概念

2.1 软件演化的基本概念

达尔文的生物进化论以优胜劣汰的原则刻画生物界的繁衍生息。生物体内部基因的交叉、突变、重组以及外界的环境因素，都致使生物不断地演化：从简单到复杂，从低级到高级。软件作为自然界的客观事物同样需要随着环境和需求的变化而变化，例如继承已有软件的优良性能、改善退化的结构、重组得到新的软件等。演化性是软件的基本属性。下面给出相关的基本概念。

定义 1(软件演化)　从广域义上说，是软件随功能、需求、环境、时间等变化而变化的过程。

定义 2(软件体系结构，SA)　是对软件系统整体组织结构和控制结构的刻画，包括对系统中各计算单元(构件)的功能分配、对各单元之间的高层交互(连接件)说明以及对软件体系结构的约束。

定义 3(软件体系结构演化)　是指由于系统需求、环境、分布、技术等因素的变化而最终导致软件体系结构的变动。

定义 4(软件体系结构演化操作)　是指组成系统的构件随软件功能变更、环境因素变化而进行的增加、删除、替换等一系列操作，其本质归结为修改构件和构件之间的关系。例如：

增加构件：当系统功能需要扩充时，要往系统中增加新的构件。

删除构件：当对系统功能进行裁剪，或当系统中的某个构件出现问题时，需要删除系统中的某个构件。

替换构件：当实现了高性能算法而对现有构件的功能进行增强或修正原有错误时，需要构件新版本代替原有旧版本。

2.2 软件体系结构演化的要求

软件演化不能随心所欲，需要满足一定的约束和要求。以下是相关要求的定义。

定义 5(构件的一致性)　是指演化更新后的构件在与其他构件交互时，必须保持原有接口的一些约束，使得继承的属性得以满足。

定义 6(构件的兼容性)　是指原有的构件集合 A 满足安全属性 P，则演化后的构件集合 A' 仍然满足 P。形式化表示为 if P holds in A, where $A=\{C_1,C_2,C_i,\cdots,C_n\}$ then P holds in A', where $A'=\{C_1,C_2,\cdots,C_i,\cdots,C_n\}$。

定义 7(正确性)　是指假定一属性 P 具有不变性，即在演化操作中保持不变。如果每一个由初始配置 G_0 演化得到的配置 G 都满足属性 P，则认为演化是正确的。形式化表示为 $\forall G\in Pro(G_0)$: P holds in G，其中 G_0 为初始配置，$Pro(G_0)$ 表示由 G_0 演化得到的所有配置。

定义 8(完整性)　假定一属性 P 具有不变性，即在演化操作中保持不变。如果对于任意配置 G 都满足属性 P，那么 G 可由初始配置 G_0 演化得到，则认为演化是完整的。形式化表示为 if P holds in G then $G\in Pro(G_0)$，其中 G_0 为初始配置，$Pro(G_0)$ 表示由 G_0 演化得到的所有配置。

只有满足系统约束的演化才是合法的，因此保证系统正确性是软件演化需解决的关键问题之一。从验证的观点看，体系结构允许改变，但系统的某些特性必须保持不变。

3 基于 delta-grammar 的软件体系结构演化的描述

软件系统规模的增大以及复杂性的提高，增加了体系结构描述系统的难度。ADL 难以理解，不便于交流，因此目前的体系结构描述更依赖基于图论的方法。本文采用的就是 delta-grammar 这种图形化的描述方法。

3.1 delta-grammar(Δ-grammar)概念

根据文献[11]，我们引用 delta-grammar 概念，定义如下。

定义 9(delta-grammar)　由一个初始图和一组 Δ-产生式组成，表示为 delta-grammar $H=(G_0,P)$，其中 G_0 表示初始图，P 表示一组 Δ-产生式。一个 Δ-产生式包括 5 个部分，由图 1 的三角形表示(故名 Δ-文法)，分别为：

the guard：位于三角形的顶端，是一个布尔表达式，当条件为真时，图重写才能发生。

the retraction：位于三角形的左边，表示在图重写过程中需删除的部分。

the insertion：位于三角形的右边，表示在图重写过程中新创建和嵌入的部分。

the context：位于三角形中央，表示在图重写过程中需保持一致不能改变的部分。

the restriction：位于三角形的底部，表示在图重写过程中不能存在并发生的部分。如果子图与 context 和 retraction 匹配，并且满足 restriction，那么子图就不能应用这个产生式。

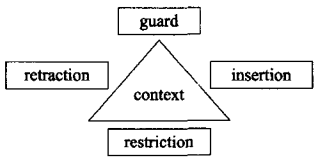

图 1　delta-grammar 产生式

需要说明的是，在 guard 中，本文又赋予它检测体系结构的一些非功能属性的特征，包括体系结构风格的约束等，也是一个布尔表达式，当条件为真时图重写才能发生。对于 restriction，在一般图文法的基础上增加了更严格的结构约束，同时用 delta-grammar 能简化一般图文法的产生式制定规则，从而避免繁琐的产生式制定过程。

对于一个图，G 应用 Δ-产生式 p 可通过以下步骤完成：

(1)查找图 G 中是否存在与产生式 p 中 context 同构的子图。

(2)如果不存在，或者如果同构的子图与 p 中的 retraction 匹配且满足 restriction，则 p 不能应用在 G 上。

(3)如果 guard 存在，则计算 guard 的布尔值。如果为 false，则 p 不能应用在 G 上。

(4)否则删除 G 中与 retraction 同构的子图中的元素，留下主图。

(5)增加 insertion 中的元素并将其嵌入主图，表示为产生式 p 中连接 context 和 insertion 的边。

定义 10(图转换,graph transformation 或图重写,graph rewriting) 记 $G \xrightarrow{p} H$,是指在一个图 G 上,应用产生式规则 p 得到新图 H。应用上述 5 个步骤,图转换过程可形式化表示为:

When($guard$==ture)

$$G \xrightarrow{p} H = \begin{cases} G + p.insertion - p.retraction, & p.restriction \text{ is not satisfied} \\ G, & p.restriction \text{ is satisfied} \end{cases}$$

因此,在初始图上应用一系列产生式规则,可以逐步推导构建体系结构,并实现体系结构的演化。此过程可表示为 $G_0 \xrightarrow{p_1} G_1 \xrightarrow{p_2} \cdots \xrightarrow{p_n} G_n$,$p_i \in P$,$G_0$ 为初始图。

3.2 软件体系结构演化操作

本节主要阐述如何应用 delta-grammar 描述图转换的基本操作,即单步演化的过程。下面用矩形表示构件,直线表示构件之间的连接关系。

3.2.1 增加构件产生式

当系统功能需要扩充时,要往系统中增加新的构件。假设初始图为单个构件 A,则定义增加构件 C_1 的 Δ-grammar 产生式如图 2 左部所示。如果初始图中存在与产生式中的 context 同构的子图,并且 guard 为真,即可在同构的子图中应用产生式。图 2 右部描述的就是初始图应用增加构件产生式的演化过程,在构件 A 上连接了构件 C_1。

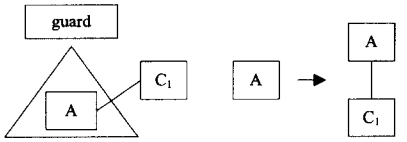

图 2 增加构件产生式及构件 C_1 增加

3.2.2 删除构件产生式

当对系统功能进行裁剪,或系统中的某个构件出现问题时,需要删除系统中的某个构件。定义删除构件 C_2 的 Δ-grammar 产生式如图 3 左部所示。同理,根据应用产生式的步骤,并考虑产生式的 restriction 部分。如果图 G 中存在与 retraction 同构的子图,同时满足 restriction,则不能应用产生式,否则可应用。图 3 右部描述的就是应用删除构件产生式的演化过程。

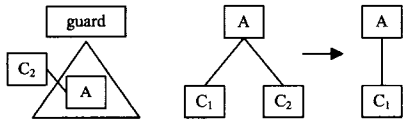

图 3 删除构件产生式及构件 C_2 删除

3.2.3 替换构件产生式

当对现有构件的功能进行增强,或修正原有错误时,需要构件新版本代替原有旧版本。定义用构件 C' 替换构件 C_1 的 Δ-grammar 产生式,如图 4 左部所示。图 4 右部描述的就是应用替换构件产生式的演化过程。由于产生式中制定了限制条件,即与构件 C_2 相连的构件 C_1 不能被替换,因此演化后的图中只有仅与 A 相连的 C_1 替换成了 C'。

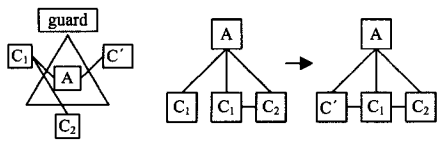

图 4 替换构件产生式及构件 C_1 替换为 C'

3.2.4 重组构件产生式

当对系统功能进行调整,或对系统功能进行扩充时,需要重新组合构件。定义 C_1,C_2,C_3 重组操作的 Δ-grammar 产生,如图 5 左部所示。产生式中增加了 C_1,C_3 间的连接,删除了 C_1,C_2 间的连接。图 5 右部描述的就是应用重组构件产生式的演化过程。

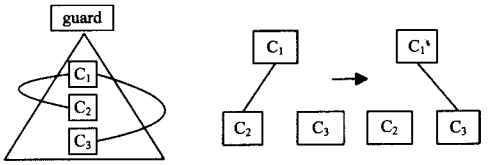

图 5 重组构件产生式及构件 C_1,C_2,C_3 的重新组合

3.2.5 拆分构件产生式

当把系统功能分解成独立的子功能模块时,需要拆解原先组合在一起的构件集合。定义 C_1,C_2,C_3,C_4 构件拆分操作的 Δ-grammar 产生式,如图 6 左部。删除 C_1 与 C_3 构件间的连接。图 6 右部描述的就是应用拆分构件产生式的演化过程。

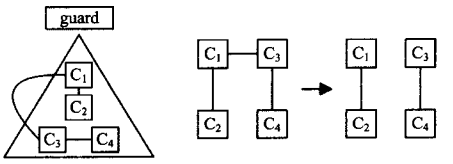

图 6 拆分构件产生式及构件 C_1,C_3 的拆分操作

3.2.6 并发关系产生式

为了提高系统的运行效率,需要对串行连接的一些构件实现并行化。定义 C_1,C_2,C_3 并发关系的 Δ-grammar 产生式,如图 7 左部。图 7 右部描述的就是应用并发关系产生式的演化过程。

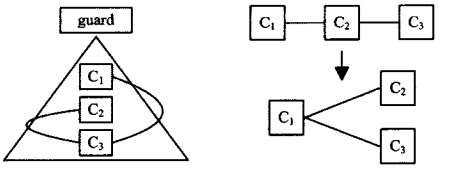

图 7 并发关系产生式及构件 C_1,C_2,C_3 由串行变为并行

4 delta-grammar 演化描述的应用举例

4.1 C/S 体系结构实例说明

近年来,电子商务应用技术发展迅速,C/S 体系结构成为电子商务信息系统的主要框架,而基于 P2P 的网络结构又进一步推进了电子商务系统的演化。图 8 就是一个在 P2P 网络环境下,基于 C/S 的电子商务信息系统体系结构。此系统包括两类构件:(1)Server 服务器,包括 Web 服务器 WebS、文件服务器 docS、数据服务器 datS 等;(2)Client 客户。其中 S 表示服务器 Server,C 表示客户 Client。他们之间的交互用直线表示。Client 是服务接受者,它向 Server 发出访问资源的请求;Server 则是服务提供者,它接受来自 Client 的请求并做出应答,将结果返回给 Client。随着用户需求和环境的改变,C/S 体系结构需要不断演化以满足需求,并在演化过程中满足体系结构风格的约束及特定的属性。

我们在这里给出以下两个演化中需考虑的属性要求:(1)一个客户只能与一个服务器连接;(2)每个服务器最多只能服务 3 个客户。由于篇幅所限,在此只讨论增加服务器、增加客

户以及替换服务器等3种演化过程。

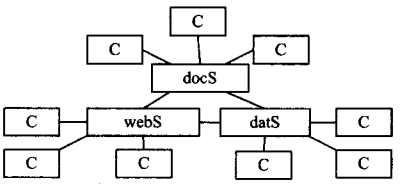

图8 P2P网络下的基于C/S的电子商务体系结构

4.2 基于delta-grammar的C/S体系结构演化

4.2.1 增加服务器

随着系统的不断运行,有些客户需要隐藏自己的IP进行访问操作。而Web服务器由于负载过重,也需要其他服务器为其减轻负载,这时代理服务器(proxy server,proS)就可以同时满足客户和Web服务器的需求。客户向代理服务器请求数据,如果请求的数据在代理服务器中有缓冲,则直接将缓冲的数据发给客户,从而降低了Web服务器的负载。因此我们制定如下产生式规则P1:表示在Web服务器和数据服务器之间增加代理服务器,如图9(a)所示,其演化过程如图9(b)所示。

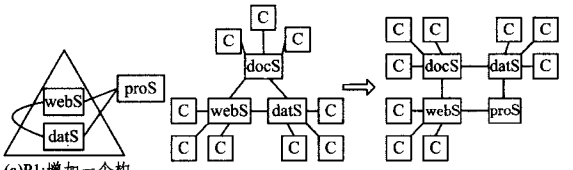

图9

4.2.2 增加客户

当系统新增了代理服务器,系统的处理能力将不断增强,请求代理服务的客户就会相继连接到代理服务器上。同时原本通过Web服务器请求资源的部分客户也开始使用代理服务器,从而代理服务器上连接了新的客户以及来自Web服务器的客户。因此我们制定如下产生式规则P2:表示在代理服务器上连接新客户;制定产生式P2′:表示客户、Web服务器、代理服务器的重新组合,如图10(a)所示。同时考虑到体系结构风格的约束,演化过程必须满足属性:(1)一个客户只能与一个服务器连接;(2)每个服务器最多只能服务3个客户。因此为了满足属性(1),在P2的restriction中必须保证新添加的客户C没有连接在其他代理服务器proS′上,否则添加失败。另外,为了满足属性(2),在P2的guard中,需确保客户的数量必须小于3才能应用此产生式。由此分别应用P2两次,应用P2′一次,即在proS上增加两个新客户C和一个来自Web服务器的客户C的演化过程如图10(b)所示。

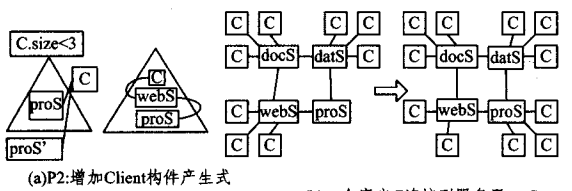

图10

4.2.3 替换服务器

系统运行过程中,可能因为硬件故障或人为的入侵,导致Web服务器一些重要数据或重要组件被破坏,造成服务器无法正常工作。为了保证客户的访问不被突然终止,这时需要激活备用的Web服务器,以便继续提供服务,从而保证整个系统正常运行。因此我们制定如下产生式规则P3:表示用WebS′替换WebS,如图11(a)所示。其演化过程如图11(b)所示。

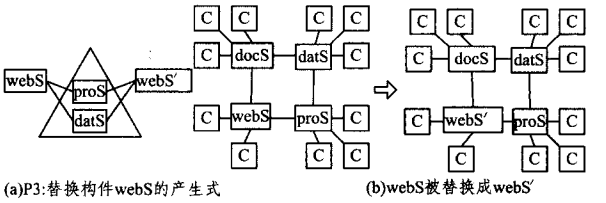

图11

以上描述的就是增加一个代理服务器,并在代理服务器上连接3个新的客户,以及当Web服务器毁坏激活备用服务器WebS′的演化过程。

4.3 举例演化的属性要求分析

为了使演化后的系统能够正常运行,需要确保演化是正确的。通常,演化后的体系结构需要满足特定的属性及其风格的约束。下面针对第2.2节中定义的软件体系结构演化的正确性和完整性对上述举例进行简略的分析。

这里选取属性P:"每个客户只能与一个服务器相连",作为演化过程中必须保持的约束要求。因为每个预设的产生式规则都考虑了这个限制条件,所以每个由初始体系结构应用产生式规则演化得到的新的体系结构都满足属性P,从而说明演化过程是正确的。另外,对于每一个新的体系结构都满足属性P,我们对其进行自底向上的图的解析操作,即从新的体系结构经过delta文法的规约直至到达初始图的过程。经过图解析最后证实都能够规约到初始体系结构,从而再次说明了演化过程是完整的。

另外,现有的技术,如ALLOY分析器、AGG图形解析工具以及模型检测等也都提供了强有力的方法对图转换进行分析和验证[12]。

结束语 为适应动态开放的网络环境和持续改变的用户需求,软件需要不断演化。本文从体系结构的层面,提出一种基于delta-grammar的方法来描述软件的演化过程。在原有的增加、删除、替换的基本操作上,新定义了重组、拆分、并发的产生式规则和操作序列,并以电子商务的C/S体系结构为例,应用delta-grammar方法描述了其演化的过程,并对演化的正确性和完整性进行分析,确保了软件系统演化的一致性。本描述方法以图为工具,方便直观、易于理解,而且文法规则可有效地刻画软件体系结构的演化操作。

本文主要致力于描述软件系统的结构层面演化,尚不涉及软件系统的行为演化。如何描述软件体系结构行为的演化,以及如何保证行为演化的一致性,是将来的主要研究工作。

参考文献

[1] 李长云,何频捷,李玉龙. 软件动态演化技术(第一版)[M]. 北京:北京大学出版社,2007

[2] Mens T, Wermelinger M. Challenges in software evolution[C]// Proceedings of the 8th International Workshop on Principles of Software Evolution. 2005:13-22

(下转第150页)

由〈Target〉、〈Rule〉和〈Obligation〉元素构成，在策略决定点做出评估，以生成和访问决策[8]。由于可能发现多项策略对于访问决策来说都是可用的(并且由于单个策略可以包含多个 Rules)，因此可以使用组合算法(Combining Algorithms)来将多个结果协调成单个决策。标准组合算法被定义为 Deny-Overrides，Permit-Overrides，First Applicable 和 Only-One Applicable 等。〈Target〉元素用于将被请求的资源关联到一个可用的〈Policy〉上。它包含的请求〈Subject〉，〈Resource〉或〈Action〉必须符合〈Policy〉或〈Rule〉的条件，以便资源更适用。〈Target〉包括一个内置的方案，用于有效地索引/查找策略。〈Rules〉提供了在一个〈Policy〉中测试相关属性的条件。可以使用任何数量的〈Rule〉元素，每一个〈Rule〉元素都可以产生一个真或者假的结果。将这些结果进行综合便产生了对该〈Policy〉的单一决策，可以是 Permit，Deny，Indeterminate 或者 NotApplicable 决策。

结束语 由于 SOA 环境分布性、异构性、动态性的特点，基于角色的访问控制已经不能适应 SOA 环境下访问控制的需求。基于属性的访问控制以主体、资源、环境的属性为基本访问控制粒度，可以更加灵活地适应开放的 SOA 环境。结合 XACML 和 SAML 实现的 ABAC 框架，由于安全基于 XML 标准，更符合 SOA 异构环境的特点。但是大量的 XML 文档会增加网络资源的开销，从而增加访问的延迟时间。而且随着访问决策粒度的细化，SAML 断言中将会包含大量的属性信息，使这种延迟更加明显。所以如何控制访问决策粒度和找到最优化的访问控制策略，从而达到安全和性能的平衡，是未来需要研究的重点。

参 考 文 献

[1] 邢少敏，周伯生. SOA 研究进展[J]. 计算机科学，2008，35(9)：13-20

[2] Sandhu R S，Coyne E J，Feinstein H L，et al. Role-based access control models[J]. Computer，1996，29(2)，38-47

[3] 尹刚，王怀民，滕猛. 基于角色的访问控制[J]. 计算机科学，2002，29(3)：69-71

[4] Yuan E，Tong J. Attributed based access control (ABAC) for Web services[C]//IEEE International Conference on Web Services(ICWS'05). 2005

[5] OASIS. eXtensible Access Control Markup Language(XACML) v2. 0[S]. OASIS Standard. http://docs. oasis-open. org/xacml/2. 0/，2005-5

[6] OASIS. Security Assertion Markup Language (SAML) v2. 0 [S]. OASIS Standard. http://docs. oasis-open. org/security/saml/v2. 0/，2005-5

[7] Steel C，Nagappan R，Lai R. 安全模式[M]. 陈秋萍，罗邓，袁国忠，等译. 北京：机械工业出版社，2006：204-233

[8] 李晓峰，冯登国，徐震. 基于扩展 XACML 的策略管理[J]. 通信学报，2007(01)：103-110

[9] 罗海，安世全. 网格访问控制及对 RBAC 模型扩展的研究[J]. 重庆邮电大学学报：自然科学版，2008，20(6)：714-718

(上接第 116 页)

结束语 本文描述了分布式交换机内部通信系统的设计和实现，并重点论述了基于 FEC 的可靠组播算法的实现。提出了一种自适应的 FEC 动态算法并计算了 RDCP 协议的性能。通过试验，验证了 RDCP 协议的有效性和可扩展性。

参 考 文 献

[1] Yang L，et al. Forwarding and Control Element Separation (ForCES) Framework[S]. RFC3746. 2004

[2] Floyd S，et al. A Reliable Multicast Framework For Light-Weight Sessions and Application Level Framing[J]. IEEE/ACM Transactions on Networking，1997，5(6)：784-803

[3] Adamson B，et al. Negative-acknowledgment(NACK)-Oriented Reliable Multicast(NORM) Protocol[S]. RFC 3940. 2004

[4] Rizzo L. Effective erasure codes for reliable computer communication protocols[J]. Computer communication Review，1997，27 (2)：24-36

[5] McAuley A J. Reliable Broadband Communication Using a Burst Erasure Correcting Code[C]//Proc. ACM SIGCOMM 90. 1990：287-306

[6] Huitema C. The case for packet level FEC[C]//Proceedings of IFIP 5th International Workshop on Protocols for High Speed Networks. 1996：110-120

(上接第 130 页)

[3] 王映辉，张世琨，刘瑜. 基于可达矩阵的软件体系结构演化波及效应分析[J]. 软件学报，2004，15(8)：1107-1115

[4] Magee J，Kramer J. Dynamic structure in software architectures [C]// Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering. 1996：3-14

[5] Luckham D. Rapide：A language and toolset for simulation of distributed systems by partial orderings of events[C]//Proceedings of the DIMACS Workshop on Partial Order Methods in Verification. 1997：329-357

[6] Allen R. A formal approach to software architectures[D]. Pittsburgh：Carnegie Mellon University，1997

[7] Kacem M H，Miladi M N，Jmaiel M，et al. Towards an UML profile for the description of software architecture[C]// The Conference on Component-Oriented Enterprise Applications. 2005：25-39

[8] 马晓星，曹春，余萍. 基于图文法的动态软件体系结构支撑环境[J]. 软件学报，2008，19(8)：1881-1892

[9] Metayer D L. Describing software architecture styles using graph grammars[J]. IEEE Trans. on Software Engineering，1998，24(7)：521-533

[10] Kong J，Zhang K，Dong J，et al. A generative style-driven framework for software architecture design[C]// Proc. of the 29th Annual IEEE/NASA Software Engineering Workshop. 2005：173-182

[11] Loyall J，Kaplan S，Goering S. Abstraction and composition in D-specifications of concurrent systems[C]// Proc. 6th Int. Workshop Software Specification and Design. 1991：52-59

[12] Bucchiarone A，Galeotti J P. Dynamic software architectures verification using DynAlloy[C]//Proceedings of GT-VMT. 2008