

面向方面的动态软件架构研究

方义秋¹, 王桂丹¹, 葛君伟²

(1 重庆邮电大学 计算机科学与技术学院, 重庆 400065; 2 重庆邮电大学 软件学院, 重庆 400065)

摘 要: 基于面向方面编程(AOP)技术, 提出了一种层次化的 AODSA 模型. 通过将横切关注点和系统需要更新的组件定义为方面组件, 利用 AOP 的动态编织技术将其织入源系统, 实现了系统的动态演化. 该模型的有效性和可操作性可以通过一个可动态演化的网络投票系统得以验证.

关键词: 软件架构; 动态软件架构; AOP; 动态演化

中图分类号: TP311.5

文献标识码: A

文章编号: 1000-7180(2011)03-0023-04

Research of Aspect-oriented Dynamic Software Architecture

FANG Yi-qi¹, WANG Gui-dan¹, GE Jun-wei²

(1 College of Computer Science and Technology,

Chongqing University of Posts and Telecommunications, Chongqing 400065, China;

2 College of Software, Chongqing University of Posts and Telecommunications, Chongqing 400065, China)

Abstract: This article is based aspect-oriented programming(AOP) technology, proposing a hierarchical model of AODSA. Through defining crosscutting concerns and the component need to be updated as aspect component, then weave it into source system by using the dynamic weaving technology of AOP, realizing dynamic evolution of software system. The model's availability and maneuverability through an online dynamical voting system to verify.

Key words: software architecture; dynamic software architecture; AOP; dynamic evolution

1 引言

近年来动态软件架构(Dynamic Software Architecture, DSA)的研究受到广泛关注^[1-3]. 如文献[1]论述了基于 π -net 的动态软件架构的建模方法, 文献[2]讨论了基于反射技术的动态软件架构实现技术, 文献[3]提出了基于 Petri 网和 Pi 演算相结合的动态软件架构实现技术. 基于反射的动态软件架构技术在系统运行时能自适应环境和需求的变化; 面向对象 Petri 网的 OPN 可以形象地描述软件体系结构的初始化模型和动态行为, Pi 演算可以描述软件体系结构的动态演化, 是目前较好的动态软件架构方法.

虽然动态软件架构的研究已取得了很好的成果, 但现有的方法主要针对系统的功能组件进行分

析和设计, 忽略了系统中的非功能组件, 使系统模块化程度低、难以扩展和维护. 随着软件规模的增大和复杂性的增强, 研究高效实用的动态软件架构方法十分必要. 基于此, 本文针对当前的动态软件架构技术, 结合 AOP 技术(Aspect-oriented Programming), 提出一种面向方面的动态软件架构(Aspect-oriented Dynamic Software Architecture, AODSA)模型.

2 AODSA 模型

动态软件架构(DSA)是允许系统在运行时发生更新的软件结构. 当前主流的动态软件体系结构模型都有一定的优越性, 如文献[4]论及的基于构件的动态软件体系结构模型, 具有良好的开放性, 能够通过控制和更新管理机制进行扩展; 文献[5]讨论的

通过 AODSA 模型与系统动态演化过程的结合,系统的演化可以以增加、删除、替换和结构重配等一些列操作来表达。

(1) 方面的增加和删除

系统在运行时通过增加方面(Aspect)组件的方式支持完善型演化。Aspect 被加入前,要先完成初始化,相对于外部连接来说,此方面是孤立和静止的,只要 Aspect 所具有的能力与其在特定的环境内所扮演的角色匹配,就可以增加该 Aspect,所以 Aspect 加入前的条件为永真。系统在运行时通过删除 Aspect 删除某些不再需要的行为,如由于添加了新的 Aspect 而不需要原有的系统行为。被删除的 Aspect 在退出前必须静止该 Aspect 改变角色,且与当前扮演的所有角色相关的 Aspect 的状态为静止,即该 Aspect 与外界是孤立的。删除的控制条件因不同的应用系统而不同,如,系统的运行环境可能禁止某些函数仍然驻留在运行栈时删除 Aspect,而有些系统也可能被特别设计成可以接受计算能力或状态突然丢失。

(2) 方面的替换

Aspect 的替换算法和 Aspect 的增加、删除算法有着密切的关系。可以把 Aspect 替换看作是 Aspect 先删除然后增加的特例,同时要满足两个必要条件:其一,正在执行的 Aspect 的状态必须移交给新的 Aspect;其二,在演化过程中两个 Aspect 都必须是活动的。它的典型实例是修正型演化和适应型演化。图 3 中用 UML 时序图来描述 Aspect 的替换过程。在算法的叙述过程中将被替换的 Aspect 称为初始 Aspect,替换的 Aspect 称为替换 Aspect,由于在替换过程中,它们所扮演的角色必须相同,因此统称为当前角色。

(3) 系统结构的重配置

系统结构的重配是指将已有的计算功能重新组合起来以修改系统的行为。系统在运行时重配置可以通过改变连接件的绑定来实现,因为连接件负责所有组件的交互。同组件的替换一样,如果组件要求的是可靠通信,就必须保证在通信过程中不会丢失消息。基于 AODSA 模型的重配置,应先在配置层定义结点实体的抽象状态,这些结点实体包括 Aspect 本身及 Aspect 内部的组件。结点实体的抽象状态可以分为四种:主动状态(active)、被动状态(passive)、静止状态(frozen)、移动状态(moving)。主动状态是结点的正常执行状态,它能够发起事务也能够响应事务;在被动状态下,结点实体只能响应事务,不能

发起事务;而在静止状态,结点实体既不能发起事务,也不能响应事务。移动状态表明结点实体正处在移动中。随后,处在运行中的系统在 Aspect 的增加、删除、替换等算法基础之上,重新组织系统的拓扑结构满足系统的要求。

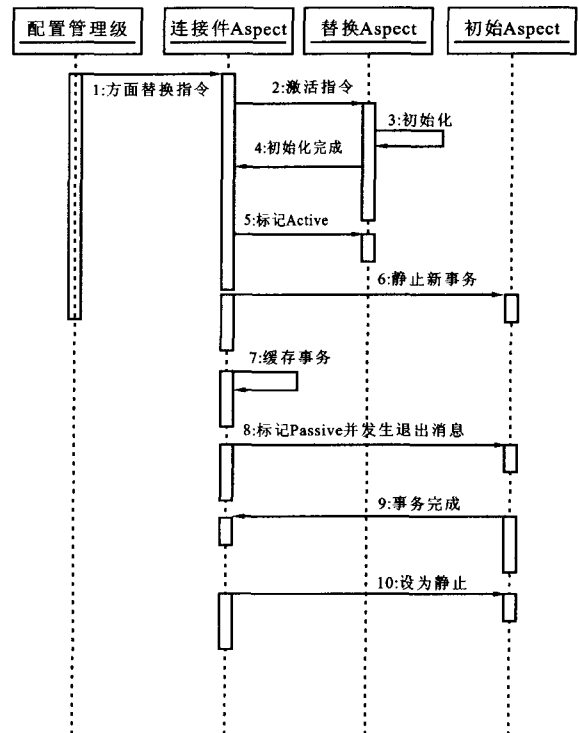


图3 方面时序图

4 案例分析

当前,网络投票已经成为一种新趋势,而在系统设计之初,设计者没有考虑到过多的用户同时登录对系统负载所造成的影响。在系统投入运行一段时间后,设计人员临时决定当投票人员达到一定数量时,当前的服务器 Server1 不能满足系统要求时,此时只有系统自动找到另一台结构相同的服务器 Server2 替代 Server1 来继续工作,以维持系统的正常运行。而且随着网络的发展,用户可以通过各种接口进行上网投票,如: WebClient、WAPClient 和 AppClient。网络投票属于客户服务器模式,服务器和客户机通过 HTTP 协议进行交互。从 WWW 环境分析,选择 AODSA 模型,把 HTTP 协议抽象为方面连接件,将客户机、服务器抽象为方面 Client 组件、方面 Server 组件,它们分别实现客户端和服务器的计算。为了简化起见,本文只考虑其中一个子系统,三个客户机和一个服务器的系统。静态系统拓扑结构如图 4 所示。

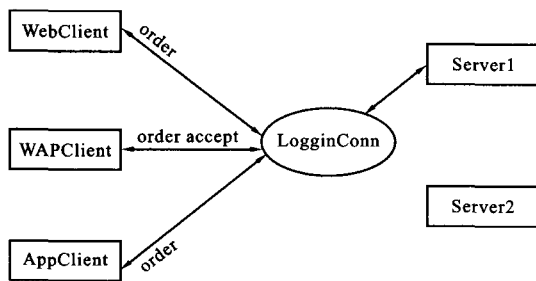


图4 系统更新前的拓扑结构

基于 AODSA 模型的演化过程为：方面配置器接收到更新发起者提出的更新请求后，向软件结构配置器提交更新请求；软件结构配置器首先对更新请求的类型进行分析，然后对更新所涉及的方面连接件和方面发出消息，要求它们做好更新准备工作；准备工作完成后，方面配置器和方面连接件向体系结构配置器返回就绪信息；一切准备就绪后，体系结构配置器通知方面执行更新；具体更新过程为：(1) 先获得 Client 的当前状态，GetComClient(Client)；(2) 若该组件处于活动状态，先将其运行实例挂起 Suspend-ComInstance(Client)；(3) 获得 Client 的接口描述 GetComDescription(Client)，以让系统知道它所请求的功能；(4) 调用执行引擎去搜索系统中满足该请求接口条件的组件，通过计算系统找到了 Aspect Server2 满足条件；(5) 于是通过新增一个连接件将两个组件关联起来，AddConnector(Client, Aspepct Server2)；(6) 激活 Client 的运行实例，ActiveComInstance(Client)。更新完毕后，向方面配置器、体系结构配置器和更新发起者通知更新执行完毕并返回更新结果；同时，体系结构配置器通知方面连接件和系统内其他方面更新结束，此时系统就能和 Server1 损坏之前一样正常运行了。演化后系统的拓扑结构如图 5 所示。

5 结束语

针对应用系统动态调整如何适应需求变化的问题，以动态软件架构的特性结合面向方面编程

(AOP) 技术，提出了面向方面的动态软件架构模型，通过在设计阶段对横切关注点的分离，降低了系统模块间的耦合度、提高了系统的内聚性、改善了代码的缠结和分散问题，从而使整个系统更易于扩展和优化。

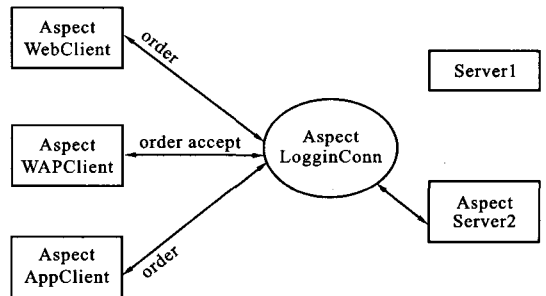


图5 系统更新后的拓扑结构

参考文献：

- [1] Xiao Fu, Xinyu Da, Zhenhua Yu. Modeling dynamic software architecture based on π -net[J]. Information and Communication Technologies, ICCTA, 2006 (2): 2861-2865.
- [2] 唐娜, 赵文耘. 基于反射的动态软件体系结构实现[J]. 微电子学与计算机, 2006, 23(9): 32-34.
- [3] 于振华, 蔡远利, 徐海平. 动态软件体系结构建模方法研究[J]. 西安交通大学学报, 2007(4): 167-171.
- [4] 王海燕. 基于构件的动态软件体系结构模型的研究[D]. 武汉: 华中师范大学, 2004: 18-19.
- [5] 瞿新南. 基于 Agent 的动态软件体系结构研究[D]. 南京: 南京航空航天大学, 2007: 25-27.
- [6] 李长云, 何频捷, 李玉龙. 软件动态演化技术[M]. 北京: 北京大学出版社, 2007: 16-17.

作者简介：

- 方义秋 女, (1963-), 副教授. 研究方向为软件工程.
 王桂丹 女, (1984-), 硕士研究生. 研究方向为软件工程.
 葛君伟 男, (1961-), 博士, 教授. 研究方向为软件体系结构、数据库、地理信息系统等.