CAR 中基于上下文 AOP 机制的实现

郭强[1]　范典[1]　陈榕[2]　周立柱[3]

[1](清华大学 深圳研究生院 软件工程中心 深圳 518055)

[2](清华大学 信息技术研究院操作系统与中间件中心 北京 100084)

[3](清华大学计算机系 北京 100084)

[摘要]本文通过分析侧面的上下文特性，并结合 CAR 构件系统中提供的对上下文的支持，提出了一种基于上下文来实现构件技术和 AOP 技术结合的方法，不同于传统的通过修改系统架构以及引入新的编程语言实现二者结合的方法。一方面充分利用现有系统的特点，另一方面使用简单，有助于 AOP 的理解和应用。

[关键词] 面向侧面　CAR　构件　上下文　代理

中图法分类号：TP311.5　　文献标示码：A

An Implementation of Context Based AOP in CAR Component System

Qiang Guo[1]　*Dian Fan*[1]　*Rong Chen*[2]　*Lizhu Zhou*[3]

[1]Software Engineering Center, Graduate School at Shenzhen, Tsinghua University, Shenzhen 518055, P.R.China

[2]Research Institue of Information Technology, Tsinghua University, Beijing 100084, P.R.China,

[3]Department of Computer Science and Technology, Beijing 100084, P.R.China

[Abstract] By analyzing the context feature of aspect and making use of Context feature of CAR component system, we propose an implementation method that combines CBSE with AOP. The method is different from traditional ones featured by modifying the architecture of system or introducing a new language. It utilizes all the features of existing systems and makes AOP understandable and easy to use.

[key word] AOP component CAR context proxy

1. 介绍

在目前的面向对象的编程模式中，仅仅用类的思想来分析实现软件系统，因此不能有效的表示软件系统的关注点。例如对于系统中历史记录功能代码，安全认证代码等，这些代码一般不代表系统的主要逻辑，但会分散在许多模块当中，实现中也会和其他代码缠结在一起，导致这些代码无法很好的模块化。这些代码在 AOP[1]中称为横切关注点，因为这些代码“横切”了系统的分解性，降低了程序的模块化程度以及重用性。而 AOP 技术通过引入多维的视角对系统进行分析，在类的基础上引入侧面的概念，实现关注点的分离，改善系统逻辑，使得系统可以更好地模块化。

构件系统强调构件之间的低耦合性，和某个单独构件的高聚合性。进一步构件是一个黑盒子实体[3]，通过接口对外提供服务并且是可以单独部署的。事实上，应用构件系统的时候，为了提高重用性系统中的一个构件不应当显示的依赖于其他构件。为了尽可能降低藕合性， 很多非主要功能的代码分布在多个构件当中。因此构件系统会遇到关注点交叉以及代码混乱等问题。而引入 AOP 技术将很好的解决这些问题[3][4]。

AOCE(Aspect Oriented Component Engineering)[9]关注如何把构件技术和 AOP 技术结合起来，虽然处于开始阶段，但已存在一些这方面的应用，例如：JBoss/AOP[5]和 JAsCo[3][4]。为了模块化 aspect，这些技术或者引入新的编程语言或者引入新的框架结构，实现都比较复杂。

CAR(Component Assembly Runtime)[11][12]是一个国内的自主知识产权的先进的构件系统，是由科泰公司开发的新一代的构件系统，其主要的目的

是从操作系统层引入构件的概念，所有的服务由构件来提供，实现软件的目标代码级的应用，是新一代的构件系统，为网络编程和 Web services 提供强大的支持。CAR 是一个面向构件的编程模型，它表现为一组编程规范，包括构件、类、对象、接口等定义与访问构件对象的规定。

CAR[11][12]中我们将通过结合侧面(aspect)和上下文(context)的概念，并且利用 CAR 中已有的上下文的实现机制，简单地实现一种面向侧面的构件引擎，同时也提出一种新的解决方案，充分利用侧面中上下文特点以及现有系统的支持，同一般的实现 AOP 和构件的方法相比，简单而且方便实际应用。

2. AOP 的上下文特性

AOP 通过分离关注点,提高系统的模块化程度，是一种新的编程模型。AOP 引入了侧面的概念，侧面是对横切关注点的抽象，使得可以把横切关注点从系统逻辑中抽离出来，从而更好的模块化。同时侧面会通过一定的编织策略在编译或运行的某个阶段同业务逻辑结合在一起，形成完整的系统。

为了实现 AOP，需要对侧面的上下文（context）的描述。上下文是 AOP 平台编织侧面时需要指定的侧面的存在条件，也就是满足什么样的条件才进行编织。因为侧面分布在系统中的各个模块中，在各个模块中如何分布，及如何与主系统结合都需要通过本身的上下文描述来确定。在 FusJ[3][4]中提出一种构件模型，其中把侧面(context)从支持面向侧面的体系结构中抽象出来，作为连接件(connector)[3][4]，连接件描述了侧面代码和构件之间的关系和接口。这样提供侧面功能的代码可以作为构件出现，而通过连接件实现构件之间的面向侧面的编织。这样推迟侧面概念的引入，而在系统实现的后期通过引入描述构件和侧面之间关系的 connecter 来实现 AOP。在 AspectJ[1][7]中，也有一套用来描述上下文的一套强大的工具，利用 crosscut、joinpoint[1][7]描述静态的 Context，也就是什么时候(Where)引入侧面。before, after, around 等用来描述在什么时间(where)引入侧面。通过上下文指出了在什么时候(When)什么地方(where)引入相应的 aspect。

3. CAR 中的上下文（CONTEXT）机制

CAR 中的上下文机制提供可执行代码的运行环境，用户在编写 CAR 程序的时候，通常调用 Instantiate 方法初始化对象。该方法参数 dwClsContext 用于指定对象与客户程序的相对位置，其可选值列表如下根据构件初始化时指定的参数，提供不同的运行环境及相应的服务。

CAR 中通过代理来实现对上下文机制的支持的[11][12]。系统接到客户端发出的初始化构件的请求后，首先找到相应的构件，根据元数据协议，从构件生成时打包好的元数据信息还原出相应的接口函数及其参数等信息，包括虚表(vtbl)[13]的结构。根据这些信息生成代理，这样可以保证代理的接口和服务端构件接口的表现完全一致。这样代理将负责转发客户所有的请求，并负责返回结果给用户；系统同时会根据不同的请求，在进程内、进程外或跨机器实例化服务端构件，而服务端会建立相应的桥头堡(stub)程序来响应代理的请求，直至请求结束。通过在本地建立一个代理构件响应客户的请求，使得用户可以透明地使用任何位置的构件，都如同使用本地的构件一样(参见图 1)。

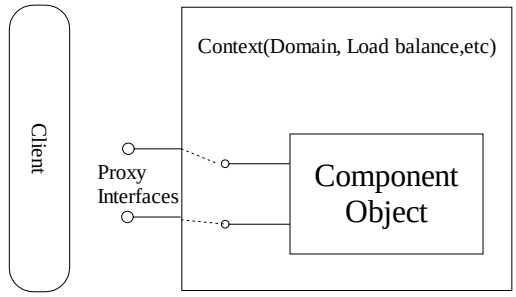

图 1:CAR 中上下文机制

由于系统负责处理代理和构件之间的参数及数据的传递，系统可以对传递的数据进行处理，比如同步，加密，安全，负载平衡等操作，从而使得客户可以透明的使用这些服务，而专注于主要功能代码的实现。这种机制使得许多问题可以很容易的解决，比如网上多媒体的同步问题，以及负载平衡、消息队列等等[11][12]。而在处理代理和构件之间数据传递的时候，增加对侧面编织的处理，可以实现对面向侧面的支持。

4. CAR 中基于上下文面向侧面的实现

通过上面的分析，我们可以发现，CAR 在上下文的实现中，已经引入了 AOP 的思想。考虑其中的对象运行空间（Domain）[11][12]等功能的实现，一方面实现这些功能的代码并不是实现客户功能逻辑的主要代码，另一方面，如果系统没有提供支持的话，这些功能代码将分布在用户的各个模块当中，将会引起代码的分布和缠结，同时由于这些功能的应用的普遍性，完全有必要在系统级提供支持。因此，不同的上下文支持相对于系统中的构件可以理解为系统级提供的侧面支持，不同的运行时上下文对应在运行时编织不同的侧面。

由于侧面本身对上下文的依赖以及 CAR 系统中提供的对上下文的支持，我们利用目前实现上下文的代理机制实现侧面的支持，在代理部分加入编织功能代码，实现动态运行时将侧面与构件编织在一起。在 CAR 中，我们认为上下文是侧面切入构件的连接点。通过在 Context 中引入 aspect 关键字，使得用户可以自己定制侧面及相应的上下文，并且可以通过指定不同的上下文，和相应的侧面结合在一起。一个简单的例子，用户可以定义宠物店的上下文，其中包含提供价格功能的侧面构件。这样猫在宠物店中的时候，自然有了价格行为，反映在 CAR 中构件语言描述就是猫构件在宠物店上下文构件中实例化的时候，系统会把价格侧面行为编织进去，这样猫就拥有了提供价格的接口。

实现 aspect 需要在原来的构件系统上引入的新的概念：

1. 上下文构件(Aspect-Oriented Context component)：该构件提供了 aspect 对应的上下文，主要描述了一种上下文，该上下文构件主要指定构件运行时可以编织的侧面构件。用户通过该构件指定运行时上下文，运行时编织相应的侧面构件。上下文构件的声明类似于普通构件，需要添加 Context(aspect)关键字。

2. 侧面构件(Aspect Component)：提供侧面功能的构件，在实例化的时候由 AOC 构件负责编织到服务端构件中。构件实例化的时候，如果指定了特定的上下文，则系统将把该上下文中的 aspect 与 component 编织在一起。Aspect 构件需要和 AOC 构件一起声明，并且需要声明为可聚合的。

3. Waver：在运行时进行侧面和构件的编织的策路。目前的编织策略是聚合。

具体实现机制如下(参见图 2)：

1. 设计、声明。通过 CDL[12]描述语言，声明相应的 AOC 构件，aspect 构件。通过添加 context 关键字来进行说明该构件是上下文构件，系统在生成上下文构件的时候，会根据同时声明的侧面构件生成查询表，当系统转发客户请求到达上下文构件的时候，上下文构件会采用轮询机制把请求转发到各个侧面构件中来查询相应的接口直至找到相应的接口或者没有相应的接口返回。同时由于上下文构件负责实例化侧面构件，其代码中将包含实例化侧面构件的部分。侧面构件需要和相应的 AOC 构件一起声明。系统会为侧面构件自动生成代理和非代理 IUnknown 接口，用来响应编织请求[13]。而需要编织侧面构件的主构件需要声明为可聚合。

2. 运行时编织。编织的工作在服务构件实例化的时候进行：

2.1. 客户端请求服务的时候，通过调用 Instantiate 初始化服务端构件，参数为所需上下文构件的 CISID(class id)。系统接到请求后，会首先根据参数进行判断，是否指定了系统上下文，初始化构件运行的环境以建立相应的代理和桥头堡程序。然后进行服务端构件的初始化，实例化服务端构件，然后系统检查是否指定了特定的用户上下文，并根据其 CLSID 实例化相应的上下文构件。在实例化上下文构件的时候，会把服务端构件的 this 指针作为参数传输进去，而 AOC 在实例化 aspect 的时候，同样会把服务端构件的 this 指针传输进去，通知 aspect 构件的委托 IUnknown，可以保证接口表现的一致性[13]。AOC 构件实例化之后，其地址将返给服务端构件，服务端构件保存 context 构件的地址，以通过 AOC 来访问 aspect 的接口。AOC 构件实例化的时候，负责将 aspect 构件实例化，并把服务构件的内存地址指针传送进去。aspect 构件实例化的时候，根据元数据协议通过服务端构件指针查询元数据，并取得服务端构件的 CLSID 及其它元信息，这样可以动态根据不同的服务端构件做出不同的响应。

2.2. 将看到是一个扩充接口后的服务构件。扩充后的构件不仅包含服务构件的接口，还有

aspect 构件的接口。当用户查询 aspect 接口的函数时，服务端构件会把请求发送给 context 构件，context 构件将通知其中的 aspect 构件的非委托 IUnkown 接口查询，检查是否有相应的接口，如果有则把相应的接口指针返回到服务构件，然后服务构件返回给客户。这样客户获得 aspect 构件的指针，访问相应的函数。当客户用 aspect 构件的接口指针进行 QI 的时候，由于 aspect 构件是可被聚合的，查询请求将直接通过委托 IUnknown、返回到服务端构件，由服务端构件负责处理查询请求。

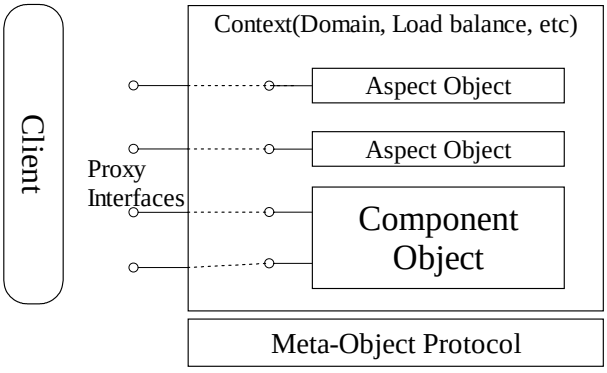

图 2：CAR 中基于 Context 的构件与侧面编制模型

5. 总结

本文提出了一种简单描述侧面构件并进行动态编织的方法。通过对侧面构件中上下文特性的抽象，结合 CAR 系统中的上下文支持，将 AOP 和构件技术相结合，目的在于提供用户的方便使用，用户只需在实例化运行时简单指定构件的上下文，系统即可自动编织相应的侧面构件。

需要强调的是，本文仅仅描述了构件级的横切，把不同的接口函数调用分发给相应的服务构件或是 aspect 构件。在下一篇论文中，将进一步探讨利用 before、after 等关键字在 AOC 构件中加入接口级和接口中函数级的描述，使系统在处理服务构件接口函数调用的时候通过对参数、关键字的解析进行 aspect 构件接口函数的调用。

参考文献

[1] Kiczales, G., Lamping, J., Lopes, C.V., Maeda, C.,Mendhekar, A. and Murphy, A. Aspect-Oriented Programming[C]. Proceedings of the 19th International Conference on Software Engineering (ICSE), Boston, USA. ACM Press. May 1997 Elisa L.A.

[2] Baniassad ,Gail C. Murphy, Christa Schwanninger and Michael Kircher. Managing Crosscutting Concerns During Software Evolution Tasks: An Inquisitive Studyss[C]. AOSD 2002, Enschede, The Netherlands.

[3] Davy Suv´ee, Wim Vanderperren, Dennis Wagelaar and Viviane Jonckers. There are no aspects[C]. Electronic Notes in Theoretical Computer Science. 2004.

[4] Suv´ee, D. Fusej: Achieving a symbiosis between aspects and components[C]. Proceedings of the 5th GPCE Young Researchers Workshop, Erfurt, Germany, 2003.

[5] JBOSS Group, “JBoss/AOP website,” Http://www.jboss.org.

[6] AOSD Website. Http://www.aosd.net.

[7] AspectJ Website. Http://www.aspectj.org

[8] Frdddric Duclos，Jacky Estublier, Philippe Morat. Describing and Using Non Functional Aspects in Component Based Applicationss[C]. AOSD 2002, Enschede, The NetherlandS

[9] John Grundy. Aspect-oriented Requirements Engineering for Component-based Software Systems[C]. Proceedings of RE’99, 7-11 June, Limerick, Ireland.

[10] Dave Thomas. Reflective Software Engineering from MOPS to AOSD[C]. Journal of Object Technology, vol. 1, no. 4, September-October 2002, pages 17-26.

[11] Koretide Website, http://www.koretide.com.cn

[12] Koretide. CAR’s manual[M]. 2004.

[13] Aiming Pan. COM’s principle and COM’s application[M]. The Tsinghua press, 1999.