基于CAR构件的AOP聚合模型的研究

王琦, 陈榕

(同济大学基础软件中心 上海 200092)

摘　要: 分离关注点是目前软件设计领域内的热门话题。通过识别、封装和集成不同种类的关注点来构件系统，从而提高系统的适应性、可维护性和重用性。AOP(Aspect Oriented Programming)技术是目前较好的分离关注点的编程范式，本文针对基于CAR构件的嵌入式系统设计，提出了AOP的动态聚合模型及其实现。不同于其他系统的静态聚合方法，从而有效的提高了系统设计的适应性和重用性。

关键词: CAR 构件; 动态聚合; 方面; 上下文; 聚合

1. 引言

软件工程中，一个系统的整体看成是不同关注点的集合。所谓关注点(concerns)是指识别、封装和处理仅仅围绕某一特定概念和目标的代码。随着面对构件编程(COP component-oriented programming)的兴起，衍生出来的用于封装不同关注点的结构是构件。关注点的种类很多，包括功能性代码和非功能性代码。一些功能性代码以构件的形式很好的封装在一起，提高了软件的复用性，但是也有一些非功能性代码比如日志记录、调试信息、安全检查、错误处理和性能优化等的实现是分散于跨系统的构件中，因而造成代码缠结、混乱、难以更改和复用的结果。

由此，面向方面编程（AOP）技术应运而生。在AOP中这些非功能性的关注点称为横切关注点（crosscutting concerns）。AOP为开发者提供了一种描述横切关注点的机制，将横切关注点用方面构件(Aspect Component)实现。使用AOP技术，开发人员可以很容易的更改，插入或者除去横切关注点，结合面向构件编程技术，极大地减小了开发的复杂度，提高了可维护性。

如何将AOP技术与现有的构件技术很好的结合起来成为问题的关键。目前，AOP技术中采用聚合技术(Aggregation)(也称为编织技术)把方面构件与由功能性代码封装而成的普通构件的结合。已有的AOP技术如AspectJ、AspectC++等都是静态聚合，在编译时进行相应的代码转化并实现聚合，这样的实现比较复杂，降低了软件的灵活性。

CAR（Component Assembly Runtime）是国内自主知识产权的先进的构件系统。CAR构件技术定义了一套网络编程时代的构件编程模型和编程规范，它规定了一组构件间相互调用的标准，使得二进制构件能够自描述，能够在运行时动态链接。本文提出了基于CAR构件中的一种新的聚合方案，称为动态聚合（Dynanmic aggregation）。加上CAR的上下文环境以及CAR构件自动代码工具，CAR构件的聚合简单而灵活，可随时聚合，随时

卸载，发展为面向方面的聚合模型。同一般的AOP和构件聚合的技术相比，极大的降低了系统设计的复杂性，提高了软件的灵活性。

2. AOP 的聚合模型

AOP 技术主要用于分离横切关注点，将横切关注点从系统中逻辑抽象出来并封装为方面构件，然后通过聚合技术将方面构件和普通构件聚合在一起。方面构件在编译或运行时被聚合到系统的普通构件中来产生整个系统。

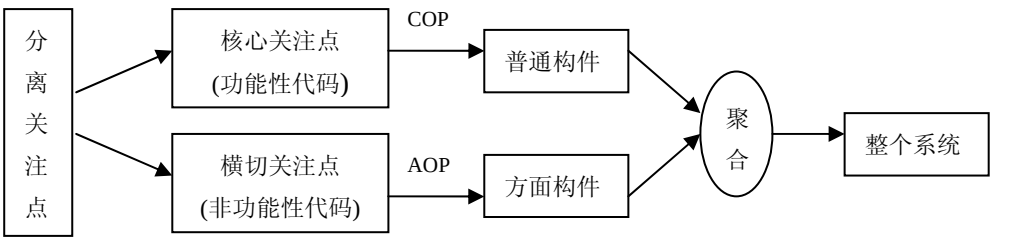

图 1 AOP 聚合模型

把方面构件与普通构件聚合在一起涉及到几个问题。第一，聚合是静态聚合还是动态聚合，这对于应用来说是非常重要的，现在的 AspectJ、AspectC++、D2-Al 都是静态聚合。第二，是支持实现 AOP 的特定语言还是对通用目的语言的扩展，前者目前有 COOL、D2-AL、TyRuba；后者有 AspectJ、AspectC++、Kava。第三，聚合层次是编译时还是运行时，AspectJ、AspectC++、D2-AL 都是编译时运行。

3. 基于CAR的聚合模型

基于 CAR 的 AOP 机制使用户能够在完全不用修改源代码的情况下简单而方便的动态聚合两个或多个 CAR 构件类，从而生成一个具有两个或多个 CAR 构件类所有接口实现的新构件类。CAR 的 AOP 技术是由方面（Aspect），动态聚合（Dynanmic aggregation），上下文环境构件(context)组成。Aspect 对象是实现动态聚合必要条件，动态聚合是上下文环境实现的基础。

3. 1 方面构件(Aspect)

方面构件是一种特殊的构件类实现，Aspect 对象的特征是可以被普通构件对象聚合，在实例化时由上下文环境构件负责聚合到功能构件。

在 CAR 里，只有 Aspect 对象才可以被聚合；一个可被聚合的 Aspect 对象必须实现两个接口，一个用于委托 IObject ，一个用于实现真正的 IAspect 接口，也就是委托和非委托接口，且非委托接口并不从 IObject 接口继承。

Aspect 对 IAspect 的实现是真正意义上的 IObject 实现，而对 IObject 接口的实现只是进行简单的转接。当 aspect 构件对象作为一个独立的构件对象存在时，对 IObject 的方法调用将会完全转接到 IAspect 接口的对应方法上。如果 Aspect 对象被其它构件对象聚合，对 IObject 的方法调用则会被委托给聚合对象,聚合对象保存 Aspect 对象的 IAspect 的接口指针，用于 Aspect 对象的真正 QueryInterface。

3．2 上下文环境构件(Context)

在 CAR 里，上下文环境是方面构件切入普通构件的连接点，，上下文环境也是一种构件，它具有普通构件的所有功能。Aspect 对象作为上下文环境的属性，来表示上下文环境的特征。一个普通构件对象如果进入了一个 Context，那么该对象将具有此 Context 的属性，即 Aspect，一旦对象离开了该 Context，环境属性就会失去（但该对象很有可能又进入了另外一个 Context， 拥有新的环境属性）。比如，一个人进入了学校，学校这个环境就会把学号、班级等属性与之聚合在一起，进入公司，公司就会把员工相应的属性与之聚合在一起。

3．3 动态聚合

动态聚合是普通构件对象在运行时随着 Context 的改变与相应的 Aspect 构件对象聚合，虽然 AspectJ 和 AspectC++里的静态聚合能够满足一定的应用需求，但在现实模型中，更多的情况却是动态聚合。

3.1.1 动态聚合

CAR构件的每个接口都是由IObject继承而来的，动态聚合是通过IObject的Aggregate方法来完成的，在构件实例化时进行。具体实现机制如下：

1．当普通构件进入一个新的 context ,context 首先根据 CLSID 实例化相应的 Aspect 构件，

2．当普通构件对象进入 context 后，context 即获得该对象的接口指针 pOuter, 并调用 Aspect 构件对象的 aggregate 方法，将 pOuter 作为参数传输进去，通知 Aspect 将与该普通构件对象聚合。

3．Aspect 构件对象及时地反调该普通构件对象的 aggregate 方法，并将 IAspect 指针作为参数传入，通知普通构件对象将与 Aspect 聚合。

4．普通构件对象保存 Iaspect 接口指针 。

5．Aspect 构件对象保存 pOuter, 并将其引用计数全部托管到普通构件对象中。

通过以上步骤就实现了聚合，最后看到的是扩充接口后的普通构件，它不仅包含自身原有的接口，还将包含 Aspect 构件的所有接口。当用户查询 Aspect 接口的函数时，普通构件会把请求发送相应的 Context 构件，Context 通知其中的 Aspect 的非委托接口 IAspect 接口进行 QueryInterface(QI)，如果有则把相应的接口指针返回到普通构件，由普通构件传给用户，这样用户就可以访问相应的函数。同时，查询请求将直接通过委托 Iobject 返回到普通构件，由普通构件负责处理引用计数。

假设普通构件构件对象 outer(构件类为 CA),其接口指针为 pOuter, 进入一个上下文环境 cxt。设 ea 中只有一个方面构件对象 aspect (构件类为 AspectB),接口指针为 pAspect, 上下文环境 cxt 将把其中的 aspect 与 outer 聚合。m_pAspect 为 outer 保存的 aspect 的接口指针，m_pOuter 为 aspect 保存的 outer 的接口指针。具体实现如图 2 所示：

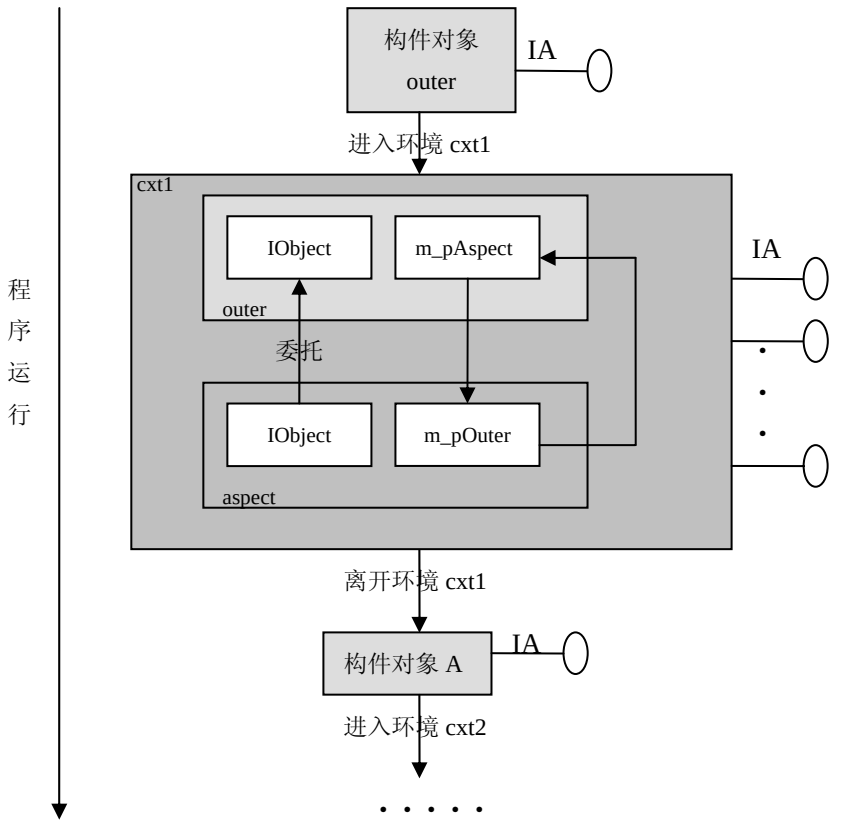

图 2 随时聚合、随时拆卸的面向方面的动态聚合模型

3.1.2 多面聚合

上面介绍的只是两个对象的聚合，普通构件对象只聚合了一个 Aspect 对象，但实际上，在面向方面编程时，往往需要一个对象聚合多个 aspect 对象，这就是多面聚合。在实现多面聚合时，创建多个 Aspect 对象，多次调用 aggregate 方法使一个普通构件对象聚合多个 aspect 对象。多面聚合的结果示意如下：

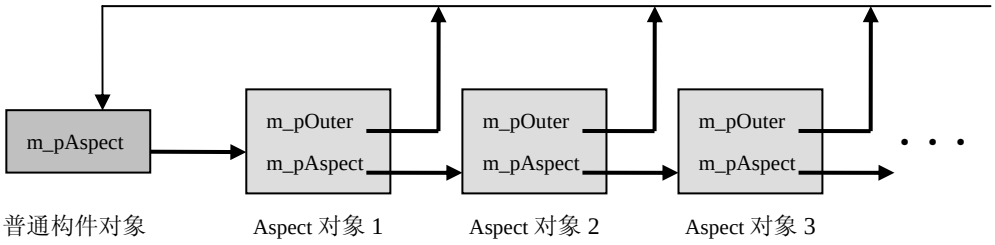

图 3 多面聚合（聚合链）

可以看出，构件对象指向下层被聚合的Aspect对象的指针(m_pAspect)构成了一个单向链表，而链表中的每一个Aspect对象的m_pOuter指针都指向最外层的普通构件对象(聚合者)。通过这种方式，QI调用可以在链表中从头向尾传递，而所有的Aspect对象的引用计数都委托给了最外层的普通构件对象。

3.1.3 动态拆卸聚合

CAR 构件库提供了 Unaggregate 方法来实现动态拆卸聚合。由于聚合同一个 Aspect 类的多个对象实例是没有意义的，因此，可以用 CLSID 来表示被聚合的 Aspect 对象。根据 CLSID 查找出要拆卸的 Aspect 构件。实现过程如下：

(1)普通构件对象通过 Aspect 对象的 QI 出相应的 IAspect 接口指针，如果当前对象找不到与之匹配的 CLSID，则会传递给聚合链里的下一个对象，直到匹配为止，并返回相应的 IAspect 指针。

(2)与 CLSID 匹配 Aspect 对象断开自己并维持剩余的聚合链表

(3) 还原引用计数。由于在聚合时Aspect对象将引用计数完全委托给普通构件对象，所以在拆卸的时候必须还原，还原的过程就是聚合时委托的相反过程。

4. 总结

本文提出了一种基于CAR构件的随时聚合随时拆卸的AOP动态聚合模型，大大提高了系统的灵活性和重用性。实际上对于AOP中的静态聚合还是动态聚合应该根据不同的应用环境选取，将来的AOP技术应该同时支持编译时静态聚合和运行时动态聚合，这样AOP就能根据根据要聚合的方面在两种模型间切换，合并两种聚合模型的优点。对依赖于运行时环境的方面采用动态聚合模型，使它可以在任意必要的时刻进行聚合；对于固定的方面，由于性能原因应该在编译时静态聚合。

参考文献

[1] Kiczales, G Lamping,J,et al. Aspect-Oriented Programming[C]. Porceedings of the 19th International Conference on Software Engineering(ICSE), Boston,USA,ACM Press,1997.

[2] Bollert Kai Heintzestr, On weaving Aspect[C]。ECOOP'99, 1999

[3] Niklas P a.hlsson , An Introduction to Aspect - Oriented Programming and AspectJ , Topic Report for Software Engineering

[4] AspectJ Website[EB/OL].http: www.aspectj.org.

[5] Koretide.CAR's Manual[M]。2004

Aggregation Model of AOP Base on CAR

WANG Qi, CHEN Rong

(Software Center, Tongji University, 200092, China)

Abstract:

Key words: AOP; CAR; Aggregation;Component; Context;Aspect