# 一种基于动态AOP的动态体系结构实现技术*)

张夏 彭鑫 赵文耘
(复旦大学计算机科学与工程系软件工程实验室, 上海 200433)

**摘要**：体系结构的运行时动态演化已经成为许多软件系统的根本需求，而支持动态体系结构的运行平台和实现框架是其中的关键问题。本文在课题组前期开发的基于Java的请求/调用式构件组装工具BSAppBuilder基础上，借助动态AOP技术，提出了一种基于AOP的动态体系结构实现方法。该方法包括了进行运行环境监控的传感器以及执行演化控制的控制中心。在演化实现机制上，该方法利用Aspect的动态织入技术实现构件间连接关系的动态改变。本文还通过一个针对系统运行负载监控的动态演化实例展示了该方法的有效性。

**关键词**：动态体系结构 基于构件的软件开发 AOP 在线演化

# A Dynamic AOP based Implementation for Dynamic Architecture

Zhang Xia, Peng Xin, Zhao Wenyun
(Fudan University Software Engineering Lab, Shanghai 200433)

**Abstract:** Dynamic evolution of software architecture is a necessary requirement for many software systems. A framework and a platform which supports the dynamic software architecture are the crucial aspects. This paper is based on a request/invoke component assembling tool(BSAppBuilder) which is developed by our research group, and proposes an approach for dynamic architecture using dynamic AOP technique. This tool contains a sensor which senses the running environment and a control center which controls the online evolution. In the process of evolution, the approach changes the connections between components using dynamic weaving technique. In the end, this paper provides a dynamic evolution example which senses the load of the system and thus shows the above approach is practical.

**Keywords:** DSA；CBSD；AOP ；Dynamic Evolution

当前，许多研究者开始关注具有自适应能力软件的开发，自适应软件的一个基本特征是软件系统能够在运行时进行动态演化，以适应需求和环境的变化。本课题组前期开发了一个基于Java的请求/调用式构件组装工具BSAppBuilder，该工具用于生成基于Web的构件化系统，本文在BSAppBuilder工具的基础上，借助动态AOP技术，提出了一种基于AOP的动态体系结构实现方法。在实现结构上，该实现工具包括了进行环境监控的传感器、具备演化决策的演化规则以及执行演化控制的控制中心；在实现机制上，利用了AOP动态织入技术以实现构件间连接关系的动态改变。本文给出了一个基于Web的构件化系统实例，在该系统上进行基于负载监控的动态演化，并且展示了该方法的有效性。

本文第一节介绍与动态软件体系结构相关的一些技术，包括CBSD开发技术、AOP技术等，第二节介绍一种动态体系结构实现方法，第三节详述该方法的实现过程。

## 1 背景介绍

### 1.1 动态软件体系结构

体系结构(software architecture,简称SA)从全局的角度为系统提供结构、行为和属性等信息，它通常是对系统的静态描述。传统的软件开发方法多数面向稳定性商业环境，它要求软件开发人员在描述软件过程时预期所有可能发生的情况,并且显式地定义这些问题的解决方案，而当软件所处的环境发生变化时，软件过程无法自适应地对这些变更进行相应调整。因此，这种静态的体系结构已不能适应现在越来越多的需要在运行时刻发生变化的系统的设计需求。

动态体系结构(DSA)的出现就是为了解决传统体系结构方案的不足，它允许系统在执行过程中修改自身的体系结构，其修改过程通常也被称为运行时刻的演化(即在线演化)或动态性。目前，DSA的实现大多借助于基于构件开发（CBSD）技术和面向方面编程（AOP）技术。

CBSD将构件组装变为主要的系统构造方式,引导了软件开发从应用系统开发转变为应用系统集成，运用该技术可以有效地提高软件开发的质量和效率，降低开发及维护成本，是近几年软件工程界研究的重点。

### 1.2 动态 AOP

面向方面编程（Aspect-Oriented Programming, AOP）是近几年出现的新技术。传统的OOP擅长于用模块概念来描述对象层次关系(hierarchy)中的共同行为，而当一组互不关联的对象模块之间有一些共同行为动作需要处理时，这时就需要用到AOP技术，它能将有关功能行为表述为独立的层，而不是将这些功能嵌入到当前模块中。这种程序描述方式改善了可读性，而且能使软件维护更为便利。OOP软件开发的构思是一个从上到下的纵式发展过程，而AOP却是个从左到右的横向过程。两者是垂直交切的关系，但又有很好的互补性[8]。

织入机制是AOP的重要机制，它的实现机制有多种，基本上可以分为两类，静态织入与动态织入。静态织入是指在业务功能代码中的适当位置，比如

---

*)本文受国家 863 计划(课题编号 2006AA01Z189)，国家自然科学基金(课题编号 60473061、60473062)资助。

张夏，男，硕士生，主要研究方向：动态软件体系结构。
彭鑫，男，博士、讲师，主要研究方向：软件体系结构、软件再工程和软件产品线。
赵文耘，男，教授，博士生导师，CCF 高级会员，主要研究方向：软件工程、电子商务。

某段代码执行前或执行后，将方面中的编码插入，从而形成混合的编码。静态织入无法做到在程序运行时，根据运行上下文动态决定插入的方面代码，动态织入则可以做到这一点。动态织入可以在程序运行时，根据上下文决定调用的方面，它们的先后顺序，增加或删除一个方面等[9]。

## 2 动态体系结构实现方法

### 2.1 方法概览

软件进行在线演化意味着软件在运行期间执行代码或配置参数等发生了改变，而软件能否进行在线演化取决于软件是否具备良好的、能够适应在线演化的结构，即软件各个功能单元能够在软件运行时刻动态被更改或替换。软件发生动态演化通常是在环境发生变化时，其本身能够捕捉到这种环境变化，根据预先设计的一些演化规则进行分析判断，以决定系统下一步的演化动作。

触发软件在线演化的环境变化因素既可以是诸如温度、气压之类的物理指标，也可以是诸如服务器访问量之类的软指标。此外，在线演化所依据的演化规则要求在软件运行期间可以进行实时配置管理。

图1 表示的是一个在线演化工具内部结构，该演化工具以后台运行的方式存活于系统的整个运行周期。演化工具(EvolveTool)通过内部传感器(Sensor)感知环境变化，当环境变化满足规则(Rules)所描述的条件时，在线演化随即开始，演化动作由执行器(Executor-Control Center) 通过改变部分系统代码或配置来完成。

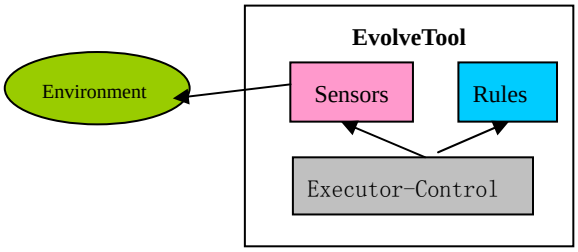


图 1 演化工具的内部结构

使用面向对象语言所开发的软件，因其语言具有较好的封装性、模块化，因而比较有利于在线演化的实施；而对于构件化的软件，因构件本身具备较好的相对独立性，在对单个构件实施在线演化时，因单个构件的变化所造成的对整个系统的影响可以限定在一个较小的范围内，因而可以使得整个系统在演化后还可以保持相对稳定性[3]。因此，基于构件的软件开发方法更易于实现软件的在线演化，后面所讨论的在线演化方法都是针对采用CBSD的构件化系统。

### 2.2 构件模型和连接器模型

构件(Component)是软件系统中具有相对独立功能的有机组成成分，是用于复用的软件实体[5]。它是软件体系结构的基本单位，通过接口与外界进行交互。

两个构件之间的连接关系必须通过连接器(Connector)进行连接，它作为构件间的交互实体在SA中扮演着重要角色。连接器有两个端口(Port)，一个是与对外请求服务构件的请求接口相连的请求端口（Request Port），一个是与对外提供服务构件的服务接口相连的服务端口（Service Port）[1]。连接器封装了构件之间的交互协议，将构件的接口需求与功能需求分割开来。

图 2 用 UML 图反映了构件、请求接口以及连接器之间的关系：

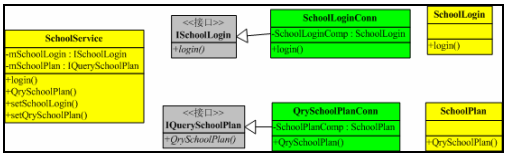


图 2 构件、连接器模型 UML 图

图2中SchoolService是请求服务的构件，SchoolLogin和SchoolPlan是提供服务的构件，ISchoolLogin 和 IQrySchoolPlan 是接口，SchoolLoginConn和QrySchoolPlanConn是连接器。

构件SchoolService要调用构件SchoolLogin和SchoolPlan的方法，通过基于接口的组装方式，SchoolService 首先定义好两个请求接口ISchoolLogin和IQrySchoolPlan，SchoolService中有类型分别为实现这两个接口的成员属性，我们称之为接口代理。SchoolService中对于SchoolLogin和SchoolPlan的方法调用，反映为对接口代理的调用。SchoolService用set方法来设置接口代理的初始值。连接器是实现请求接口的类，里面包含真正调用提供服务的构件方法的代码，由支持工具自动生成连接器的代码。

本文采用构件描述语言(Component Description Language，简称CDL)来对构件进行描述，以下给出一个请求服务构件和一个提供服务构件描述的例子：

```
<component name="SchoolService" class="wing.crgk.web.service.SchoolService" type="non-ejb">
    <description></description>
    <developer>ty</developer>
    <request-interface name="login" class="wing.crgk.web.service.ISchoolLogin">
        <method name="login" type="boolean">
            <param name="yxdm" type="String" />
            <param name="pwd" type="String" />
            <param name="pwdsMap" type="java.util.HashMap" />
        </method>
    </request-interface>
    <request-interface name="QrySchoolPlan" class="wing.crgk.web.service.IQrySchoolPlan">
        <method name="queryYxzy" type="java.util.ArrayList">
            <param name="yxdm" type="String" />
            <param name="jhk" type="java.util.ArrayList" />
        </method>
        <method name="queryYxzyOnline" type="java.util.ArrayList">
            <param name="yxdm" type="String" />
            <param name="jhk" type="java.util.ArrayList" />
        </method>
        <method name="queryZymc" type="String">
            <param name="yxdm" type="String" />
            <param name="zydm" type="String" />
            <param name="jhk" type="java.util.ArrayList" />
        </method>
    </request-interface>
</component>
```

图3 请求服务构件SchoolService的CDL描述

```
<component name="SchoolLogin" class="wing.crgk.component.SchoolLogin" type="non-ejb">
    <description>login session bean</description>
    <developer>ty</developer>
    <service-method name="login" type="boolean">
        <param name="yxdm" type="String" />
        <param name="pwd" type="String" />
        <param name="pwdsMap" type="java.util.HashMap" />
    </service-method>
</component>
```

图4 提供服务构件SchoolLogin的CDL描述

在基于接口调用的构件组装关系中，连接器作为构件连接的中介，系统设计者通过配置构件和连接器的连接关系来对应用系统的体系结构进行建模，在组装后生成的可运行的应用系统中，连接器对系统使用者来说是透明的[1]。

## 2.3 动态演化实现机制

为了实现体系结构演化，除了具备一个合理的演化工具框架之外， AOP的织入机制是实现动态演化的关键。本文为实现该技术，采用JBoss服务器，它是一个开源的符合J2EE规范的应用服务器，并且提供了AOP框架。下面将详述该织入机制的实现。

在实现AOP织入过程中，需要用到指示/拦截器(interceptor)和切分点(pointcut)。在 JBoss AOP中，用拦截器来实现指示，通过定义拦截器，让它拦截方法调用、构造器调用和域访问。切分点告诉AOP框架，哪些拦截器绑定到哪些类，哪些原数据将应用于哪些类，或者哪一个引导将被引入哪些类[8]。

连接器代码是在构件组装过程中由组装工具自动生成的，而请求服务构件（SchoolService）指向哪个连接器是在系统载入类时使用AOP技术动态添加对应的连接代码而生成的。

为了实现对某个类的拦截，需要定义一个拦截器。在JBOSS AOP中，所有的拦截器必须实现org.jboss.aop.Interceptor接口：

```
public interface Interceptor
{
    public String getName();
    public InvocationResponse invoke(Invocation invocation) throws Throwable;
}
```

在JBoss AOP中，被拦截的域、构造器和方法一律被转成通用的invoke方法调用。方法的参数被填入一个Invocation对象，而方法的返回值、域的存取以及构造器则被填入一个InvocationResponse对象,Invocation对象同时还驱动拦截链[8]。

当定义完拦截器后，需要定义切分点，用于指定将拦截器挂接到具体的类，甚至可以定义具体挂接到类的具体哪个方法。切分点写在一个配置文件中(jboss-aop.xml)，当JBoss服务器运行时，会自动加载该配置文件，以调用相关拦截器对某些类进行拦截。

以下例子演示如何实现对一个类(Test)的拦截，类Test如下所示：

```
class Test{
    private ILogin _login;
    public Test(){
    }
    public SetLogin(ILogin login){
        _login = login;
    }
}
```

类Test有一个实现接口ILogin的_login成员变量，需要在运行期间使用成员函数SetLogin指定具体使用哪一个ILogin实现类。

下面来具体探讨下这种动态演化机制，首先，需要定义一个切分点(pointcut)，如图5所示，该切分点定义了对代码中 实例化Test对象 的拦截，每次拦截都会使用到拦截器类TestInterceptor。

```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
    <bind pointcut="execution(Test->new())">
        <interceptor class="TestInterceptor" />
    </bind>
</aop>
```

图5 切分点定义

接着，需要编写拦截器TestInterceptor，用于实现在调用类Test构造函数时，自动指定一个实现了ILogin接口的成员，该拦截器定义如图6：

```
import org.jboss.aop.*;
import java.lang.reflect.*;
public class TestInterceptor implements Interceptor
{
    public String getName() { return 'TestInterceptor'; }
    public InvocationResponse invoke(Invocation invocation) throws Throwable
    {
        Test test = (Test)invocation.invokeNext();  //调用真正的操作
        test.SetLogin( new connector.Login() );
    }
}
```

图6 拦截器类TestInterceptor

如图6所示，在系统运行时刻，实例化一个Test对象时，JBoss服务器会根据切分点配置文件调用TestInterceptor的invoke函数对Test类进行拦截，在生成Test对象之后（调用invokeNext操作），使用Test对象的SetLogin方法，即可实现在系统运行期间动态指定Test类的提供服务接口(_login成员)，当运行时刻改变所提供的服务接口时，我们实质上就实现了动态演化功能。

# 3 系统实现及实例研究

下面将详述动态演化工具的技术细节及所应用到的系统的详细动态演化过程。

## 3.1 演化规则

演化工具结构如图1所示，演化工具中所用到的演化规则定义了演化工具所用到的传感器(Sensor)名称、每个具体的演化行为及触发该行为的条件。演化规则采用xml语言进行描述，xml schema如图7所示：

图7 演化规则 schema

上述schema表明：一个演化规则配置文件可包含多个规则(rule)，每个规则由演化条件(condition)和演化结果(results)组成。图 8 用xml 描述了一条演化规则：

```
<?xml version="1.0" encoding="GB2312" ?>
<rules>
    <rule name="LoginConnRule" sensor="LoginSensor">
        <condition>
            <element>student_num</element>
            <element>5000</element>
            <operator>morethan</operator>
        </condition>

        <results>
            <result>
                <operation>changeConn</ operation>
                <param>BusyLoginConn</param>
            </result>
        </results>
    </rule>
</rules>
```

图 8 演化规则

图 8 所示演化规则定义如下：演化工具使用 **LoginSensor** 构件感知环境，当 登陆学生数目

大于 **5000** 时，执行 **changeConn** 操作，参数为 **BusyLoginConn**，即让构件 StudentService 替换使用 **BusyLoginConn** 连接器。

## 3.2 实例研究

上海市成人高考网上报名系统是我们实验室为上海市教育考试院开发的一个Web应用系统。这个系统可以供考生网上填报志愿并缴纳报名费以及考试费，供各个院校网上设置招生专业计划人数、各项费用以及查看报考该院校的考生人数、个人资料以及缴费情况，我们以这个实例来详细说明此在线演化工具。

本课题组前期开发了一个构件组装工具(BSAppBuilder v2.0)，它是一个面向Web应用开发的、支持基于接口调用的集成机制的构件组装工具。用户可以通过BSAppBuilder v2.0进行Web资源管理、Web层控制流设计、构件管理、构件组装、Web应用部署等等[1]，BSAppBuilder v2.0运行界面如图9 所示：

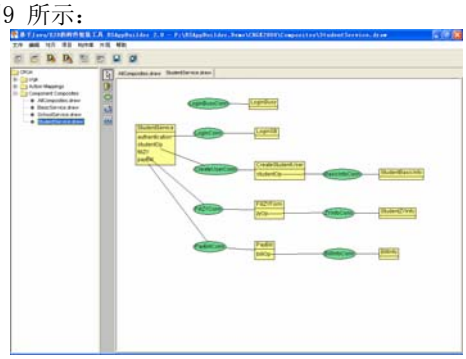

图9 BSAppBuilder v2.0 -- 构件装配

利用BSAppBuilder v2.0可进行基于构件组装的Web应用开发，在完成Web应用开发并且部署到JBoss服务器之后，整个Web系统就可投入使用，使用本文所提出的演化工具可以使得该系统能够在线演化。

以下给出一个演化实例：对于前面提到的网上报名系统，在系统设计之初，设计者并没有考虑到过多的用户同时登陆对系统负载所造成的影响，在系统投入运行一段时间之后，设计人员临时决定当学生登陆数达到一定量时，登陆验证服务使用另一构件（该构件起到让登陆用户排队等待的作用，或者仅仅显示“当前服务器忙”之类的提示信息）。这就要求负责演化功能的部件能够感知环境变化，在环境因素（如学生登陆数目）发生变化时，演化工具根据演化规则，在系统运行阶段执行在线演化。以下将详细叙述整个在线演化过程：

首先，定义切分点(如图10所示)，在系统代码中 实例化StudentService对象 时，使用拦截器类StudentServiceInterceptor对该部分代码进行拦截。

在拦截器类StudentServiceInterceptor::invoke函数中，首先调用原实例化操作(invokeNext)，得到一个StudentService对象，接着实例化一个LoginConn对象(连接器，实现了IAuthentication接口)，最后调用StudentService对象的SetAuthentication指定成员函数，这样，就实现了StudentService类指向LoginConn连接器的功能。当需要在运行期间动态改变StudentService类所指向的连接器时，只需改变切分点内所指定的拦截器类名，替换为另一个符合要求的拦截器类，JBoss服务器会重复前述操作，就可以改变StudentService类所指向的连接器，从而实现动态演化。

```
<?xml version="1.0" encoding="UTF-8"?>
<aop>
  <bind pointcut="execution(wing.crgk.web.service.StudentService->new())">
    <interceptor class="StudentServiceInterceptor" />
  </bind>
</aop>
```

图10 切分点定义

```
import org.jboss.aop.joinpoint.Invocation;
import org.jboss.aop.advice.Interceptor;

public class StudentServiceInterceptor implements Interceptor{
  public String getName() { return "StudentServiceInterceptor"; }
  public Object invoke(Invocation invocation) throws Throwable {
    try {
      System.out.println("<<< Entering MyInterceptor");
      wing.crgk.web.service.StudentService requestcomp =
          (wing.crgk.web.service.StudentService) invocation.invokeNext();
      wing.crgk.web.service.IAuthentication logincomn = new connector.LoginConn();
      requestcomp.setAuthentication(logincomn);
      return requestcomp;
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    } finally {
      System.out.println(">>> Leaving MyInterceptor");
    }
  }
}
```

图11 拦截器类StudentServiceInterceptor定义

图12、13给出了系统体系结构的演化过程，图中以Conn结尾的是连接器，其它实体都是构件。StudentService相当于一个代理构件，具体的功能需要通过连接器请求其它构件来完成。StudentService内的authentication方法用于用户验证，该功能由构件LoginSB(或LoginBusy)完成，具体使用哪个构件由StudentService使用哪个连接器来决定。图12为负载较轻时的部分体系结构，此时StudentService使用LoginConn连接器，即使用LoginSB构件；图13为负载较重时的部分体系结构，此时StudentService使用LoginBusyConn连接器，即使用LoginBusy构件。如上所述，整个系统在演化工具的配合下，改变了部分体系结构，从而实现了动态演化功能。

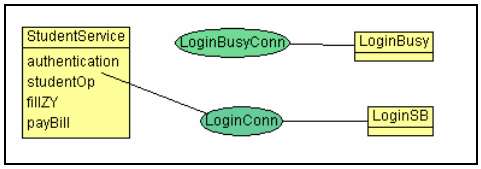


图 12 负载较轻时的部分体系结构图

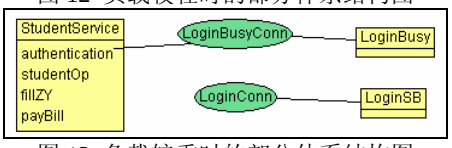


图 13 负载较重时的部分体系结构图

# 4 总结与展望

软件的在线演化是近几年软件工程的一个热点研究问题。本文借助CBSD和AOP技术，提出了一种基于动态AOP的动态体系结构实现方法，实现了一个在线演化工具。

本文所涉及的构件比较简单，未涉及关于构件状态、事务处理等方面的问题，对于一些较复杂系统的在线演化，需要考虑上述问题，以制定一种更为精细的在线演化方法;对演化规则作更全面的形式化定义，在这基础上实现一个演化规则管理器;

如何将演化规则与AOP技术里的切分点、拦截器作自动化关联，是本演化工具在下一阶段需要改进的地方。

## 参考文献


[1] 叶菲,赵文耘,彭鑫,张志. 面向Web应用开发的基于接口调用的构件组装工具. 计算机工程,2005.12

[2] 余萍,马晓星,吕建,陶先平. 一种面向动态软件体系结构的在线演化方法. 计算机学报, Vol.17, No.6, June 2006

[3] 王晓鹏,王千祥,梅宏. 一种面向构件化软件的在线演化方法. 计算机学报, Vol.28 No.11 Nov.2005

[4] 赵欣培,李明树,王青等. 一种基于Agent的自适应软件过程模型. 软件学报, Vol.15, No.3 2004

[5] NATO CO-5957-ADA. NATO Standard for Management of a Reusable Software Component Library[S]. Tokyo : NATO Communications and Information System Agency, 1991, Vol.2: 32-43

[6] 梅宏,陈锋,冯耀东,杨杰. ABC:基于体系结构、面向构件的软件开发方法. 软件学报, Vol.14, No.4 2003

[7] 梅宏,申峻嵘. 软件体系结构研究进展. 软件学报, Vol.17, No.6, June 2006

[8] Bill Burke, Adrian Brock. Aspect-Oriented Programming and JBoss. May.28, 2003

[9] A. Popovici, T. Gross, and G. Alonso. Dynamic Weaving for Aspect-Oriented Programming. Proceedings of AOSD, Enschede, The Netherlands (2002).



| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 姓名(中文) | 张　　夏 | | 姓名(英文) | | | 性 别 | | 出生年月 | |
| 职称(职务) | | | 电 话 | | | Email | | | |
| 手机 | 13585588114 | 传真 | | | 通 信 地 址 | | | | |
| 研 究 方向 | 中 文 | 动　　态　　软　　件　　体　　系　　结　　构 | | | | | | | |
| | 英 文 | | | | | | | | |
| 基 金 资助 | 中 文 | | | | | | | | |
| | 英 文 | | | | | | | | |