BuOA:一种企业级 Web 应用体系结构风格

叶 蔚[1,2],罗睿辞[1,2],刘学洋[2,3],张世琨[2,3]
(1.北京大学信息科学技术学院,北京 100871;2.高可信软件技术教育部重点实验室,北京 100871;
3.北京大学软件工程国家工程研究中心,北京 100871)

摘 要: 提出了一种针对企业级 Web 应用的体系结构风格:面向业务单元的体系结构(Business unit Oriented Architecture,BuOA).与分层体系结构风格对系统的"横向"划分不同,BuOA 将 Web 应用"纵向"分解为一组业务单元,其中每一个业务单元描述一个完整且内聚的业务功能.对业务单元之间的交互方式进行了分类,总结为四种模式:观察、注入、织入和绑定.提供了 BuOA 开发工具和运行环境.开发实例表明 BuOA 在设计、实现和部署等软件生命周期阶段能够有效地控制系统复杂性,并对企业级 Web 应用的并行开发和动态演化提供了良好的支持.

关键词: 业务单元;连接件;面向业务单元的体系结构;软件体系结构风格;模块化
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2013) 11-2120-07
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2013.11.002

BuOA:An Architecture Style for Enterprise Web Applications

YE Wei[1,2],LUO Rui-ci[1,2],LIU Xue-yang[2,3],ZHANG Shi-kun[2,3]
(1. *School of Electronics Engineering and Computer Science*, *Peking University*, *Beijing* 100871, *China*;
2. *Key Laboratory of High Confidence Software Technologies*, *Ministry of Education*, *Beijing* 100871, *China*;
3. *National Engineering Research Center for Software Engineering*, *Peking University*, *Beijing* 100871, *China*)

Abstract: This paper proposes BuOA (Business unit Oriented Architecture), a novel architecture style for enterprise Web applications. Compared with the traditional layer architecture style, BuOA vertically decomposes a Web application into a group of BUs (Business Units), each of which implements a complete and cohesive business function. To establish loosely coupled relationship between BUs, interactions between them are categorized into four patterns: observing, injecting, weaving and binding. We also provide development toolkits and runtime environment to support development and deployment of BuOA-based applications. Based on a concrete example, we show that BuOA is able to manage system complexity well on design, implementation and deployment stages in software lifecycle, with favorable support on parallel development and dynamic evolvement for enterprise Web applications.

Key words: business unit; connector; business unit oriented architecture; software architecture style; modularization

1 引言

随着计算机网络的发展和 Web 开发技术的日臻成熟,企业级 Web 应用也逐渐成为构建复杂业务系统的主流方式.企业级应用通常涉及到持久化数据、并发访问和大量的用户界面,具有动态演化和数据共享等特征[1].虽然构件化和服务化的软件开发技术为应用的复杂性控制提供了较好的支持,但是从系统的部署和构件依赖的解析机制等方面来看,企业 Web 应用整体上仍然呈现出一种单体的结构[2].因此,需要一种更加符合 Web 应用特征的体系结构来支持设计、实现和部署等软件生命周期阶段.

在 Web 开发技术和软件开发方法学的影响下,Web 应用逐步演化成为一个多层次的系统.现代 Web 应用大体上可以分为表示层(也通常称为 Web 层)、业务逻辑层和数据层.像 Enterprise Java Bean(EJB)[3]和 WSDL Web 服务[4]这样的构件模型主要在数据层和业务逻辑层上为系统开发与运行提供支持.为了在体系结构设计上为 Web 应用提供一个整体的高层描述,需要综合考虑应用各层以及各层之间的关系,给出一种更加粗粒度的抽象.

与此同时,随着分布式软件开发成为一种典型的开

收稿日期:2013-01-08;修回日期:2013-03-09;责任编辑:蓝红杰
基金项目:国家自然科学基金(No.61202070)

发组织模式[5],为了支持高效的并行开发,Web 应用需要具有更好的模块性.本文将高内聚同时与外部环境低耦合的构件称之为模块.一种常用的实践是将 Web 应用设计成层次化的 Web 模块、业务模块和数据访问模块.对于这种模块划分,模块之间的大量依赖会使得模块的并行协同开发不易控制.相比而言,从面向业务的角度出发,将贯穿各层次的一个完整功能实现作为一个模块,能够增加并行开发中模块的自身控制,简化模块开发者之间的协同.

对于 Web 应用的部署,在特定中间件的支持下,Web 应用可以在一定程度上实现业务功能的动态更新.实际应用中的功能更新通常要求应用各层的实现同时进行修改.Web 应用一般以 War 包[6]的形式进行部署,传统的中间件难以支持这种动态的整体修改,意味着系统更新需要重新部署才能生效.Web 应用的体系结构需要在运行时为业务功能的延迟绑定和动态更新提供指导及其实现机制,才能真正支持系统的动态演化.

基于以上考虑,针对 Web 应用的固有特征及其开发模式,本文通过将模块化的思想引入到设计、实现和部署等软件生命周期阶段,提出了一种新的企业级 Web 应用的体系结构风格:面向业务单元的体系结构(Business unit Oriented Architecture, BuOA),并给出了工具支持.

2　相关研究

体系结构是一个系统的基础组织,包括构件、构件之间的关系、构件与环境的关系,以及指导系统设计和演化的原则[7].体系结构风格也称为体系结构模式,描述了系统基本的结构化组织方案,可以作为具体软件体系结构的模板[8].一些体系结构风格,如层、管道和过滤器和模型-视图-控制器[9],其描述方法的研究已有不少工作[10].这些风格在 Web 应用中也有广泛的应用,通常用于解决特定上下文中的某些问题,而 BuOA 的提出则旨在为 Web 应用设计、实现和部署各阶段提供体系化的支持.

一些企业级应用的体系结构风格近年来得到了深入的研究.面向服务的体系结构(Service Oriented Architecture,SOA)包含了一个可被调用的构件集合,其中构件的接口规约可以发布和发现[11].对于 Web 应用,SOA 的实现主要关注于将核心的业务逻辑实现为 Web 服务以提高系统的灵活性,因而服务组合技术广泛应用于 SOA 的实现,但是 Web 应用内部的层次结构特征不是 SOA 关注的重点.Gartner 于 2003 年引入了"事件驱动的体系结构"(Even Driven Architecture, EDA),用以描述基于事件的设计范型.EDA 的研究可以追溯到 C2 体系结构风格[12],在 C2 中异步的通知消息和请求消息是构件之间通信的唯一途径.JB/HM 则是基于青鸟软件生产线实践提出的一种基于层次消息总线的软件体系结构风格[13].文献[14]基于 REST(Representational State Transfer)提出了面向资源的体系结构(Resource Oriented Architecture, ROA),系统中的对象被抽象为资源且赋予唯一标识符,通过 RESTful 的 Web 服务对外界提供访问.ROA 主要给出了一种 Web 层的设计原则,缺乏对业务功能总体抽象的指导.

针对实现与部署阶段的软件体系结构,学术界和工业界也提出了一系列模块化相关的技术与标准.Portlet 是一种可插拔用户界面构件,能够在 Web 门户中管理和展示[15],但是 Portlet 之间的互操作难以实现.OSGI 是一种构建动态模块化软件的技术,为分布式和嵌入式应用提供了一种模块化的体系结构[16].服务构件体系结构(Service Component Architecture, SCA)提供了一组规范,将业务功能作为一系列服务来提供,并将这些服务组合到一起以创建满足特定业务需要的解决方案[17].SCA 是基于 SOA 的模型,因此前面所述的 SOA 在支持 Web 应用方面的局限性也体现在 SCA 中.

可见,目前在 Web 应用的模块化设计、并行开发以及动态部署和演化方面仍然缺乏一种统一体系结构风格的指导.接下来本文将通过一个三维视角对 Web 应用进行分析,尝试提出一种更加符合企业级 Web 应用特征的体系结构风格.

3　面向业务单元的体系结构

3.1　概述

BuOA 的提出的初衷是从面向业务的角度出发,通过将模块化的思想引入到设计、实现和部署等软件生命周期阶段,控制复杂性的同时,增强各阶段开发或演化的灵活性.BuOA 的设计目标包括:(1)在设计阶段,遵循"关注点分离"原则,一个构件实现 Web 应用中一个完整且内聚的业务功能,构件之间具有相对简单的连接关系;(2)在实现阶段,构件的开发相对独立且易于集成,构件之间连接关系的实现可以通过灵活的配置完成;(3)在部署阶段,构件的接口能够实现动态的绑定,构件能够进行在线的更新、删除或者替换.

BuOA 中的构件模型称之为业务单元(Business Unit, BU),代表了 Web 应用中实现了完整且内聚业务功能的基本单元.以业务单元为系统的基本元素,我们进一步将业务单元之间的关系提炼为四种模式:观察(Observing)、注入(Injection)、织入(Weaving)和绑定(Binding).由此我们得到 BuOA 的定义:BuOA 是一个业务单元集合,业务单元之间的任何一个连接都属于观察、注入、织入和绑定四种模式中的一种,如图 1 所示.

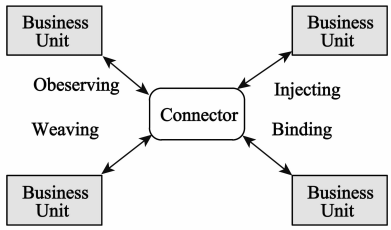


图1 BuOA概览

3.2 构件模型

为了明确业务单元的形态,需要对 Web 应用本身的构造结构进行分析.从层次化的视角来看,Web 应用通常被划分成表现层、业务逻辑层和数据访问层.从面向业务的视角来出发,Web 应用则可以分解为一组业务功能.从交互的视角来看,Web 应用具有良好定义的接口来实现与外部的互操作.而 Web 应用的接口应该具有一个属性集合表示其内部状态,一个操作集合来操控内部状态,以及一个事件集合来表示状态的改变.这三个视角呈现出正交的形态,将这三个视角整合到一起则形成了 Web 应用的一种三维视图,其中三个维度分别为层次维、功能维和接口维.如图 2 所示.

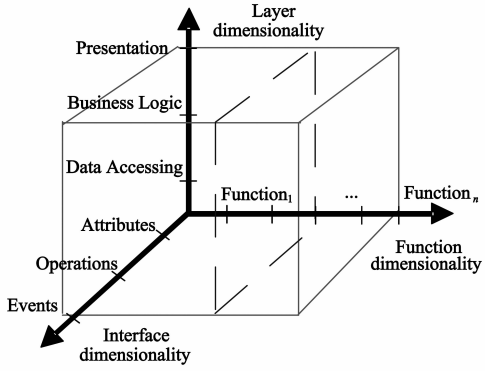


图2 企业级Web应用的三维视角

作为一个完整且内聚业务功能的抽象,业务单元所表达的正是功能维度的一个切面,如图 2 中的虚线框所示.从这个三维视图可见,业务单元的内部结构包含了表现层、业务逻辑层和数据访问层,对外暴露属性、操作和事件作为其接口,接口是通过内部结构实现.业务单元作为连接件中定义的某种角色,与其它业务单元连接.业务单元模型通过 UML 符号体系描述如图 3 所示.

下面从另外两个维度来进一步探讨业务单元的内涵.图 4 中虚线框所表示的两个切面,一个是接口维的切面,一个是层次维的切面.针对接口维在“Operations”点的切面,Web 应用的操作集合从功能维看,分布于各个业务单元中;从结构维看,分布于表示层(如一个 HTTP Post 方法)、业务逻辑层(如一个 Web 服务)和数据层(如 Java 数据访问对象的方法).针对层次维在“Data Accessing”的切面,数据访问层从功能维度看,分布于各个业务单元中;从接口维看,数据访问层能够提供属性集(如数据连接的配置属性)、操作集(如数据访问对象的方法)和事件集(如数据更新或者异常事件).

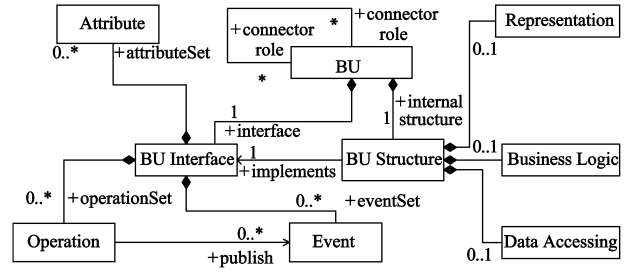


图3 业务单元模型

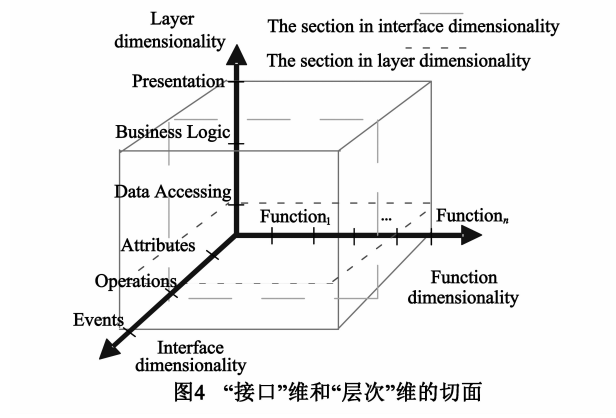


图4 “接口”维和“层次”维的切面

以功能维切面为出发点,通过将业务功能整体进行模块化抽象,我们得到了一种面向业务单元的体系结构风格.如果我们从另外两个维度出发,可以很自然地找到与其它体系结构风格的对应.例如,如果从层次维出发,关注各层的抽象以及各层的关系,则与传统的分层体系结构风格相吻合;如果从接口维出发,考虑“Operations”之间的协作,则思路与面向服务的体系结构是相似的;考虑基于“Events”的交互方式,则能得到一种事件驱动的体系结构.

BuOA 并非代替这些体系结构风格,以 BuOA 为贯穿软件生命周期的主体风格,其他风格在 Web 应用的设计与实现中能够作为补充有机地融合进来.

3.3 连接件模型

观察、注入、织入和绑定四种关系模式各对应一种连接件类型,构成了 BuOA 中的连接件模型.我们分别用 B、E、O 和 A 表示一个体系结构中的业务单元集合、事件集合、操作集合和属性集合.对于一个特定的业务单元 $b\in B$,用 $E(b)$、$O(b)$和 $A(b)$表示其事件集合、操作集合和属性集合.

3.3.1 观察

"观察"连接件基于"发布/订阅"范型支持 *BU* 之间事件驱动的交互.它定义了两种角色:Publisher 和 Subscriber.如图 5 所示,Publisher 和 Subscriber 之间的连接通过"订阅规则"(subscription)描述,订阅规则定义为三元组:

(*subscriber*, *event*, *operation*),其中 $subscriber \in B$, $event \in E$, $operation \in O(subscriber)$

表示当 event 事件(由某个 Publisher 发布)发生时,subscriber 的操作 operation 将被调用.基于"观察"连接件,Publisher 和 Subscriber 彼此透明,从而松散耦合.

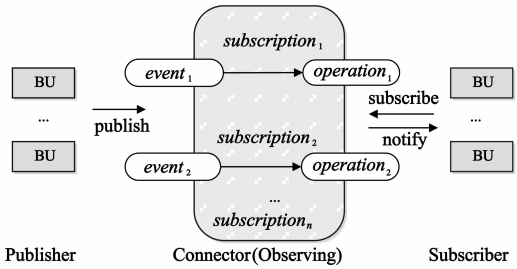


图5 "观察"连接件

3.3.2 注入

"注入"连接件基于"控制反转"[18]的思想管理业务单元之间操作调用的依赖.它定义了两种角色:Provider 和 Consumer.如图 6 所示,Provider 和 Consumer 之间的连接通过"依赖"(dependency)描述,依赖定义为三元组:

($consumer$, $operation_i$, $operation_j$),其中 $consumer \in B$, $operation_i \in O(consumer)$, $operation_j \in O$

表示 consumer 的 $operation_i$ 操作依赖于 $operation_j$ 操作(由某个 Provider 提供),连接件将 $operation_j$ 动态注入到 $operation_i$ 中.

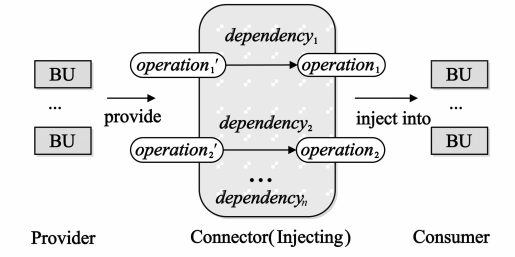


图6 "注入"连接件

3.3.3 织入

"织入"连接件基于"面向切面编程"(Aspect Oriented Programming,AOP)[19]中的概念,表达实现特定关注点的业务单元与应用该关注点的业务单元之间的关系."织入"连接件定义了两种角色:Adviser 和 Target.Adviser 和 Target 之间的连接通过"切面"(aspect)描述,切入点定义为三元组:

(*operation*, *target*, *pointcut*),其中 $operation \in O$, $target \in B$ pointcut 为 target 所包含操作的表达式,定义织入的条件和方式,pointcut 的形式化描述不在本文关注的范围.上述定义表示 operation(由某个 Adviser 提供)将以基于 pointcut 织入到 target 的操作调用中.基于"织入"连接件,业务单元能够以 *AOP* 的方式动态增加横切行为(如日志和事务等).

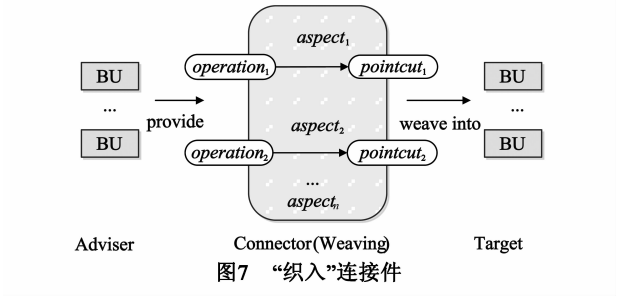


图7 "织入"连接件

3.3.4 绑定

"绑定"连接件特定于业务单元的 Web 表示层.一个业务单元在 Web 层通常对应于应用 URL 命名空间的一个子空间."绑定"连接件定义了两种角色:Slave 和 Master.Slave 和 Master 之间的连接通过"绑定"(binding)描述,绑定定义为三元组:

(*slave*, *attribute*, *master*),其中 $slave \in B$, $master \in B$, $slave \neq master$, $attribute_i \in A(slave)$

表示 slave 将 attribute 所表示的 URL 命名子空间绑定到 Master 维护的全局空间(通常为 URL 根路径).如图 8 所示,BuOA 中具有唯一的 Master.Salve 能够动态绑定到 Master,使得应用在 Web 层易于进行功能扩展.

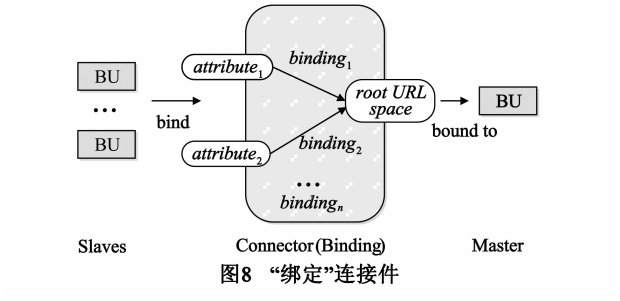


图8 "绑定"连接件

基于业务单元接口和连接件模型,通过"订阅规则","依赖","切面"和"绑定"的定义就能够以配置的方式描述业务单元之间交互.

4 工具支持

4.1 业务单元运行实体

为了将 BuOA 对 Web 应用的支持扩展到实现与部署阶段,需要进一步提供 BuOA 的开发工具与运行环境.我们选择将 OSGi 作为业务单元实现的底层技术.OSGi Bundle 的元数据规范、服务和事件机制,分别可用

于实现业务单元接口中的属性、操作和事件;而 OSGi Bundle 动态化和模块化特征也能够为 Web 应用的部署和演化提供支持.我们将业务单元分为三个 bundle:一个 Web bundle、一个 Service bundle 和一个 Persistence bundle.它们以 Virgo Plan[20]的形式组成一个独立运行时单元.三个 bundle 暴露的接口集合共同组成了业务单元的接口.

4.2 连接件运行实体

连接件的运行实体通常包含在中间件平台中."注入"和"织入"连接件的实现需要一个 OSGi 环境下的 IoC 和 AOP 容器,而"观察"连接件的实现需要支持 Bundle 之间基于事件的通信.Spring DM(Spring Dynamic Modules)[21]在 OSGi 的服务层提供了上述功能.因此,我们基于 Spring DM 来实现这三种连接件,通过灵活的 XML 配置即可定义业务单元之间的订阅、依赖和切面.

针对"绑定"连接连接件的实现,一种方法是在 OSGI 中使用 HttpService 服务[16],基于该服务将每个 BU 注册到一个 URL 子空间.但是 HttpService 存在一些限制,比如不支持 HTTP 过滤器和 HTTP 监听器,而且由于类加载机制问题不能保证传统 MVC 框架正常运行.我们采取了另外一种方法,引入了 Virgo Snaps[21]来为业务单元分配 URL 空间.Virgo Snaps 建立了 OSGi 容器和 Web 服务器之间的适配机制,使得 bundle 之间可以共享 Web 上下文.

4.3 BuOA 运行环境与开发工具

我们以 Virgo 服务器(集成了 Spring DM)作为业务单元运行的中间件,同时实现了 BuOA 的开发框架 Eleven 和相应的开发工具.Eleven 屏蔽了 Spring DM 和 Snaps 等底层实现技术,并提供了若干企业级 Web 应用的公共服务,如业务单元安装、认证服务以及默认的用户模型等功能.开发框架与 Virgo 服务器共同构成了 BuOA 的运行环境.

我们提供了三种 Maven[22]脚手架来生成各层的 bundle,允许开发者通过简单的配置创建业务单元的预定义内容,生成可执行的代码骨架.BuOA 开发工具减少了开发者的重复工作,屏蔽了 BuOA 底层实现的细节,使得开发者只需关心业务逻辑的实现.这样我们就为 Web 应用设计、实现与运行提供了整体的支持,如图 9 所示.

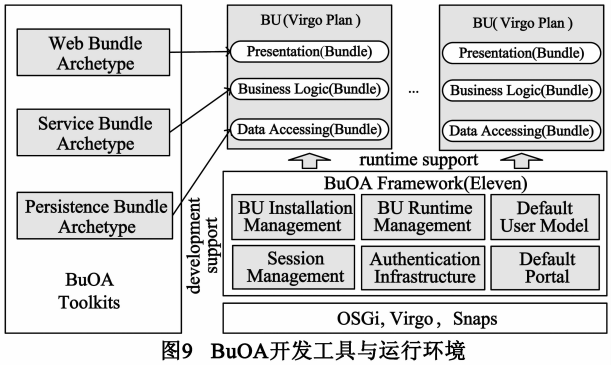


图9 BuOA开发工具与运行环境

5 实例验证

5.1 Web 应用实例概述

本节基于一个工程实例来验证 BuOA 模型与工具对于企业级 Web 应用的支持.这一应用是一个支持企业进行设计协同的 Web 系统.我们基于 BuOA 将系统分解为 13 个业务单元,业务单元的接口以及它们之间的关系通过四种连接件模型进行了设计.限于篇幅,我们选择其中 3 个业务单元进行描述,如图 10 所示.

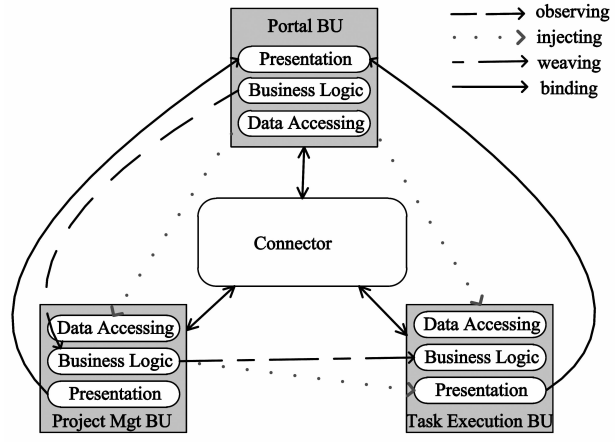


图10 应用实例描述

在这一实例中:

- 具有三个业务单元,分别为门户(portal),项目管理(projectMgt)和任务执行(taskExe);
- projectMgt 具有针对特定项目的认证规则,portal 会通过"观察"连接件监听 projectMgt 的认证服务,而且一旦 projectMgt 的认证服务安装或者升级,portal 也会更新整个 Web 应用的认证规则;
- projectMgt 通过一个活动记录器(activityLogger)记录项目中所有类型的活动.活动记录器服务通过"织入"taskExe 用来记录所有的任务执行信息;
- taskExe 的 Web bundle 中的控制器更新任务状态的时候,需要"注入"projectMgt 中的服务进行必要的项目信息更新;
- portal 是"绑定"连接件中的 Master, projectMgt 和 taskExe 是 Slaves.

5.2 实例的实现

5.2.1 业务单元实例

业务单元实现为 Virgo plan,通过 XML 文件描述其包含的各层 bundle.比如,projectMgt 的 XML 描述如下所

示.

```
< plan name = "projectMgt" version = "1.0.0" >
< artifact type = "bundle"
  name = "projectMgt. persistence" version = "1.0.0"/ >
< artifact type = "bundle"
  name = "projectMgt. service" version = "1.0.0"/ >
< artifact type = "bundle"
  name = "projectMgt. web" version = "1.0.0"/ >
< /plan >
```

每个 bundle 的属性通过它的 Manifest.mf 文件来配置和扩展,操作和事件则通过 Spring DM 的配置文件来定义.

5.2.2 连接件实例

portal 包含一个 AuthenticationFilter 服务,该服务通过一组 AuthenticationInterceptor 实例来执行认证操作.当 projectMgt 需要增加一种特定认证规则时,projectMgt 只需增加 AuthenticationInterceptor 接口的一种实现即可,*portal* 会观察系统上下文中的 AuthenticationInterceptor 服务的变化并进行自动更新.这一"观察"连接转化为 Spring DM 配置,如下所示.

```
< osgi: list id = "AuthenticationInterceptor"
  interface = "org. eleven. web. AuthenticationInterceptor >
  < osgi: listener bind-method = "onBind"
   unbind-method = "onUnbind" ref = "authentica-
tionFilter"/ >
< /osgi: list >
```

Spring DM 的事件机制允许一个 bundle 在生命周期中的状态更新得到及时通知,所以能够有效支持"观察"连接件的实现.而对于"注入"和"织入"连接件的实现,Spring DM 的编程与配置模型尤为适用,限于篇幅不再赘述.

对于"绑定"连接件,通过 Virgo Snaps 将 projectMgt 和 taskExe 绑定到 portal,任何对于 URL 根路径"/"的请求都会被 portal(即 Master)直接处理,而对"/project"和"/task"的请求则会被分派到 projectMgt 和 taskExe 处理.各个业务单元可使用完全不同的 Web 技术来处理 Web 请求,增加了实现的灵活性.

5.3 讨论

应用的第一个版本以传统的层次体系结构为主体风格进行设计和开发.原系统设计为 18 个模块,包括 1 个数据访问模块,1 个 Web 模块和 16 个服务模块,分为三个项目存在.模块之间主要通过 Spring 实现静态的"注入"连接.在开发中我们遇到了以下问题:

(1)开发人员为了测试系统必须检出所有代码进行编译和部署,非常耗时;

(2)一个业务功能的修改,涉及多个开发人员负责的模块,容易造成冲突,协同效率低;

(3)系统小范围修改也需要整个系统重新部署以及重新启动,系统维护困难.

在第二版本进行功能升级的相关开发时,我们基于 BuOA 对系统进行了重构.

5.3.1 对设计的支持

我们将系统重新设计为 13 个业务单元,一共包含 38 个 OSGi Bundle.少部分业务单元可能包含一个以上的业务逻辑层 bundle,或者没有数据层 bundle.在新的设计中,业务单元之间的连接数量(4 种连接件的实例数)为 228 个,而重构前模块之间的连接数量(直接依赖数)为 375 个.同时,连接关系的实现从静态绑定转换为了动态绑定.模块之间不仅具有更低的耦合性,同时具有更好的动态性.

以 BuOA 作为主体风格,其他风格在 Web 应用的设计与实现中也能够作为补充有机地融合进来.例如:(1)在业务单元内部依然采用层次体系结构;(2)业务单元的操作之间遵循 SOA 的协作范型;(3)业务单元 Web 层接口基于 ROA 风格设计,实现则采用了 MVC 和管道风格;(4)观察连接件的应用是 EDA 的一种实现.以 BuOA 作为企业级 Web 应用的贯穿生命周期的基础体系结构,可以有效地融合其他体系结构风格.

5.3.2 对开发的支持

系统由 6 名开发人员实现,每人负责 2 到 3 个业务单元的开发.最终的实现包含了十二万行代码,其中 Java 代码占 65%,JavaScript 代码占 15%,JSP 代码占 8%.而 XML 代码在新版本中的比例增加到了 7%,因为业务单元的接口描述和连接关系大部分通过 XML 配置实现.基于 BuOA,开发时具有以下优势:(1)一个业务单元的开发由一个开发人员负责,开发的自主控制性更好,并行开发效率更高;(2)Eleven 提供了强大的基础设施,开发人员只需关注业务功能的实现;(3)持续集成更为简单,每个业务单元可以被单独构建和测试.

5.3.3 对部署与演化的支持

在部署和演化的支持上我们获得了以下优点:(1)基于 Maven 脚本直接从业务单元仓库中动态下载业务单元即可部署;(2)业务单元能够动态的安装、启动、停止、更新和卸载,而整个系统无需重启;(3)基于连接件模型,业务单元中的操作可以动态的注册、获取和监听服务,这提供了在线演化的有效机制.

6 总结

本文从面相业务的角度出发,将模块化的思想引入到设计、实现和部署等软件生命周期阶段,给出了一种针对企业级 Web 应用的体系结构风格 BuOA.基于 Web 应用的一种三维视图,提出了 BuOA 的构件模型

"业务单元";通过提炼业务单元之间的交互模式,讨论了 BuOA 的连接件模型.本文同时给出了 BuOA 的工具支持与运行环境,进行了实例验证.未来我们将进一步完善 BuOA 模型,比如在连接件模型上支持 Web 异步请求这样的细粒度交互的抽象.此外,我们希望基于 BuOA 引入业务过程建模与管理的相关技术.

参考文献

[1] M Fowler. Patterns of Enterprise Application Architecture[M]. Addison-Wesley, 2002.

[2] S R Kaegi, D Deugo. Modular java Web applications[A]. Proceedings of the 2008 ACM symposium on Applied computing [C]. New York: ACM, 2008. 688 – 693.

[3] Oracle Inc Enterprise JavaBeans Technology[S/OL]. http://www. oracle. com/technetwork/java/javaee/ejb/index. html, 2013-01-09.

[4] W3C. Web Services Description Language (WSDL) 1.1[S/OL]. http://www.w3.org/TR/wsdl, 2013-01-09.

[5] B Sengupta, S Chandra, V. Sinha. A research agenda for distributed software development[A]. Proceedings of the 28th International Conference on Software Engineering [C]. New York: ACM, 2006. 731 – 740.

[6] Oracle Inc. Web Application Archive[OL]. http://docs.oracle.com/javaee/5/tutorial/doc/bnadx.html, 2013-01-09.

[7] IEEE. IEEE recommended practice for architectural description of software-intensive systems[S]. Washington: IEEE Computer Society, IEEE Std1471-2000, 2000.

[8] Garlan D. What is style? [A]. Proceeding of the Dagshtul Workshop on Software Architecture [C]. http://www. cs. cmu. edu/afs/cs/projec/able/ftp/style iwass95/style-iwass95. Pdf, 1995.

[9] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. Pattern-Oriented Software Architecture. Vol. 1: A System of Patterns[M]. West Sussex: John Wiley&Sons, 1996.

[10] 谭佳,孙艳春,梅宏.ABC 方法中体系结构风格建模的研究[J].电子学报,2006,34(5):969 – 976.
Tan Jia, Sun Yan-chun, Mei Hong. Modeling architecture style in ABC methodology [J]. Acta Electronic Sinica, 2006, 34(5): 969 – 976. (in Chinese)

[11] Min Luo, M Endrei, P Comte, P Krogdahl, J Ang, T Newling. Patterns: Service-Oriented Architecture and Web Services[M/OL]. http://www. redbooks. ibm. com/redbooks/pdfs/sg246303.pdf, IBM Redbooks publication, 2004.

[12] RN Taylor, N Medvidovic, KM Anderson, et al. A Component-and message-based architectural style for GUI software [J]. IEEE Transactions on Software Engineering, 1996, 22(6): 390 – 406.

[13] 张世琨,王立福,常欣等.基于层次消息总线的软件体系结构描述语言[J].电子学报,2001,29(5):581 – 584.
Zhang Shi-kun, Wang Li-fu, ChangXin, et al. Hierarchical message bus-based software architecture description language [J]. Acta Electronic Sinica, 2001, 29(5): 581 – 584. (in Chinese)

[14] Leonard Richardson, Sam Ruby. RESTful Web Service[M]. O'REILLY Press. 2007.

[15] Oracle Inc Portlet Specification (JSR-000168)[S],. http://www.jcp.org/aboutJava/communityprocess/final/jsr168/.

[16] OSGi Alliance. OSGi Service Platform Release 4[S/OL]. http://www. osgi. org/Download/Release4V40, 2013-01-09.

[17] OASIS. Service Component Architecture [S/OL]. http://www.oasis-opencsa.org/sca, 2013-01-09.

[18] M Fowler. Inversion of control containers and the dependency injection pattern[OL]. http://www. martinfowler. com/articles/injection.html, 2004/2013-01-09.

[19] G Kiczales, J Lamping, et al. Aspect-oriented programming [A]. Proceedings of the European Conference on Object-Oriented Programming (ECOOP)[C]. Springer-Verlag LNCS 1241. 1997. 220 – 242.

[20] SpringSource Inc. Virgo[OL]. http://www.eclipse.org/virgo/, 2013-01-09.

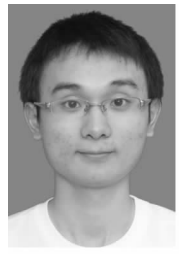
[21] Eclipse Foundation. Spring Dynamic Modules[OL]. http://www.springsource.org/osgi, 2013-01-09.

[22] Apache Software Foundation. Maven[OL]. http://maven.apache.org, 2013-01-09.

作者简介

叶 蔚 男,1985 年生于江西全南.2011 年于北京大学信息科学技术学院获得博士学位,现为北京大学信息科学技术学院博士后,主要研究领域为基于 Web 的软件工程,软件体系结构,应用集成等.

E-mail: wye@pku.edu.cn



罗睿辞 男,1988 年生于湖南株洲.北京大学信息科学技术学院博士研究生,主要研究领域为基于 Web 的软件工程与软件体系结构.