基于构件技术的应用框架元模型的研究*

胡文蕙+, 赵 文, 张世琨, 王立福

(北京大学 信息科学技术学院,北京 100871)

Study of Application Framework Meta-Model Based on Component Technology

HU Wen-Hui+, ZHAO Wen, ZHANG Shi-Kun, WANG Li-Fu

(School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

+ Corresponding author: Phn: +86-10-62761900, E-mail: huwh@cs.pku.edu.cn, http://www.cs.pku.edu.cn

Received 2003-02-21; Accepted 2003-04-15

Hu WH, Zhao W, Zhang SK, Wang LF. Study of application framework meta-model based on component technology. *Journal of Software*, 2004,15(1):1~8.
http://www.jos.org.cn/1000-9825/15/1.htm

Abstract: This paper proposes an application framework meta-model based on UML (unified modeling language) notation from the perspective of construction and composition of application frameworks, gives the semantic of framework elements, and specially discusses the hot-spots' description methods and their implementation mechanisms. Finally, the paper presents an example of the application framework meta-model, i.e., the telecommunication integrated business system framework, and proposes a reuse method of the application framework.

Key words: framework; framework meta-model; hot spot

摘 要: 从应用框架构造和组成的角度,使用 UML(unified modeling language,统一建模语言)符号体系,提出了框架元模型,给出了框架内部组成元素的语义,特别是对扩展点的表示和实现机制进行了详细的讨论.最后以电信综合营业系统框架为例,给出了框架元模型的一个实例,并给出了应用框架的复用方法.

关键词: 框架;框架元模型;扩展点

中图法分类号: TP311 **文献标识码**: A

对于应用框架,一直以来没有一个统一的定义,下面给出两个最常用的定义:(1) "框架是一个系统全部或者部分的可复用设计,通常由一组抽象类和类之间的协作组成[1]".(2) "框架是一个能够被开发人员实例化的系统骨架[2]".这两个定义是相互补充的,前者从复用的角度描述了框架,而后者给出了框架的目的.根据以上两个定义可知:(1) 框架既分割了应用领域中的类、定义了各部分的主要责任和类与对象的协作关系,还规定了控制流

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2001AA113171 (国家高技术研究发展计划(863))

作者简介: **胡文蕙**(1977－),女,山西五寨人,博士生,主要研究领域为软件工程,应用框架及构件技术;**赵文**(1967－),男,博士生,主要研究领域为软件工程,工作流及相关技术;**张世琨**(1969－),男,博士,副教授,主要研究领域为软件工程,软件体系结构;**王立福**(1945－),男,博士,教授,博士生导师,主要研究领域为软件工程,信息安全技术.

程;(2) 框架记录并实现了其应用领域的主要公共设计决策.由此可以认为,框架是一个“部分实现”的软件体系结构,是支持软件设计复用和实现复用的技术.

人们最常开发和使用的框架有两种:面向对象框架(object-oriented framework,简称 OOF)[3]和基于构件的框架(component based framework,简称 CBF).OOF 的复用是通过对框架中的抽象类进行特殊化的方式来定义框架行为的,每一个抽象类派生一个子类,并在子类中给定所有纯虚方法的具体实现,然后就可以复用这些具体的子类来开发特定应用系统.因此,OOF 是基于继承的框架,也称为白盒框架.随着 20 世纪 90 年代构件技术的兴起,出现了基于构件的框架(CBF),即将基于继承的面向对象框架通过用构件接口中方法的调用来替代对象类中方法的重载,转换成为基于构件的框架.基于构件的框架由互相协作的构件组成,并通过对构件接口的扩展来实现应用系统.相对于基于继承的框架,CBF 又被称为基于组装的框架.由以上简单分析可知,OOF 和 CBF 之间的主要区别在于实现技术和提供扩展点的机制不同.

本文从框架构造和组成的角度出发,使用 UML 符号体系,给出了基于构件技术的应用框架元模型,描述了组成框架的元素、框架的边界元素以及框架与元素、元素与元素之间的关系,并且结合电信综合营业系统框架的实践,给出一个框架元模型的具体实例.

1 框架元模型

本文通过对 IBM San Francisco 商业应用框架[4]和 Taligent 公司 CommonPoint[5]的分析研究,以及青鸟商业领域应用框架和电信综合营业系统框架的实践,提出了基于构件技术的应用框架元模型,如图 1 所示.

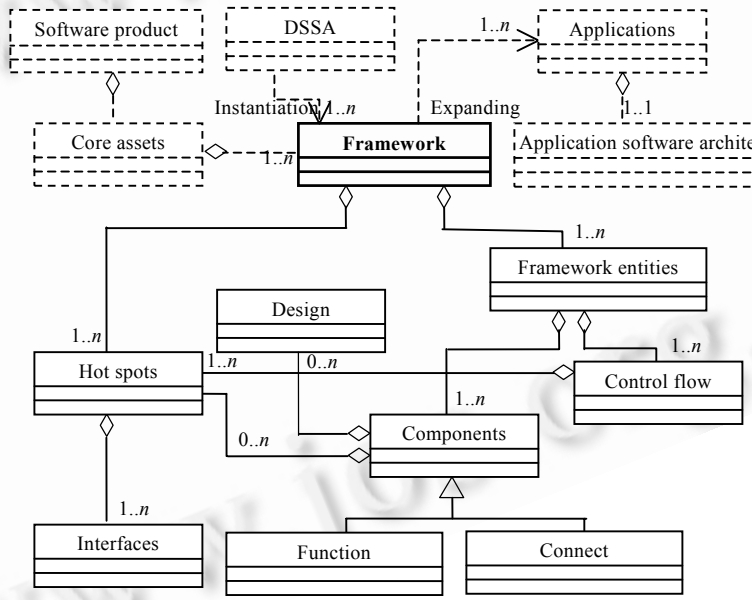

Fig.1 Application framework mate-model

图 1 应用框架元模型

位于图 1 上半部分的虚线矩形标识了框架的边界元素,并以虚线表示它们与框架之间的关系,但是这些边界元素并不是框架的组成部分.位于图 1 下半部分的实线矩形标识了框架的组成元素,以实线表示框架与组成元素以及组成元素之间的关系.

1.1 框架的边界元素

1.1.1 软件产品线、核心资产库

一个软件产品线为一组具有相似系统需求的软件系统提供了适当的开发环境,这些需求是针对一类相似的商业行为或市场部门的[6].核心资产库是软件产品线的基础,包括软件产品线开发过程中产生和相关的一切

软件制品,例如领域模型、领域知识、产品线体系结构、工作流程和应用生成器等.应用框架作为一类重要资产是软件产品线核心资产库的组成部分,框架的设计和生产属于资产库建设的范畴.

1.1.2 DSSA(domain-specific software architecture,特定领域软件体系结构)

同一领域中系统的需求必然具有显著的共性,领域模型描述了需求上的共性,而 DSSA 针对领域模型提出了解决方案.对于需要开发许多相似应用系统或升级较为频繁的领域,建立针对共同需求的软件体系结构将促进软件复用而得到显著的回报,这样的体系结构就是 DSSA.DSSA 不是单个系统的表示,而是能够适应领域中多个系统的需求的一个高层次的设计.它包含构件以及构件互联的规则.当开发本领域的一个新系统时,可以使用这些构件,并且按照这些规则构成满足当前系统需求的特定的系统结构.

DSSA 侧重于描述应用系统中包含哪些构件、构件之间的接口是怎样的,说明了如何将系统功能分配到构件和构件之间.而框架是 DSSA 的部分实现,支持基于 DSSA 的应用系统的开发.

1.1.3 应用软件体系结构、应用

框架实现了应用领域中主要的公共设计决策,一个框架可以扩展成为多个应用系统.每个应用系统都是在框架的基础上进行适应性扩展,根据应用系统的特性形成完整的可执行系统,而且每一个应用系统都与惟一的软件体系结构相对应.

1.2 框架的内部元素

1.2.1 扩展点

语义.

框架中支持灵活扩展和定制的机制称为框架的扩展点,支持符合应用系统特定需求的实现[7].扩展点为实现领域变化性提供了必要的支持,这里的变化性主要体现在以下两个方面:(1) 被领域模型封装且没有给出设计细节;(2) 框架设计阶段给出了解决方案,但在实现阶段没有提供代码.

语法.

扩展点作为一类扩展机制,是框架能否支持大粒度复用和为用户提供足够灵活性的关键所在.根据以上给出的两个方面的变化性和特定领域应用系统的固有特性,可以把扩展点分为主要的 4 类:数据、功能、界面和业务过程扩展点.针对不同扩展点的具体特点,可以采用不同机制实现不同类型的扩展点,下面给出不同扩展点的描述方法,并举例说明扩展点的实现机制.

(1) 数据扩展点

数据扩展点主要针对以下一类变化性:领域模型虽然定义了操作的名称和含义,但是操作应用的对象和操作实施的方法并没有确定.因此,为了完成相同的操作需要不同的属性支持,例如,同样是计算图形的面积,计算圆的面积需要圆的半径,而计算矩形的面积需要知道矩形的长和宽的值.这种情况下需要设置抽象类,抽象类中定义了通用的属性和操作,应用系统开发者需要继承抽象类,在子类中定义新的属性,并实现抽象类中的抽象函数.

操作是构件提供的接口功能,其实现细节被构件封装,对操作内部实现细节的改变并不影响操作与其他构件的交互,因此,对数据扩展点的描述只需要包括构件名、定义操作的抽象类名、操作名、已实现的子类名等.假设文中提到的构件符合 COM 构件标准,一个构件提供若干个接口,每一个接口包含若干个操作.

HOT_SPOT Hot Spot Name
{
TYPE: ⟨Data⟩ //数据扩展点
COMPONENT_ABSTRACT CLASS_OPERATION: //定义操作的抽象类
⟨Component Name, Class Name, Operation Name⟩
⟨Parameter 1, Parameter 2,…, Parameter *n*⟩ //操作的参数列表
SUBCLASSES_LIST: //实现操作的子类列表
⟨Subclass Name 1: description about subclass 1⟩ //描述该子类的适应环境
⟨Subclass Name 2: description about subclass 2⟩

…

DESCRIPTION: ⟨Description Name⟩ //必要的文字描述

OPTIONAL: //可选项

DIAGRAM_PRESENTATION: //扩展点的图形描述

⟨Diagram Name, File Name, Directory Name⟩

}

(2) 功能扩展点

功能扩展点侧重于支持同一个系统中的同一个功能在不同环境下不同的实现方法的动态替换.功能扩展点可以通过对框架中功能构件接口的操作来实现,下面是一个简单的功能扩展点实现示例,并使用UML符号体系来描述扩展点的形式和使用方式.

构件 InvoiceManager 通过接口 Printer 对外提供打印发票 PrintInvoice()的功能,构件 Calculator 通过接口 Calculator 对外提供计算税率 CalRate()的功能,PrintInvoice()需要调用 CalRate()实现计算税率的功能.但是在不同应用系统中,发票具有不同的打印格式,而且随着环境的改变,计算税率的方法也有差异.如图 2 所示,接口 Printer 并不直接实现其操作,而是由 Printer1 或者 Printer2 实现.框架可以为接口提供一个缺省的实现 Printer1,框架使用者在扩展框架的过程中可以对 Printer1 进行适应性修改,或者重新实现 Printer2,而运行时刻该调用哪一个具体的操作由构件 InvoiceManager 决定,可以支持运行时刻的适应性修改.同理,为接口 Calculator 提供了两者不同的计算方法 WeekendRateCalculator()和 WorkdayRateCalculator(),构件 InvoiceManager 的操作 PrintInvoice()在运行时刻,根据上下文环境的需要动态决定调用哪一个具体的方法.该例采用了 Strategy 设计模式[8],可以为数据对象提供一组相互独立的、可以替换的算法,其层次结构可以帮助将系统中的公共功能抽象出来,使其容易被使用、修改和替换.功能扩展点还可以利用其他设计模式进行设计,例如 Decorator 模式[8]可以解决动态地给一个构件添加额外功能的问题,就扩展功能而言,Decorator 模式比生成子类的方式更为灵活.

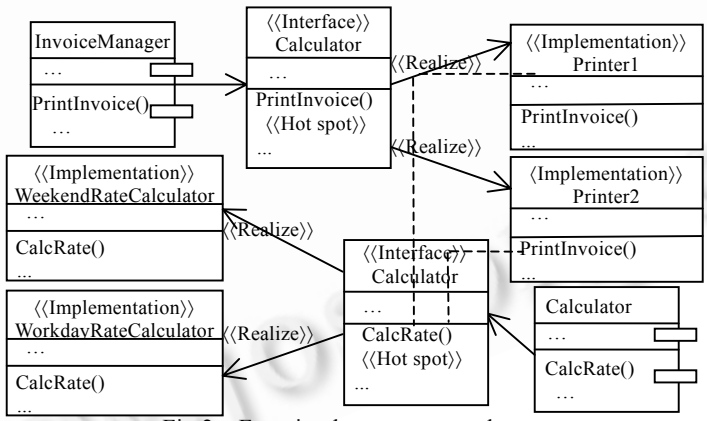

Fig.2 Function hot-spot example

图 2 功能扩展点示例

无论是否采用设计模式的设计思想,都可以用明确标识实现扩展点的构件接口的方法来描述功能扩展点,下面给出功能扩展点的描述.

HOT_SPOT Hot Spot Name

{

TYPE: ⟨function⟩ //功能扩展点

COMPONENT_INTERFACE_OPERATION: //定义扩展点的接口

⟨Component Name, Interface Name, Operation Name⟩

⟨Parameter 1, Parameter 2,…, Parameter *n*⟩ //操作的参数列表

IMPLEMENTATION_LIST: //已实现扩展点的类列表

⟨Class Name 1: description about implementation 1⟩

〈Class Name 2: description about implementation 2〉

…

CALLING_COMPONENT_ LIST: //与其他哪些构件进行交互

〈Component Name 1, Interface Name 1, Operation Name 1〉

〈Component Name 2, Interface Name 2, Operation Name 2〉

…

DESCRIPTION: 〈Description〉 //必要的文字描述

OPTIONAL:

DESIGN_PATTERN: 〈Design Pattern Name〉 //使用何种设计模式

DIAGRAM_PRESENTATION: //扩展点的图形描述

〈Diagram Name, File Name, Directory Name〉

}

(3) 界面扩展点

界面扩展点主要针对不同用户对同一操作界面有着不同风格要求的情况,可以使用 MVC(model-view-controller)[9]模式来处理系统界面的扩展.MVC 模式将业务逻辑和用户界面分离开来,使得用户界面做到“即插即用”,而且对用户界面的修改不会影响到业务逻辑,使得系统易于演化和维护.应用系统开发者首先设计出符合个人喜好和系统要求的界面模板,然后系统可以采用参数化的方式实现对界面模板的调用.例如,构件 InvoiceManager 打印发票的操作 PrintInvoice()在执行前并不了解发票的打印格式,在执行过程中调用参数指定的发票界面模板,按照模板的格式进行打印.

下面对界面扩展点的描述方法为支持参数化调用提供了必要的信息,记录了哪一个构件接口操作将调用界面模板、操作的参数列表,以及每一个参数可选的参数值,由参数值组合确定将调用哪一个界面模板.

HOT_SPOT Hot Spot Name

{

TYPE: 〈Interface〉 //界面扩展点

COMPONENT_INTERFACE_OPERATION: //调用界面模板的构件操作

〈Component Name, Interface Name, Operation Name〉

〈Parameter 1, Parameter 2,…, Parameter *n*〉 //调用界面模板需要的参数

INTERFACE_TEMPLATE_LIST: //已有的模板列表

〈Template Name 1, Parameter 1, Parameter 2,…, Parameter *n*〉

〈Template Name 2, Parameter 1, Parameter 2,…, Parameter *n*〉

…

DESCRIPTION: 〈Description〉 //必要的文字描述

OPTIONAL:

DIAGRAM_PRESENTATION: //扩展点的图形描述

〈Diagram Name, File Name, Directory Name〉

}

(4) 过程扩展点

过程扩展点处理的变化性主要体现在需要为应用系统开发新的构件以实现框架没有提供的功能,而且要将新构件集成到框架中,形成新的业务过程.

框架元模型中的构件从承担责任的角度分为功能构件和连接构件,其中业务功能的扩展体现在功能构件中,而业务过程的扩展体现在连接构件中,同时连接构件负责功能构件之间的交互.连接构件的实现有多种途径,例如构件组装工具、脚本和过程引擎等.本文以过程引擎为例,给出作为构件框架连接件的过程引擎模型.应用系统开发者首先定义业务流程的规则(框架可以提供业务流程定义工具),过程引擎根据已定义的业务规则执行业务流程.

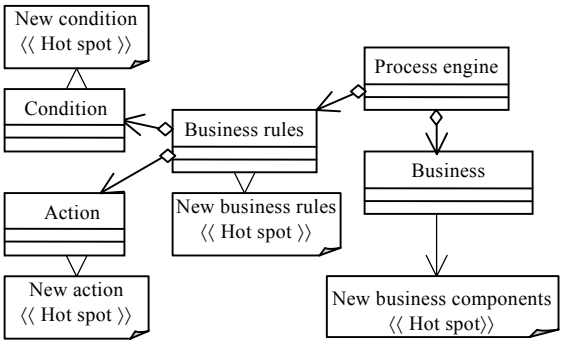

Fig.3 Process engine model
图 3 过程引擎模型

图 3 中,过程引擎模型的组成元素有:过程引擎、业务规则、条件、动作和功能构件.其中过程引擎负责根据已定义的业务规则调用功能构件;业务规则表达了业务流程中的业务逻辑.在一个应用系统中存在若干个完整的业务流程,每一个业务流程需要不同构件提供的功能协作完成,业务规则就是用来表达业务构件的激发顺序和规则.业务规则由一组条件和动作来表达,每一个条件都和在该条件下执行的一组动作相关联,同时动作由相应的功能构件完成.概念上的动作与实际完成操作的构件相分离,这样,应用开发者通过替换完成某动作的功能构件就可以实现业务逻辑的改变,而无须直接修改代码.业务规则是与实现时选用的构件规范和技术无关的,只保存构件接口和方法的激发条件和顺序,因此框架使用者可以很容易地根据应用需求扩展控制流程.此外,框架中实现的过程引擎是与领域应用无关的,该过程引擎可以被其他应用框架所复用.

对过程扩展点的描述中需要说明框架中有哪些连接构件组成、连接构件与功能构件的交互接口是怎样的,以及各个连接构件有什么功能等.

HOT_SPOT Hot Spot Name
{
TYPE: ⟨Process⟩ //过程扩展点
CONNECT_COMPONENT_1: //连接构件列表
⟨Component Name, Interface Name, Operation Name⟩ //对外提供的接口和操作
⟨Parameter 1, Parameter 2,…, Parameter *n*⟩ //操作的参数列表
⟨Description⟩ //文字描述连接构件的功能
CONNECT_COMPONENT_2:
⟨Component Name, Interface Name, Operation Name⟩
⟨Parameter 1, Parameter 2,…, Parameter *n*⟩
…
DESCRIPTION: ⟨Description⟩
OPTIONAL:
DIAGRAM_PRESENTATION: //扩展点的图形描述
⟨Diagram Name, File Name, Directory Name⟩
}

1.2.2 设计模式

设计模式是针对反复出现的设计问题给出的、经过实践检验的、可复用的解决方案[7].通常,设计模式与领域无关,可用于不同的应用系统和不同的领域.它记录了现有的、专家的设计知识,可以用来为具体的设计问题提供适当的解决办法.框架是一种可复用的、可适配的软件,它要有灵活的结构以易于扩展.设计模式提供了可以用来解决这种问题的解决方案,因此,框架扩展点可以利用设计模式的思想来实现,框架中通常可能包含若干个设计模式.

设计模式和框架之间的区别在于[8]:

(1) 设计模式比框架更抽象一些.框架被表示为用某种编程语言实现了的代码,而模式不是这样.只有那些用来说明模式的例子才表现为代码的形式.模式还包括许多其他描述信息,如用途、可用性、结果等,这些信息在框架中常常没有描述.

(2) 设计模式是比框架小一些的结构.框架可以包含多个设计模式.

(3) 框架比设计模式要专用一些.框架总是与某个特定的应用领域相关,而设计模式常常是通用的,可以应

用于多种应用领域.

1.2.3 构 件

应用框架由一组协作的构件组成,根据构件在框架中的作用可以分为功能构件和连接构件.功能构件为应用系统提供业务功能,同时功能构件支持业务功能的扩展和与功能相关的数据扩展以及界面的扩展.连接构件负责业务规则和业务逻辑的实现,同时支持业务过程的扩展和灵活定制.

1.2.4 控制流

框架是一种运行时体系结构,这种结构被称为“反向控制(inversion of control)”[9],它使得应用程序过程能被框架的集成机制(即连接构件)所控制.当事件发生时,首先由框架的集成机制对事件作出反应,然后通知对该事件感兴趣的构件,并执行特定的代码过程.反向控制允许框架(而不是应用程序)决定激活哪一个方法来响应外部事件(如最终用户发出的 Windows 消息或到达通信端口的数据包).在传统的基于构件的开发过程中,开发者通过一个主程序调用构件提供的功能,应用系统开发者决定何时调用构件,并负责维护整个系统的结构.而对于框架而言,应用系统开发者需要决定向框架中插入哪些构件并可能开发一些新的构件,因此认为框架决定了系统的整体结构和控制流,而应用系统开发者编写的内容被框架调用.

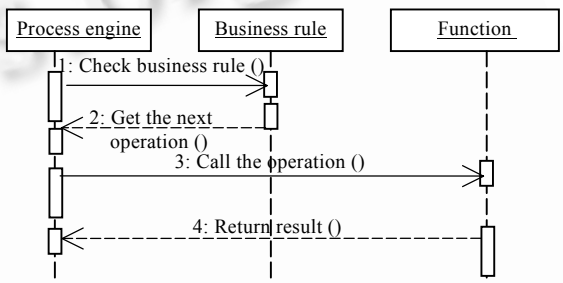

Fig.4 Framework control on running time

图 4 运行时刻的框架控制

上文介绍的过程引擎确定了框架的整体结构和控制流,图 4 表达了运行时刻的过程引擎、业务规则和业务构件之间的控制流程.在功能构件完成某一操作以后,程序控制返回到过程引擎,此刻,过程引擎需要知道下一个操作由哪一个构件完成,于是,过程引擎发起检查业务规则的操作,通过查找已定义的业务规则选择下一个操作:“2:得到下一个操作()”的返回结果包括下一个操作、包含该操作的构件以及操作所需的参数.由过程引擎激发相应功能构件的相应操作,在操作完成之后,程序控制再次返回过程引擎.

2 实例研究

本节结合电信综合营业系统框架的实践研究,给出框架元模型的一个实例.电信综合营业系统框架的目标在于改变目前电信运营商多种业务独立建设、分散管理的体制,为电信运营商的多业务实施提供“一单清、一台清”,建立一致的体系结构和统一的用户界面,使得新业务的扩充和演化更加容易,并实现业务工作流程的自动化定制和管理.

电信综合营业系统框架采用了基于构件的框架技术,如图 5 所示的电信综合营业系统框架中各个组成成分分别实例化了框架元模型中的内部元素.框架中包含两种角色的构件:功能构件和连接构件,其中框架的数据、功能和界面扩展点在功能构件中被实现;连接构件是灵活定制业务流程的过程扩展点的实现机制.由于功能构件只提供了单一的业务功能,并没有实现完整的业务流程,连接构件负责根据预先定义的业务规则,将业务构件提供的功能集成为具有逻辑意义的业务流程,作为可复用的控制流程.

应用系统开发者可以复用该框架通过扩展来定制一个符合特殊需求的电信综合营业系统,还可以方便地为现有的系统添加新的业务流程、数据结构、界面和算法.框架提供如下几种方法来完成这些任务:

(1) 直接使用框架所提供的默认实现来创建一个应用系统.该方法不需要加入任何新的算法或数据类型.这是最简单的方法.

(2) 在框架已经创建的业务流程模板和界面模板的基础上进行适应性修改,创建新的应用系统.

(3) 利用系统提供的可视化界面设计和业务规则定义的功能,设计新的界面模板和业务流程.

(4) 重新实现框架中的功能构件接口,改变业务功能的运算逻辑.

(5) 用一个新功能构件实现相应的接口,替代原来的功能构件.

(6) 实现新的功能构件,定义新的业务规则,形成新的业务流程.

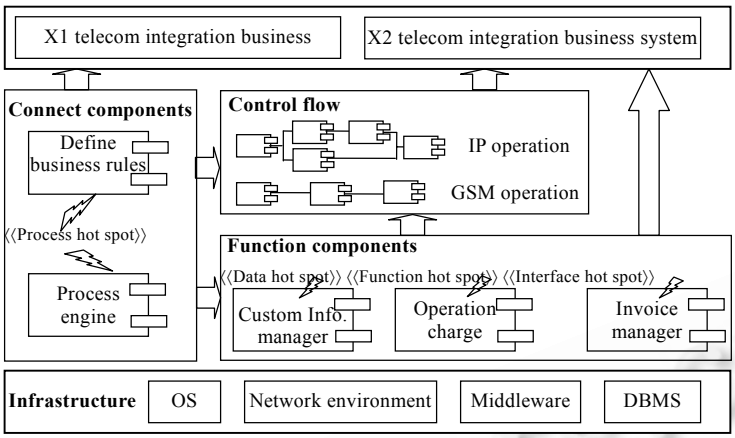

Fig.5 Telecom integration business system framework
图 5 电信综合营业系统框架

3 结束语

本文以对已有成熟框架的研究和若干特定领域框架的开发为基础,提出了基于构件技术的框架元模型,描述了框架内部组成元素和边界元素,并进一步阐述了各元素与框架之间的关系,特别是给出了应用框架的重要组成成分——扩展点的表示方法和实现机制,对应用框架的设计和实现具有借鉴和指导意义.但是对于框架的形式化描述和扩展点的实现机制还有待于进一步加强和完善.

References:

[1] Johnson RE. Frameworks = (Components + Patterns). Communications of the ACM, 1997,40(10):39~42.
[2] Rumbaugh J, Jacobon I, Booch G. The UML Reference Manual. New York: Addison-Wesley, 1999.
[3] Andert G. Object frameworks in the Taligent OS. In: Proc. of the Compcon'94. Los Alamitos: IEEE CS Press, 1994. 112~121.
[4] Architecture of the San Francisco frameworks. 1998. http://www.research.ibm.com
[5] Cotter S, Potel M. Inside Taligent Technology. New York: Addison-Wesley, 1995.
[6] Carnegie Mellon University. The Software Product Line Practice Version 4.1, 1997. http://www.sei.cmu.edu
[7] Braga RTV, Masiero PC. An approach for frameworks construction and instantiation using pattern languages. In: Proc. of the Int'l. Conf. on Computer Science, Software Engineering, Information Technology, e-Business, and Applications. ACIS, Foz do Iguacu, 2002. 305~310.
[8] Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. New York: Addison-Wesley, 1995. 175~184; 315~324.
[9] Fayad M, Schmidt DC. Object-Oriented application frameworks. Communications of the ACM, 1997,40(10):43~54.