基于模式的业务构件代码生成方法

冯锦丹,战德臣,聂兰顺,徐晓飞
(哈尔滨工业大学计算机科学与技术学院,黑龙江哈尔滨 150001)

摘 要: 在企业管理软件中业务构件可以通过参数化和配置技术被复用以适应业务需求的变化,这种构件粒度大、可变参数多、结构复杂难于开发.为了提高业务构件的开发效率,确保开发正确性,本文提出一种面向大粒度构件,基于模式的构件代码生成方法.从已有应用系统中抽取业务构件的典型软件模式,定义一种支持多种编程语言的模式描述语言,用其构造基于模式的非实例化构件,并采用模型驱动的思想,将对应模式下富含业务语义的构件实现模型转换为目标程序,以实现业务构件的自动生成.在此方法指导下,开发一套可视化建模与代码生成工具组,并利用此工具自动生成了运行在 J2EE 平台上的采购管理系统中部分业务构件,应用结果验证了本方法的有效性和实用性.

关键词: 代码生成; 模式; 模型驱动的架构; 业务构件; 业务构件模型
中图分类号: TP3 **文献标识码**: A **文章编号**: 0372-2112 (2008) 12A-019-06

Pattern Based Code Generation Method for the Business Component

FENG Jin-dan, ZHAN De-chen, NIE Lan-shun, XU Xiao-fei
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin, Heilongjiang 150001, China)

Abstract: The business component is reused to adapt requirement changes by parametrization and configuration in large scale software for enterprise, but it is developed difficulty because of the large granularity, various parameters and complexity of the structure. This paper presents a pattern based code generation method for the large-grained component to improve the efficiency and correctness of development. A specific formal pattern description language is defined to describe the typical software patterns abstracted from the existing components programmed with multiple programming languages. A code generation algorithm is proposed to transform component models for business semantics into target program based on these patterns. The visualization modeling tool and code generator have been developed to generate automatically business components based on J2EE platform. Result of a case purchase system shows effectiveness of this method.

Key words: code generation; pattern; MDA; business component; business component model

1 引言

为了增强市场竞争力,企业的业务需求总是不断变化的,这就要求管理软件必须具有柔性以适应持续变化的业务需求[1].随着面向对象技术的成熟,构件化的软件体系结构和基于构件的企业管理软件开发方法也逐渐成为解决此问题的首选方案.构件的复用与重构技术可以提升构件的适应能力,其中参数化与动态配置技术可用于开发具有适应性的构件,使其在应用过程中满足持续变化的需求[2,3].这种适应性强的构件程序表现出结构与业务功能复杂、规模大、可变参数多等特征.这些特征交织在一起导致了其开发效率低、周期长、正确性难以保证等问题.

模型驱动的软件开发方法提供了一种提高软件开发效率和质量的有效途径,其以模型为中心,通过建立正确的业务模型,利用代码生成器将模型转换为软件代码.模型驱动的软件开发只有在代码生成技术相对成熟的基础上才能真正得以实现.早期代码生成的研究大都集中于编译领域,之后被引入基于高级语言的软件开发领域,推进了软件开发的工业化规模化[4].随着模型驱动的软件体系结构(Model Driven Architecture, MDA)的发展,代码生成技术被广泛应用于企业应用软件(Enterprise Software Application, ESA)的开发,支持从平台相关模型到可执行代码的自动转换[5,6].基于模式的代码生成方法引起了 ESA 软件研究人员的广泛关注,文献[7]将软件开发划分为基于概念模式规约的问题域,基于设计模式的设计域和实现概念模式的解空间,采用设计模式指导概念模型到服务程序的生成.文献[8]建立基于

收稿日期:2008-10-22;修回日期:2008-12-20
基金项目:国家自然科学基金(No. 60773064);国家 863 高技术研究发展计划(No. 2006AA01Z167, No. 2006AA04Z165, No. 2006AA04Z150, No. 2007AA01Z128, No. 2008AA04Z101, No. 2008GG10004010)

XML 模式的柔性代码生成器为分布式系统提供了分布式的代码生成基础. 也有一些研究基于 XSLT 和 Velocity 模板引擎实现了业务模型到程序的转换[9].

这些研究大都侧重于细粒度程序,如函数、方法、语言类等的生成,缺少面向业务语义的大粒度构件生成方法的研究. 在现实中大多数 ESA 构件都具有相似的结构和不同的业务语义. 受到 MDA 思想的启发,提出一种以业务构件模型为中心,基于模式的构件代码生成方法. 将构件的共性抽象为模式,基于模式定义非实例化的构件,而将那些面向业务语义的个性内容描述为业务构件模型. 基于模式的非实例化构件与模型之间存在型-值关系,前者提供了基于语法的构件型描述,后者提供了面向业务语义的构件值描述. 通过代码生成工具将这两部分内容融合起来,可转换得到一个具体的业务构件.

2 业务构件的模式定义

业务构件(Business Component,BC)是企业生产和经营过程中所处理的业务对象(如业务单据)的软构件实现结果,包含对象的数据信息及相关的业务活动等. 业务构件是一种具有完整业务语义的大粒度构件,称其为大粒度构件是相对于小粒度构件而言的,例如一般概念上的 EJB、CORBA、DOC 等分布式构件可看作是小粒度构件. 网络环境下的业务构件大都采用分层的结构,每个层次分别由若干小粒度构件组装而成,且在各自的环境(客户端或服务器端)中运行,相互衔接为用户提供完整的服务.

总结多年的企业管理软件开发经历,我们发现企业管理软件总是遵循一些特定的模式. 通过对构件库中已有构件的分析,可发现业务构件中也同样存在各种各样的软件模式(Software Pattern,SP),可从中选择一种作为代码生成目标的标准模式. 如图 1 所示,从总体上看软件模式可划分为面向使用者的用户模式(User Pattern,UP)与面向软件的实现模式(Realization Pattern,RP)两类. 用户模式体现为软件与客户端用户的交互风格,实现模式是从软件开发角度表现为一种实现方案或运行机制. 前者是后者的外在表现,后者是前者的技术支撑. 在软件外观层次,用户模式分别从用户界面的结构和用户操作的流程两个方面描述软件的外特性. 在软件实现层次,实现模式依据表示层、业务逻辑层、数据层和其间的关系模式来描述构件的实现方式. 将那些用于实现父结点的模式被称作子模式. 可见,业务构件模式是由一系列子模式(用户子模式和实现子模式)构成的集成体,父模式与子模式之间存在包含关系,两个实现子模式之间需要根据接口模式进行关联.

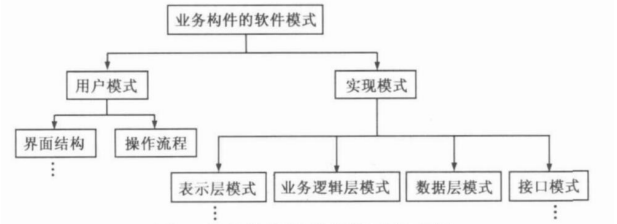

图1 业务构件的软件模式构成树

业务构件的软件模式可以表示为 $BC\text{-}P = (PID, Sum, Con, UP, RP)$,其中 PID 表示软构件模式的唯一标识; Sum 描述模式的功能或作用; Con 描述了此模式的适用条件或环境; $UP = (UIA, OPF)$, $UIA = (BlockSet, DataSet, OperSet, IN)$, $BlockSet = \{B \mid B$ 表示用户界面中的区域$\}$, $DataSet = \{data \mid data$ 表示用户界面中数据项$\}$, $OperSet = \{oper \mid oper$ 表示操作项$\}$, $IN = \{in \mid in = (B, \{data\}, \{oper\})\}$ 表示数据集与操作集在不同区域里的隶属关系, OPF 表示用户界面上的操作流程; $RP = (PL\text{-}P, LL\text{-}P, DBL\text{-}P, 3L\text{-}P)$,依次由表示层模式、业务逻辑层模式、数据层模式和层间关系模式共同构成. 定义 RP 的主要目标是从代码实现角度指导业务构件的开发,为此提出一种源代码级的实现模式描述语言(Realization Pattern Description Language,RPDL)以形式化的描述 RP,其可定义为:

〈RPDL〉::=〈静态代码〉|〈动态代码〉
〈动态代码到静态代码的转换规则〉
〈静态代码〉::=〈目标程序语言所描述的内容〉
〈动态代码〉::=〈动态变量开始符〉
〈待实例化的动态变量〉
〈动态变量结束符〉
〈待实例化的动态变量〉::=
〈用程序语言 A 定义的变量〉
〈动态代码到静态代码的转换规则〉::=
〈转换规则的开始符〉
〈基于 A 的转换规则内容〉
〈转换规则的结束符〉

其中静态代码是指目标程序中的代码,动态代码则是需要依据用户指定数据源而变化的部分,其最终将被转换为静态代码. 转换规则的开始符与结束符限定了转换指令的起止位置,采用高级语言 A 描述的转换规则负责定义动态变量、获取动态变量的赋值数据源并构造转换框架. 动态代码与其转换规则都不会出现在最终的目标程序中. 依据不同的实现平台,程序语言 A 可是 JAVA 或 JSP 或 C++ 等. 使用 RPDL 描述的实现模式当只含静态代码时则无需转换,直接成为目标代码. RPDL 具有一定的通用性,既支持依据用户的功能需求定义各自的实现模式,又可定义多种高级程序语言的实现模式. 鉴于篇幅限制,本文仅给出表示层中数据项的实现子模式描述,如下列各项所示.

```
(1)〈& &                    //表示转换规则的开始符号
(2) …                       //定义待实例化的动态变量
(3) …                       //获取用户指定数据集内容
(4) Iterator it = getDtateSet();
(5) while(it.hasNext()){ …//获取各动态变量值
(6) &&〉                    //表示转换规则的结束符号
(7)〈td〉〈#lableName#〉:〈input
   name="〈#controlID#〉"type="〈#dataType#〉"
   value="〈%=〈#controlID#〉%〉"size="〈#dataSize#〉"
   〉〈/td〉                 //基于JSP的静态与动态代码混合体
(8)〈&&} &&〉               //对应转换规则中的while{符号
```

上述各项是关于JSP界面以单元格形式显示多个数据项的实现模式的代码描述,这个片段从(2)到(5)行是采用JAVA语言定义的转换规则内容,其主要功能是对(7)行中由〈#和#〉限定的动态代码进行初始赋值.行(5)是此模式的核心功能,表示第(7)行中代码会依据数据源基数而多次迭代出现,每次出现时动态代码将被实例化为不同静态代码值.

3　基于模式的业务构件模型

业务构件模型(Business Component Model,BCM)是业务语义集成体基于特定 *BC-P* 的构件实现模型.依据业务构件的软件模式树结构,给出其详细语法描述.一个BCM可由唯一标识、选定的构件模式、基于分层结构的核心模型、用于支持可变性的配置模型以及外部接口模型共同组成,其可定义为:

〈业务构件模型〉::=〈模型标识〉〈构件模式〉
〈表示层模型〉〈业务逻辑层模型〉〈数据模型〉
〈配置模型〉〈扩展服务〉〈各部分间的关系集〉

3.1　表示层模型

表示层是业务构件与用户直接交互的接口.一个业务构件总是由若干个相互关联的用户界面构成,为用户提供提交请求与反馈处理结果等服务.各用户界面之间存在相互调用或一般性的链接关系.表示层模型的可定义为:

〈表示层模型〉::=〈界面集〉〈界面关系-集〉
〈界面〉::=〈界面名称〉〈分区集〉〈分区关联-集〉
〈界面关系〉::=〈调用式关系〉|〈链接式关系〉
…

依据用户界面内部的语义关联性,可将用户界面划分为若干个区域分别予以定义.所谓分区是指将用户界面的各组成部分按其作用和功能划分成不同区域,每个分区含有若干数据集、操作集和状态集.状态集可控制数据或操作的可见可用等可变属性.例如(1)索引区:为用户提供检索数据的条件输入接口,且依据检索条件显示与定位记录值;(2)主表区:装载业务对象的核心数据集和相应的操作集;(3)明细区:装载业务对象的辅助数据集和操作集.下面以主表区为例说明其构成关系:

〈主表区〉::=〈主表标识〉〈布局属性〉〈状态〉
〈数据项集〉〈编辑操作集〉〈操作模式集〉
〈布局属性〉::=〈主表标题〉〈数据显示行列数〉
〈数据项〉::=〈属性〉〈显示标签〉〈编辑控件集〉
〈数据项属性〉::=〈可见性〉〈可写性〉〈位序〉…
〈编辑操作〉::=〈增加〉〈修改〉〈删除〉[其他]
〈其他操作〉::=〈显示标签〉〈扩展函数〉〈属性〉
〈编辑操作的属性〉::=〈可见性〉〈可用性〉
…

同一界面内部的分区与不同界面之间的分区之间都存在一定的关联,只有明确描述了这些关联才能确保界面结构的合理与操作流程的畅通.可采用两两关联的原则建立分区之间的关联,由于数据和操作是构成分区的核心要素,因此建议使用比较运算符逐一映射两个分区内部的数据,描述并发执行性以建立操作之间的联系,主表区与其附带的明细区之间可能存在父子表与索引表等关联关系,其定义如下:

〈分区关联〉::=〈父子表〉|〈索引表〉|〈关联表〉…
〈父子表〉::=〈关联标识〉〈主表标识〉
〈子表标识〉〈父子表关联规则〉
〈父子表关联规则〉::=
〈数据集的关联规则〉|〈操作集的关联规则〉
…

3.2　业务逻辑层模型

业务逻辑层是响应客户端的用户需求并进行处理的核心逻辑程序.基于J2EE平台上业务逻辑层的实现模式可表述为:它由若干包和类组成,其中常规操作的方法实现可由代码生成器依据模式定义而自动生成获取,存储在默认类文件中;那些根据用户需求在扩展机制的约束下由程序员手工开发的方法则存储于扩展类文件中;核心控制类负责对其进行服务编排与方法调用.这三者相分离既有助于提高模型的复用度,又有助于缩小定制扩展时修改程序的范围.

〈业务逻辑层模型〉::=〈包〉
〈包〉::=〈包名〉〈类-集〉〈类间关系集〉
〈类〉::=〈控制类〉|〈默认类〉|〈用户扩展的类〉
〈控制类〉::=〈类名〉〈成员变量〉〈映射〉〈构造方法〉
〈映射〉::=〈函数选择符〉
(〈默认类中的成员方法〉|
〈用户扩展类中的成员方法〉)
〈默认类〉::=〈类名〉〈成员变量〉
〈自动生成的成员方法〉〈构造方法〉
〈自动生成的成员方法〉::=〈索引区的方法〉
〈主表区的方法〉〈明细区的方法〉
〈索引区的方法〉::=〈查询方法〉〈选中方法〉
〈主表区的方法〉::=〈查询方法〉〈增加方法〉
〈修改方法〉〈删除方法〉
〈用户扩展类〉::=〈类名〉〈成员变量〉
〈用户扩展的成员方法〉〈构造方法〉

…

表示层与业务逻辑层之间的映射包含两个部分:(1)表示层的数据项与逻辑层中变量的对应关系;(2)表示层的操作控件被事件触发后,可通过逻辑程序中函数选择符调用与其相对应的方法.

3.3 数据层模型

数据层模型是在关系数据模式下描述业务构件的数据库外模式模型,描述了构件业务逻辑层处理的数据视图与实际数据库中的数据视图之间的映射关系.业务逻辑层的逻辑数据表与数据库中的数据表除了在名称上映射之外,其内部的数据字段也要进行映射,对于任意的逻辑字段都必须至少存在一个数据库字段与其对应.其定义如下:

〈数据层模型〉::=〈逻辑数据表集〉
〈DB 数据表集〉
〈逻辑数据表与 DB 数据表的映射关系〉
〈逻辑数据表〉::=〈逻辑数据表名〉
〈逻辑字段集〉
〈DB 数据表〉::=〈DB 数据表名〉〈DB 字段集〉
〈逻辑数据表与 DB 数据表的映射关系〉::=
〈表表映射〉〈字段映射〉
〈表表映射〉::=〈逻辑数据表名〉〈DB 数据表名〉
〈字段映射〉::=〈逻辑数据表.逻辑字段名〉
〈DB 数据表.DB 字段名〉

3.4 构件配置模型与接口模型

在软件层面采用角色代表岗位职能权限,用户表示终端的登陆者.面向业务语义的业务构件具有可变性,建模人员通过建立配置模型,组合与控制构件内部的可变属性,依据不同的权限约束与用户关注的内容,为其提供权限相当的可执行功能和适当的数据结果.配置模型的存在使得同一个业务构件能够在更多的情景中被复用,提高了构件的可用性.接口模型描述了业务构件模型所能提供的服务声明,需求的服务声明,及服务所需的参数类型声明等信息.

4 基于模式的业务构件生成方法

业务构件的代码生成过程就是将基于某种软件模式的业务构件模型转换为业务构件代码的过程.其基本思想如图2所示,当一个业务构件模型是若干业务构件模型模式的实例的集成体,这些模型的模式可以通过一系列转换模式等价地映射为一组代码模式时,就能够得到一个业务构件程序,这个构件是由上述代码模式的实现结果组装而成的.其中,实现模型到代码转换的关键要素是转换模式,是由一组衔接模型模式与代码模式的等价映射规则组成.模型的模式,代码的模式与转换模式必须成组定义与应用才能保证模型到代码转换过程的正确性.可以假设,在转化模式一定能够支持模型正确转换的前提下,只要依据模型模式建模就一定能获取相符的业务构件程序.

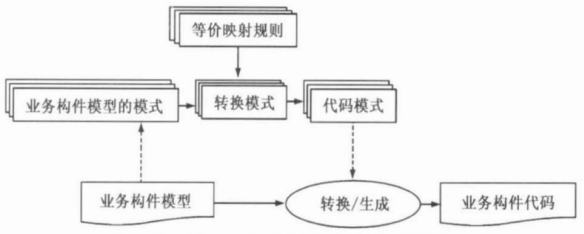

图2 基于模式的业务构件代码生成原理

业务构件的软件模式理论充分映射了这种模型到代码的生成思想.具体而言,业务构件的模型是基于特定源代码模式的模型,其提供了构件非共性内容的实例.而基于模式的非实例化构件提供了代码模式和转换模式,其中静态代码与动态代码的结合体体现了一定的代码模式,动态代码到静态代码的转换规则是等价映射规则的具体实现.

由此给出一种基于模式的业务构件代码生成算法.输入:基于某种模式的业务构件模型 X;输出:业务构件代码.其内容如下:(1)解析业务构件模型的模式标识,在模式库中检索并获取相应模式的非实例化构件;(2)扫描非实例化构件内容,当找到一个动态代码时,检索 X 中与其对应的数据进行代码转换;(3)重复执行(1)直到非实例化构件中不存在动态代码为止,转到(4);(4)打印最终的构件代码.

5 支持工具的功能与实现

本文提出大粒度业务构件软件模式和模型的终极目标是实现构件的模式化工业化开发.在上述思想的指导下,研发了一套基于 J2EE 平台的业务构件代码生成工具,并将其应用于采购管理系统的开发过程.本章简要给出代码生成器的实现与应用情况.

业务构件代码生成过程可分为两个阶段:第一阶段,建模者在已有的业务构件模式库中选择一种特定的软件模式作为建模基础,根据此模式需求采用填空

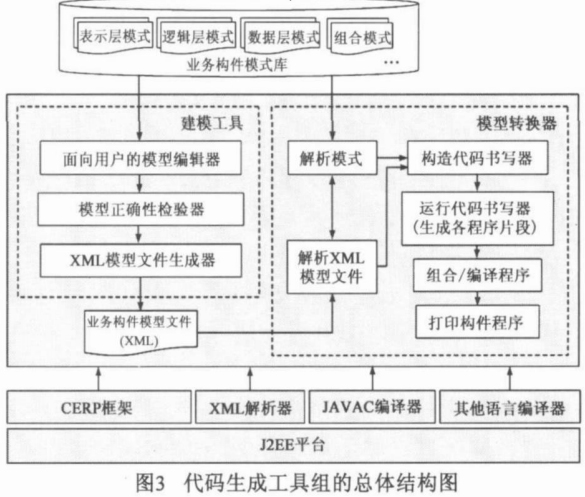

图3 代码生成工具组的总体结构图

式方法建立一个业务构件模型. 模型的各组成部分均符合特定的子模式约束. 工具组将面向用户的模型信息转换为面向软件的、采用 XML 语言描述的可执行模型文件. 第二阶段,将基于 XML 的构件模型文件作为解析对象,将其与匹配的非实例化构件融合,打印出目标构件代码. 因此,工具组由建模工具与模型/代码转换器两个核心部分构成,其体系结构如图 3 所示.

代码生成工具组采用三层架构的实现方式,采用 JSP 语言开发用户界面,业务逻辑层由 JavaBean 实现,数据库采用 SQLserver 予以支持,工具组的运行界面如图 4 所示.

图4　代码生成工具组的界面

模型编辑器可依据不同的软件模式提供不同类型模型要素的定义编辑区,通过模型正确性验证后生成以 XML 形式存在的模型文件. 在模型转换成代码过程中,依据模型文件的各个构成部分所匹配的模式类型分别构造临时代码书写器. 如以 J2EE 平台上实现编制订单功能的构件生成为例,书写器将分别生成 JSP、JAVA 和数据库文件,在多语言编译器的支持下进一步将 JAVA 源代码编译成 CLASS 文件,最后打包并输出业务构件程序.

应用代码生成工具组进行采购管理系统开发的过程中,统计 50 个业务构件的手工开发与自动生成的耗时情况,可发现每个大粒度构件的平均开发周期由 2 天缩短为 0.7 天,开发效率提高了约 65 %,并且构件质量与规范化程度也得到了较好的保证.

6　结论

本文提出了一种基于模式的业务构件代码自动生成方法,给出了业务构件的软件模式树,面向源代码级别的实现模式描述语言 RPDL 和描述方法. 在特定软件模式的基础上详细定义了业务构件实现模型的各个构成部分. 并进一步提出了基于转换模式的代码生成原理与算法,此方法具有一定通用性,可应用于多种实现平台的 ESA 软件开发. 最后,通过支持工具与实验说明了此方法对提高大粒度构件开发效率与质量具有很好的支持作用. 自动生成的业务构件具有较多的可变属性,必须经过权限和个性化相关配置后才能部署到应用系统. 我们进一步的工作包括完善构件配置模型,开发配置与部署工具以支持构件全生命周期的自动化实现.

参考文献:

[1] Kulkarni Vinay, Reddy Sreedhar. A model-driven architectural framework for integration-capable enterprise application product lines[A]. Model Driven Architecture-Foundations and Applications: Second European Conference[C]. Bilbao, Spain: Springer Publishing Company, 2006. 1 - 12.

[2] Cosmin E Oancea, PStephen m Watt. Parametric polymorphism for software component architectures[A]. Proceedings of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications[C]. San Diego, USA: ACM Press, 2005. 147 - 166.

[3] Ali Arsanjani, Hussein Zedan, James Alpigini. Externalizing component manners to achieve greater maintainability through a highly re-configurable architectural style[A]. Proceedings of the International Conference on Software Maintenance[C]. Montreal, Quebec, Canada: IEEE Computer Society, 2002. 628 - 637.

[4] Ngolah C F, Yingxu Wang. Exploring java code generation based on formal specifications in RTPA[A]. Proceedings of the 2004 Canadian Conference on Electrical and Computer Engineering (CCECE '04)[C]. Niagara Falls, Canada: IEEE CS Press, 2004. 1533 - 1536.

[5] Neema S, Kalmar Z, Feng Shi, Vizhanyo A, Karsai G. A visually-specified code generator for simulink/stateflow[A]. IEEE Symposium on Visual Languages and Human-Centric Computing[C]. Washington DC, USA: IEEE Computer Society, 2005. 275 - 277.

[6] David S, Frankel. Model Driven Architecture: Appplying MDA to Enterprise Computing[M]. New York: John Wiley & Sons, Inc, 2002.

[7] Vicente Pelechano, Oscar Pastor, Emilio Insfrán. Automated code generation of dynamic specializations: an approach based on design patterns and formal techniques[J]. Data & Knowledge Engineering, 2002, 40(3): 315 - 353.

[8] Madhusudhan Govindaraju. XML schemas based flexible distributed code generation framework[A]. Proceedings of the IEEE International Conference on Web Services (ICWS 2007)[C]. Saltlake City, UT, USA: IEEE Computer Society, 2007. 1212 - 1213.

[9] T Sturm, J von Voss, M Boger. Generating code from UML with velocity templates[J]. The Unified Modeling Language. 2002, 2460: 379 - 386.

作者简介:

冯锦丹 女, 1980 年生于辽宁锦州. 哈尔滨工业大学计算机科学与技术学院博士生, 研究方向为软件复用、软构件与代码生成.
E-mail: fjd1127@163.com

战德臣 男, 博士, 教授, 博士生导师, CCF 高级会员. 主要研究领域为软件复用与软件体系结构、模型驱动架构、现代企业管理、数据与知识工程.
E-mail: dechen@hit.edu.cn

聂兰顺 男, 博士, 讲师, 主要研究领域为软件体系结构、软件复用、大型管理软件.

徐晓飞 男, 院长, 博士, 教授, 博士生导师, CCF 高级会员. 主要研究领域为管理与决策信息系统、数据库、企业资源计划与供应链管理技术、服务科学.

(上接第 38 页)

[4] LIU G J, WANG J W, et al. Data hiding in neural network predictioni errors[A], Lecture Notes in Computer Scence (Advances in Neural Networks-ISNN 2006)[C], Heidelbergpp: Springer Berlin, 2006. 273 - 278.

[5] PARK Y-R, KANG H-H, et al. A steganographic scheme in digital images using information of neighboring pixels[A]. Lecture Notes in Computer Scence (Advances in Natural Computation)[C]. Heidelbergpp: Springer Berlin, 2005. 962 - 967.

[6] PARK Y-R, KANG H-H, et al. An image steganography using pixel characteristics[A]. Lecture Notes in Artificial Intelligence (Computational Intelligence and Security)[C]. Heidelbergpp: Springer Berlin, 2005. 581 - 588.

[7] ZHANG X P, WANG S Z, Steganography using multiple-base notational system and human vision sensitivity[J]. IEEE Signal Processing Letters, 2005, 12(1): 67 - 70.

[8] YANG C Y, LIN J C, Image hiding by base-oriented algorithm [J]. Optical Engeering, 2006, 45(11): 117001 - 1 - 117001 - 10.

[9] WESTFELD A. Space filling curves in steganalysis[A]. Proccedings of SPIE: Security, Steganography and Watermarking for Multimedia[C]. San Jose, California: SPIE, 2005. 28 - 37.

[10] VOLOSHYNOVSHIY S, PEREIRA S, et al. Attacking modelling: towards a second generation watermarking benckmark [J]. Signal Processing, 2001, 81(6): 1177 - 1214.

[11] WANG Z, BOVIK A C, et al. Image quality assessment: From error visibility to structural similarity[J]. IEEE Transaction on Image Processing, 2004, 13(4): 600 - 612.