

.NET 下基于 PowerDesigner 和 CodeSmith 的 软件自动化开发技术

朱晓辉 王杰华 石振国 陈苏蓉

(南通大学计算机科学与技术学院 南通 226019)

摘 要 针对当前软件开发效率低下、难以快速适应需求变化等问题,提出一种新的软件自动化开发技术。在一定的约束条件下通过 PowerDesigner 完成数据库概念模型的设计和物理模型转换,在 CodeSmith 代码引擎的支持下,通过自定义的模板来实现软件代码的自动生成。有效解决了基于数据库的 MIS 系统的开发效率和软件质量问题。通过在实际项目中的应用表明,该技术能有效提升软件开发效率,降低开发成本。

关键词 PowerDesigner, CodeSmith, 代码模板, 软件自动化开发

中图分类号 TP311.1 **文献标识码** A

Software Automation Development Technology by PowerDesigner and CodeSmith in .NET Framework

ZHU Xiao-hui WANG Jie-hua SHI Zhen-guo CHEN Su-rong

(College of Computer Science and Technology, Nantong University, Nantong 226019, China)

Abstract Concerning the inefficiency of software development and the difficulty in quickly adapting the changes of requirement, this paper described a new technology for software automation development. It solved the problem of software automation development by designing conceptual model of database and converting to physical model by PowerDesigner with some constraints and creating code automatically by the support of CodeSmith and customer templates. It solved the problem of development efficiency and software quality for MIS which is based on database. By applying in fact project, it showed that the solution could improve the efficiency of software development and reduce the costs.

Keywords PowerDesigner, CodeSmith, Code templet, Software automation development

随着计算机的普及和信息技术的发展,越来越多的企事业单位加快了信息化建设的步伐,纷纷建立了以数据库为核心的 MIS 系统。软件需求的快速增长使软件产业遭遇到了明显的技术瓶颈。主要表现为:(1)软件开发中的低效率、低质量、高成本等问题伴随软件功能和复杂度的增加而日渐突出^[1];(2)软件运行中的维护和升级难度随实际需求的快速变化而越来越大;(3)客户需求的变化导致软件开发过程中的大量返工,并带来很多不确定性问题。因此,如何借鉴工业生产中自动化生产的思想,探索新的软件开发方法和技术,实现软件开发自动化,从而提高软件开发效率和质量是软件工程研究领域的热点问题。文献[2]中利用 DataSet 中所保存的数据表结构信息来自动生成三层架构下的实体层,但由于 DataSet 中不能保存数据表的主外键关系,因此无法在实体层中体现这种主外键关系,带来了很大的局限性。文献[3]中通过自定义主从表的关系来体现这种主外键关系,但由于不能利用物理数据库的本身所具有的主外键关系而需要用户手工进行定义,因此实用性也不强。其它的一些解决方案也大多停留在理论研究上,并未达到预期效果。

根据当前应用软件开发的现状和发展趋势,针对目前广泛应用的以数据库为核心的 MIS 系统,本文以 .NET3.5 开发平台为基础,提出了一种基于 PowerDesigner 和 CodeSmith 的软件开发自动化技术,实现了软件三层架构下代码的自动化开发,包括实体层、数据访问层、业务逻辑层、表现层,从而在实际应用过程中大大提高了软件开发效率,缩短了开发周期,提高了软件质量,并且能灵活应对开发过程中用户需求的不断变化。

1 系统整体架构

本文所实现的软件开发自动化技术基于数据模型驱动^[4]风格来进行设计。整个系统主要由数据模型、生成引擎、代码模板等几个部分组成。软件自动化开发平台的基本架构如图 1 所示,包括了 PowerDesigner 数据库设计平台、CodeSmith 代码自动生成平台、UI 界面资源包、自动生成的程序源代码 4 个部分。

PowerDesigner 是 Sybase 公司的一个 CASE 工具集。通过 PowerDesigner,系统分析员可以很方便地建立 DFD 图(数

到稿日期:2009-09-10 返修日期:2009-11-21 本文受江苏省高校自然科学基金项目(07KJB520096),南通市重大科技专项基金项目(XA2008004)资助。

朱晓辉(1976—),男,硕士生,讲师,主要研究方向为软件工程、计算机软件与理论,E-mail:zhufirst@vip.sina.com;王杰华(1965—),男,硕士,副教授,主要研究方向为数据加密、网络安全;石振国(1960—),男,博士,副教授,主要研究方向为人工智能、网络计算;陈苏蓉(1977—),女,硕士生,主要研究方向为分布式数据库。

据流程图)和 E-R 图(实体-关系图)^[5],从而完成数据库概念模型的设计。然后再通过 PowerDesigner 中内置的 DataArchitect 工具把概念模型直接转化为各种数据库系统的物理模型,从而自动完成物理数据库的创建。

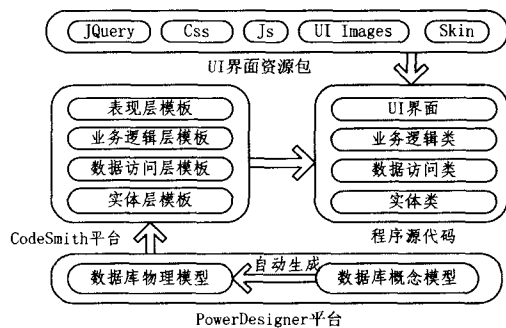


图1 系统整体架构

CodeSmith 是一个代码生成引擎工具,它可以根据所建立的数据库物理模型来执行代码模板并自动生成程序源代码。其中实体层模板根据物理模型中的数据表来生成对应的实体类,数据访问层模板生成封装有增删改查等常用数据操作的数据访问类,业务层模板生成对应的业务逻辑类,表现层模板分别针对数据表的增删改查 4 个操作生成相应的添加、修改、删除、查询 4 个 Web 页面。

UI 界面资源包主要为 Web 表现层提供额外的服务和支 持,包括了 JQuery 开源插件、自定义 CSS 样式表、自定义 JS 文件、图片资源等。UI 界面资源包和 Web 界面代码一起构成了软件系统的表现层。

2 软件自动化开发关键技术

2.1 基于 PowerDesigner 的数据库建模

概念模型的设计直接影响到数据库物理模型,而物理模型又是 CodeSmith 引擎自动生成程序源代码的基础。考虑到数据库物理模型中支持的一些语法规则及关键字等可能会与编程语言中的语法规则和关键字有冲突,因此,需要制定一套概念模型设计的标准来保证自动生成的程序源代码没有语法上的歧义性和二义性。下面列出概念模型设计的标准。

(1)概念模型中的表名和字段名不能是相应编程语言中的关键字。由于数据库中的每个表都将转换成对应的一个实体类,表名将相应地自动映射成实体类的类名,字段名映射成实体类中的成员变量名。因此,若表名和字段名是相应编程语言中出现的 关键字,自动生成的源代码将会出现语法错误。同时为了确保生成的源代码符合常用的命名习惯,约定概念模型中表名统一采用 Camel 命名法,字段名统一采用 Pascal 命名法。

(2)概念模型中表的外键字段名不能跟外键所在表的表名相同。由于外键字段在实体类中将被映射成外键所在表的一个实体类对象,两者名称若相同就会出现成员变量名和类型名一致的情况,这在任何程序设计语言中都是不允许的。

(3)概念模型中表名或字段名中禁止出现空格。虽然表名或字段名中出现空格在数据库系统中是允许的,但由于在编程语言中类名及变量名中间不能加入空格,因此概念模型中的空格将导致在代码生成时会出现类名及成员变量名中出现空格的情况,这是不允许的。

(4)概念模型中每个字段都应添加相应的注释。添加注

释可以更好地理解该字段的作用和含义。同时在生成实体类时,这些注释信息可以自动转化为实体类中每个字段的注释,这样将更大地增强程序源代码的可读性。

(5)概念模型中字段的注释必须按照如下的格式:“字段注释”+“\$”+“控件类型”。其中“\$”是一个分隔符,“控件类型”表示该字段在 Web 页面上对应的控件的类型。默认情况下“控件类型”为空,表示该字段在 Web 页面上的控件类型是文本框,以便表现层模板能根据“控件类型”的描述自动生成相应的控件及客户端验证代码。

在完成概念模型设计后,通过 PowerDesigner 的自动转化工具(PDM)直接转化成相应的物理数据库模型^[5],从而完成整个数据库系统的设计。

2.2 基于 CodeSmith 的代码模板设计

CodeSmith 是一个基于模板的代码生成工具,它使用类似于 ASP.NET 的语法来编写模板,用户可以通过自定义的模板来生成任意类型的代码或文本。与其他许多代码生成工具不同,CodeSmith 不局限于特定的应用程序设计或体系结构。使用 CodeSmith,可以生成包括简单的强类型集合和完整应用程序在内的任何东西。CodeSmith 本身就内置了专门用于获取数据库结构信息的数据库操作组件如“SchemaExplorer. TableSchemaCollection”,“SchemaExplorer. TableSchema”等来处理数据库中的表结构信息。通过这些组件可以获取数据库物理结构详细的设计信息如表名、字段名、字段类型、主外键关系、字段备注信息等,这使我们基于 CodeSmith 来直接分析数据库结构并自动生成基于多层架构的程序源代码成为了可能。基于 CodeSmith 的代码模板分为实体层、数据访问层、业务逻辑层、表现层 4 个部分,分别用于生成多层架构^[6]下对应层的程序源代码。

2.2.1 实体层模板

实体层模板通过 CodeSmith 代码引擎内置的数据库组件来获取物理数据库模型中的表结构,以单个表为单位,生成所有的实体类代码。由于一个数据表中的外键是另一个表的主键,因此外键在实体类中将被映射成该外键表所对应的实体类的对象。我们把表中的字段划分为 3 类:主键字段、外键字段、其它字段。主键字段唯一代表了一个实体对象,而外键字段代表了该实体对象中包含的子对象。表中每个字段都在实体类中映射成相应的成员变量和对成员变量进行存取操作的属性过程。对于外键字段还需要额外映射成外键对象。通过这种方法实现了物理模型中的表到实体层中实体类的一一映射和转换。为了更好地满足程序源代码书写规范,模板中定义了两个字符串格式化的方法 CamelFormat()和 PascalFormat(),以分别对字符串按照 Camel 命名法和 Pascal 命名法进行格式化输出。实体层模板的主要算法伪代码描述如下:

```
// 用 TableSchemaCollection 对象依次获取物理模型中所有表结构信息
foreach (数据表 tmpTable in 数据表集合)
{
    CamelFormat(tmpTable, tableName) // 生成实体类名
    // 用 TableSchema 对象依次获取表中所有字段信息
    foreach (字段 tmpField in tmpTable)
    {
        // 依次把字段名生成成员变量名
        PascalFormat(tmpField)
```

```

GetRemark(tmpField) // 获取该字段注释
// 创建该字段的属性过程
CreateProperty(tmpField)
If(IsForeignKey(tmpField)) // 判断是否为外键
{
    // 创建子对象
    CreateForeignObject(tmpField)
    // 创建子对象的属性过程
    CreateForeignProperty(tmpField)
}
}
}

```

2.2.2 数据访问层模板

通过分析可知, MIS 系统对数据库的操作主要表现为对表中数据的添加、修改、删除、查找 4 种。由于不同的系统有不同的业务需求, 也就有不同的数据访问形式, 因此, 如何设计数据访问层模板, 使其自动生成的数据访问层代码既能最大限度地包括各种形式的数据访问需求, 又能灵活方便地进行功能的扩展显得尤为重要^[7]。这里采用了 .NET 中分部类 (partial class) 的概念, 把一个数据访问类的代码分散到两个不同的类文件中, 其中一个类文件专门存放由模板自动生成的程序源代码, 另一个存放用户自定义的数据访问层代码, 从而实现了自动生成的程序代码和自定义的程序代码在物理上互相独立, 逻辑上又是一个完整的整体。当用户需求改变时, 可以只重新生成和覆盖以前由模板自动生成的代码部分, 确保自定义的代码不会被轻易覆盖和修改。为了尽可能满足各种数据访问需求, 为表中每个字段都定义了基于该字段的独立的查询数据和删除数据的方法。同时模板中还提供了可通过自定义 T-SQL 语句实现数据操作的通用方法, 以满足少数情况下用户自定义数据访问方法的需要。在多层架构中对数据库的查询操作都是通过返回实体类对象或对象集合的方式来实现的。因此在查询操作中还需要判断数据表中是否有外键。若有外键, 则需要调用该外键所对应的数据访问类中的相应方法来获得外键对象。数据访问层模板的算法描述如下:

```

// 依次获取物理模型中所有表结构信息
foreach (数据表 tmpTable in 数据表集合)
{
    // 生成数据访问类名
    String.Format("DAL"+ CamelFormat(tmpTable, tableName))
    // 依次获取表中所有字段信息
    foreach (字段 tmpField in tmpTable)
    {
        // 生成以该字段名为形参的查询方法
        CreateSelectMethod(tmpField)
        // 生成以该字段名为形参的删除方法
        CreateDeleteMethod(tmpField)
        // 生成以该字段名为形参的判断方法
        CreateIsHaveMethod(tmpField)
        If(IsPrimaryKey(tmpField)) // 判断是否为主键
        {
            // 生成以该类对象为形参的修改方法
            CreateUpdateMethod(tmpTable)
        }
    }
}
}

```

```

// 生成一个通用的以 SQL 语句为参数的查询方法
CreateComnSelectMethod(SQL)
}

```

其中在 CreateSelectMethod(tmpField)方法中还需要判断 tmpField 是否是外键, 若是外键的话, 则还需要生成外键字段所对应的子类对象, 其算法描述如下:

```

CreateSelectMethod(tmpField)
{
    If(IsForeignKey(tmpField)) // 判断是否为外键
    {
        CreateForeignObject(tmpField) // 创建子对象
    }
}
}

```

2.2.3 业务逻辑层模板

业务逻辑层模板根据已经生成的实体层、数据访问层及数据库物理模型来自动生成相应的业务层代码。由于不同的具体应用, 软件的业务逻辑会有很大差异, 因此如何能最大限度地自动生成业务逻辑层的代码是需要考虑的一个重点问题。这里我们把业务逻辑层代码分为 3 个部分: 一是对数据访问层的简单调用和封装; 二是常用的、通用的业务逻辑如登录、注册、退出等; 三是与具体应用密切相关的业务逻辑。由于第一和第二部分的业务逻辑层代码跟数据库物理模型密切相关, 因此通过业务逻辑层模板可以实现这两个部分代码的自动生成。而第三部分代码与具体的业务需求有密切关系, 因此需要用户进行自定义开发。但因为有了前面两部分的自动生成的业务层源代码, 用户完全可以基于已有的业务层代码来进行自定义开发, 所以工作量也将大大减少。业务逻辑层模板也采用了分部类的技术, 以便能对业务逻辑层的功能进行动态扩充。业务逻辑层的算法描述如下:

```

// 依次获取物理模型中所有表结构信息
foreach (数据表 tmpTable in 数据表集合)
{
    // 生成业务逻辑类名
    String.Format("BLL"+ CamelFormat(tmpTable, tableName))
    // 依次获取表中所有字段信息
    foreach (字段 tmpField in tmpTable)
    {
        // 创建通用的业务逻辑层方法
        CreateComnBLLMethod()
        // 封装和调用访问层的查询方法
        CallSelectDALMethod(tmpField)
        // 封装和调用访问层的删除方法
        CallDeleteDALMethod(tmpField)
        // 封装和调用访问层的判断方法
        CallHaveDALMethod(tmpField)
        If(IsPrimaryKey(tmpField)) // 判断是否为主键
        {
            // 封装和调用访问层的修改方法
            CallUpdateDALMethod(tmpTable)
        }
    }
    // 封装和调用访问层的通用查询方法
    CallComnDALSelectMethod(SQL)
}
}

```

2.2.4 表现层模板

表现层模板根据数据库物理模型及已生成的实体层和业务逻辑层代码依次为每个数据表自动生成添加、修改、删除、查询 4 个 Web 页面代码。由于 Web 页面涉及到 JS 文件、CSS 样式、图片资源及 JS 插件等多种页面元素,因此表现层模板需要借助于已定义的 UI 界面资源包才能完成整个表现层代码的自动生成^[8]。在 UI 界面资源包中包含了自定义 CSS 样式表、各种按钮所需要的图标、各种背景图片、常用的 JS 文件及第三方的开源 JS 插件-JQuery 等。其中 JQuery 开源插件主要用于对页面上的控件元素进行验证及页面布局的控制。表现层代码的生成难点在于对应于数据表中的字段,应该用哪种控件在页面上来呈现和表示。这里通过在概念模型设计时规定字段的注释必须按照如下的格式来填写:“字段注释”+“\$”+“控件类型”。其中“控件类型”即表示该控件在 Web 页面上需要呈现的控件类型。默认情况下,若字段没有呈现信息的关键字,则用文本框控件来表示该字段;若该字段是表的外键时,则用下拉列表框来表示;否则按照“控件类型”的描述来生成页面控件。表现层模板的基本算法描述如下:

```
//依次获取物理模型中所有表结构信息
foreach (数据表 tmpTable in 数据表集合)
{
    CreateAddPage(tmpTable)//创建添加纪录页面
    CreateUpdatePage(tmpTable)//创建更新纪录页面
    CreateDeletePage(tmpTable)//创建删除纪录页面
    CreateQueryPage(tmpTable)//创建查询纪录页面
}

其中 CreateAddPage(tmpTable)方法的主要算法描述如下:
CreateAddPage(tmpTable)
{
    //依次获取表中所有字段信息
    foreach (字段 tmpField in tmpTable)
    {
        //根据备注字段判断应产生何种页面控件
        switch( remark of tmpField)
        {
            //默认为空或 textbox
            Case “” or “textbox”:
                CreateTextBoxControl()//生成文本框控件
            Case “checkbox”
                CreateCheckBoxControl()//生成复选框
            Case “dropdownlist”
                CreateDropDownControl()//生成下拉列表框
            .....
        }

        //根据字段类型调用 JQuery 插件完成数据验证。
        switch(type of tmpField)
        {
            Case “int”
                生成 int 型 JQuery 验证脚本的代码。
            Case “string”
                生成 string 型 JQuery 验证脚本的代码。
            Case “datetime”
                生成 datetime 型 JQuery 验证脚本的代码。
            .....
        }
    }
}
```

}
} 其它几个相关的页面创建算法与以上的算法类似,这里不再赘述。

3 应用实例分析

在一个基于 B/S 架构的 CNPrice 电子商务项目中采用了本项目的软件自动化开发技术。该项目是一个以国外买家为用户群的电子交易平台,其内容界面如图 2 所示。

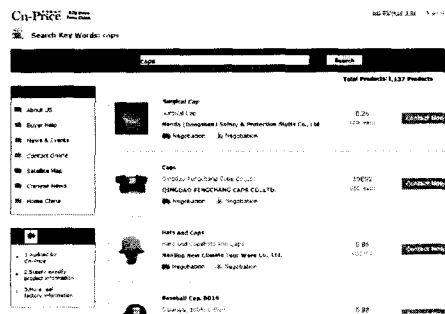


图 2 CNPrice 内容界面

其数据库由 104 个表构成,整个项目的后台代码量为 7 万余行。自动生成代码与自定义代码的分布比例情况如图 3 所示。

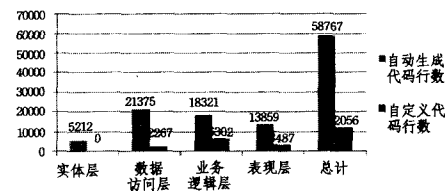


图 3 代码分布比例

其中,实体层代码 100% 自动生成,数据访问层代码的自动生成率达 90.4%,业务逻辑层代码的自动生成率为 74.4%,表现层代码的自动生成率为 79.9%,平均代码自动生成率为 82.9%。从以上分析数据可以看出,采用了本文所提出的软件开发自动化技术后,大大提高了软件开发的速度和效率。同时由于采用了分部类技术,使得自动生成的代码和自定义扩展的代码可以分开保存,因此即使当底层数据库进行大规模调整,也只需简单地通过 CodeSmith 引擎及代码模板重新生成所有代码,并对自定义扩展的代码做相应的调整即可,大大减轻了因用户需求变化而引起的软件维护的工作量。

结束语 本文提出的软件开发自动化技术较好地实现了软件的自动化开发,能灵活应对用户需求的不断变化和调整。通过 PowerDesigner 来完成数据库概念模型和物理模型的建立,并采用 CodeSmith 引擎及模板技术来实现软件开发过程的自动化。从实际应用可以看出,本方案能自动生成大部分的代码,具有较强的实用性。

参考文献

- [1] 尤枫,汪须忠,李铮,等. 应用软件自动化开发的实现研究[J]. 计算机工程与设计,2007,28(10):2478-2480
- [2] 张静,孔芳,杨季文. 一个基于 Java 的代码生成工具的设计与实现[J]. 微电子学与计算机,2007,24(6):222-224

(下转第 247 页)

出了对比结果。

表 2 KLWFD 与经典说话人识别方法 GMM,SVM 的对比结果

	5 person		10 person		20 person	
	ER	Speed(s)	ER	Speed(s)	ER	Speed(s)
GMM	1%	1.24	2%	2.15	3.5%	3.56
SVM	0%	0.21	0.5%	0.52	2.75%	0.86
KLWFD	0%	0.43	0%	0.85	2.5%	1.52

表 2 中速率为语音测试时每个语音片段平均识别需花费的时间。可以看出,基于核类内特性保持 Fisher 判别方法的说话人辨别不论从识别率还是从识别速度上均全面优于传统的经典说话人识别技术高斯混合模型法。在与支持向量机的对比测试中,本文方法的识别率占优,但识别速度略逊一筹,这是由支持向量机的稀疏性所致。但支持向量机是一种二元分类器,应用于多元分类时,必须构建多个二元分类器以间接实现多分类。如本例进行 20 人辨别实验时,采用一对多分类方法的支持向量机必须构建 $C_{20}^2=190$ 个不同的二元分类器,这大大地增加了训练模型的复杂度与繁琐性。而核类内特性保持 Fisher 判别方法可直接实现多元分类,训练一次就能实现分类模型,相比支持向量机,更加便捷与人性化。

结束语 Fisher 判别分析与局部保持 Fisher 投影都是降维分类的热门技术,但面对具有不同分布特性的分类数据时,都不能全面兼顾。本文折衷考虑全局投影与局部保持的平衡,将样本之间亲和力以权值形式融入类内散度矩阵,提出类内保持 Fisher 判别方法,以适应不同分布的分类数据。并利用核技巧非线性化,使之能应付线性不可分的分类场合。核化过程导致算法计算复杂度剧增,当训练样本增加时,内存量与计算量过大使实时应用性降低。本文对训练算法进行修改,使应用于说话人辨别等实际问题成为可能。但是核矩阵本身随样本量增多而耗费大量内存空间,说话人辨别应用时总人数受到限制,因此改进这类问题将是我们今后着力研究的方向。

参 考 文 献

[1] Hinton G E,Salakhutdinov R R. Reducing the dimensionality of data with neural networks[J]. Science, 2006, 313(5786): 504-507

[2] Baudat G, Anouar F. Generalized discriminate analysis using a kernel approach[J]. Neural Computation, 2000, 12(10): 2385-2404

[3] Belhumeur P N, Hespanha J P, Kriegam D J. Eigenfaces vs.

Fisherfaces; Recognition using class specific linear projection [J]. IEEE Trans. on Pattern Anal. Machine Intell,1997,19(7): 711-720

[4] Yang J, Yang J-Y. Why can LDA be performed in PCA transformed space? [J]. Pattern Recognition,2003,36(2):563-566

[5] 邢玉娟,李 明,张亚芬. 基于 PCA 和核 Fisher 判别的说话人确认[J]. 计算机工程与设计,2008,29(15):3983-3985

[6] 谭萍,邢玉娟,李明. 基于核 Fisher 判别的说话人辨认[J]. 科学技术与工程,2008,8(8):2230-2233

[7] He X F,Niyogi P. Locality Preserving Projections[A]// Thrun S, Saul L, Scholkopf B, eds. Advances in Neural Information Processing Systems 16[C]. Cambridge,MA: MIT Press,2004

[8] Sugiyama M. Dimensionality Reduction of Multimodal Labeled Data by Local Fisher Discriminant Analysis[J]. Machine Learning Research,2007,8:1027-1061

[9] 徐勇,杨静宇,金忠,等. 一种基于核的快速非线性鉴别分析方法[J]. 计算机研究与发展,2005,42(3):267-374

[10] 高秀梅,杨静宇,杨健. 一种最优的核 Fisher 鉴别分析与人脸识别[J]. 系统仿真学报,2004,16(12):2864-2868

[11] Liu Qing-Shan,Huang Rui. Face recognition using kernel based fisher discriminant analysis[A]//Proceedings of the 5th International Conference on Automatic Face and Gesture Recognition [C]. Washington,DC, USA,2002:187-191

[12] Liu Wei, Wang Yun-hong. Null space based kernel fisher discriminant analysis for face recognition[A]// Proceeding of the 6th International Conference on Automatic Face and Gesture Recognition[C]. Seoul,Korea,2004:369-374

[13] Mika S,Rätsch G, Weston J, et al. Fisher discriminant analysis with kernels[A]//Proceedings of IEEE International Workshop on Neural Networks for Singal Processing [C]. Madison, Wisconsin, August 1999:41-48

[14] He X F, Yan S, Hu Y, et al. Learning a Locality Preserving Subspace for Visual Recognition[A]//Proceedings of Ninth IEEE International Conference on Computer Vision[C]. Nice, France, 2003:385-392

[15] Belkin M,Niyogi P. Laplacian eigenmaps for dimensionality reduction and data representation[J]. Neural Computation, 2003, 15(6):1373-1396

[16] Zelnik-Manor L,Perona P. Self-tuning spectral clustering[A]// Saul L K, Weiss Y, Bottou Y, eds. Advances in Neural Information Processing Systems 17[C]. Cambridge, MA: MIT Press, 2005:1601-1608

(上接第 159 页)

[3] 金雪云,徐卫东,刘红卫. 基于 .NET 的代码生成工具的设计与实现[J]. 计算机工程与应用,2004,21:123-125

[4] 陈翔,王学斌,吴泉源. 代码生成技术在 MDA 中的实现[J]. 计算机应用研究,2006,1:147-150

[5] 轩兴涛. 基于 PowerDesigner 模型驱动机制下的全程建模研究 [J]. 西安石油大学学报:自然科学版,2008,23(6):104-106

[6] 何天,侯宗浩. 基于 Petshop 与 Duwamish 的多层架构设计与实

现[J]. 计算机应用,2006,26:257-259

[7] 李梦,李凡,李京. Web 应用自动构建框架——WACF[J]. 计算机工程,2007,33(9):97-99

[8] 张海英,万建成. Web 用户界面自动生成系统中的界面模板 [J]. 计算机工程与设计,2006,27(18):3491-3493

[9] Griffiths T, Barclay P. Teallach: A model-based user inter-face development environment for object database[Z]. Los Ala-mitos,CA: User Interfaces to Data Intensive Systems,1999:86-96