第 35 卷 第 9 期 计 算 机 工 程 2009 年 5 月
Vol.35 No.9 Computer Engineering May 2009
·软件技术与数据库· 文章编号：1000—3428(2009)09—0060—03 文献标识码：A 中图分类号：TP31;TP39

基于元数据的快速开发平台设计与实现

蔡昭权，卢庆武，郑宗晖，罗 伟
(惠州学院网络中心，惠州 516007)

摘 要：针对应用软件需求的快速变更和系统开发的慢速响应之间的矛盾，提出一种全新的系统开发方法，利用元数据和构件方法，采用SOA 的思想，将业务逻辑封装到构件内部，以工作流技术控制流程，实现软件系统的快速开发平台。测试结果表明，采用该快速开发平台，可加快应用软件的开发速度，提高软件的质量，并可以在多种行业使用。

关键词：元数据；构件；工作流；面向服务；快速开发

Design and Implementation of Rapid Development Platform Based on Metadata

CAI Zhao-quan, LU Qing-wu, ZHENG Zong-hui, LUO Wei
(Network Center, Huizhou University, Huizhou 516007)

【Abstract】Aiming at the contradiction between fast changing needs of applications and slow response of software development, according to SOA and metadata, by encapsulating business logic in software component, this paper implementes a new rapid development platform using workflow technology to control the process. Test results show that the new platform can speed up software development and enhance software's quality, and it can be applied in a wide range.

【Key words】metadata; component; workflow; service-oriented; rapid development

1 概述

随着网络技术和电脑的普及应用，各个行业都在推广信息管理系统，无论是物料管理系统(MRP)、客户管理系统(CRM)、还是企业管理系统(ERP)，都是由专业软件工程师通过某种固定的语言开发，交付给企业管理者使用。由于软件工程师一般缺乏企业管理的专业知识，使得开发和使用者之间有一道难以沟通的鸿沟，开发出来的产品往往似是而非，不能全面满足客户需求。并且这些系统都是由上万、甚至上十万、百万行代码完成的，开发工作量巨大，系统很复杂，一旦完成就很难修改，往往牵一发而动全身，导致客户难以更新。针对上述问题，利用元数据和组件技术结合，开发出了一套快速开发平台，构建过程完全图形化、可视化。平台可根据用户管理需求任意构建应用管理系统，也可随意以B/S, C/S 混合结构方式构建应用系统，同时也可运行多种语言管理窗口。针对数据的增、删、改、查，在组件内部即可解决[1]。使用者变成开发者，从而快速解决业务和技术难以沟通的问题[2]。

市面上快速开发平台不少，包括基于 Java 的 FastUnit、CHARISMA、基于.Net 的极致软件等，在下面 3 个方面用到了元数据配置：数据实体元数据，用户界面元数据，业务流程元数据，可配置性不足。本系统在各方面都采用了元数据管理，使程序更加灵活、容易扩展；FastUnit 等采用的是直接管理数据库的功能、界面操作复杂，缺少模板，从而使二次开发变得困难，难以掌握。本系统对于几个行业有现成的模板供编辑，从而大大加快了开发流程。最关键的是上面的系统都没有封装业务逻辑，从而增大了二次开发的难度，增加了二次开发的成本。

2 设计思路

元数据是“关于数据的数据”，它不仅影响到系统的构建，还影响到程序的运行逻辑[3-5]。考虑到用户的信息记录方式和表现方式经常改变，采用元数据方式配置用户的信息描述，在呈现的时候按照用户设定的方式呈现。在系统架构中，通过元数据来控制程序的运行，将每个行业归纳出来的公共的、构成体系的东西加以综合，再针对领域知识相关行业独特的属性，采用元数据的方式加以补充，然后将整体打包成一个SOA 构件[6-7]，此构件中包括相应的业务逻辑，并向外提供标准的接口，再将这些构件通过系统架构组合起来。

系统架构如图 1 所示，分为物理数据层、数据驱动层、基础构件层、集成构件层、产品框架层。

(1)物理数据层：负责数据的存储[8]，支持 MSSQL, MySQL, Oracle, DB2 等主流数据，也可以采用 Access、文本文件等形式保存小量本地数据，在离线状态下，将数据暂时保存到本地，待网络接通时统一提交，物理数据层还负责元数据的管理。

(2)数据驱动层：数据驱动层在系统中占据核心的地位，数据驱动不仅实现和物理数据层的交互、逻辑对象和物理结构的映射，还实现了逻辑规则和逻辑数据的封装与数据模型

基金项目：惠州市科技计划基金资助项目(2008G11, 2008G14, 2008P06)

作者简介：蔡昭权(1970－)，男，副教授、硕士，主研方向：计算机网络，软件技术，数据库，信息安全；卢庆武、郑宗晖、罗 伟，工程师、硕士

收稿日期：2008-12-05 **E-mail：**cai@hzu.edu.cn

的建立和智能化驱动，是完全基于面向对象的数据建模与驱动平台，在这一层实现了数据建模、数据和界面的关联、数据的自动存取等功能。考虑到效率的问题，本层用 C++开发，如图 2 所示。

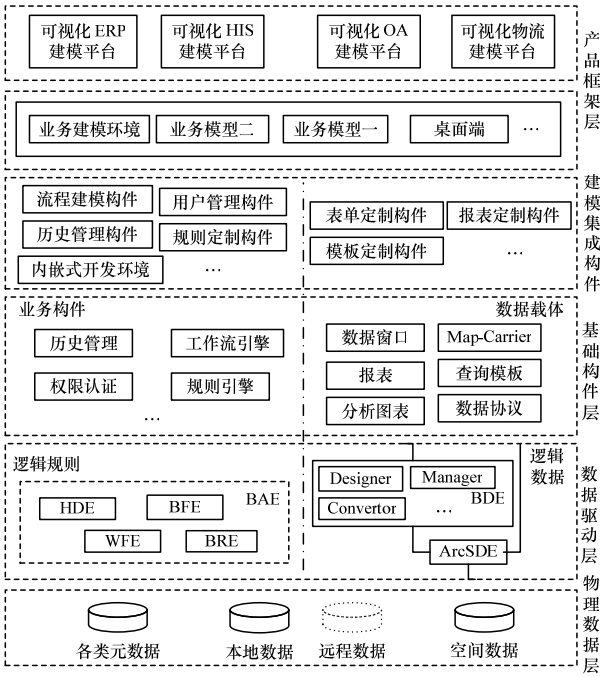

图 1　系统架构

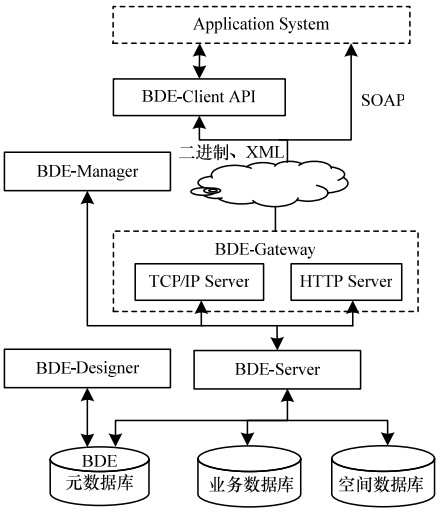

图 2　数据驱动方式

(3)基础构件层：基础构件层实现了基本控件、报表、图标等的数据封装，例如文本输入框，在原来的基础上增加了数据对象以实现文本和逻辑对象的关联、增加了数据规则实现数据验证等功能，还实现了基本的业务构件的管理，例如关键数据(元数据)的修改记录(历史管理)、基于角色的权限认证管理、工作流引擎(采用 WWF)和工作流组件(Activity)、业务规则管理(规则引擎)等的管理，本层采用 C#语言开发。

(4)建模、集成构件层：基础构件层只是一些比较基础的控件和对基本引擎的管理，对基础构件层的进一步抽象，形成了集成构件层。包括对某个逻辑对象的页面生成、模板生成、规则定制页面生成、工作流定制页面生成、业务模型建立等功能。高级用户(管理员)操作这个层面的程序，定制给业务人员使用的程序逻辑和业务流程，采用 C#开发。

(5)产品框架层：产品框架层包括了建立业务模型的环境和程序固化的模板，业务人员可以选择其中的某个模板，并在业务建模环境中修改、配置，使系统适应自己的使用规则和使用习惯。这个提供给普通用户使用，本层采用 C#开发。

3　实现

本系统每个模块的实现采用 SOA 规则，即每个部分都是向外界提供服务，下面简单介绍元数据、数据驱动、基础构件和系统启动流程等关键部分的实现方法。

3.1　元数据实现

本系统分模块进行各个部分的实现，下面将元数据记录及其表现做简单介绍。

为了保存用户建立的数据表和数据列，用 7 个表来存储元数据的基本信息和修改记录。为节省篇幅，没有列出数据存储列的详细信息，简单列出关键表的信息，如下：

Meta_ObjectEntity → 对象描述、元数据描述表，例如产品、出货记录表等

Meta_ObjectEntityField → 对象的栏位或者属性描述元数据描述表

Meta_ConfigurableTable → 存放用户配置的表信息

Meta_ConfigurableTableField → 存放用户配置的列信息

Meta_ConfigurableTableRelation→ 记录表之间的关联

Meta_ConfigurableTableValue → 记录表的值，用于行数据的权限管理

Meta_ConfigurableLogicRule → 记录业务逻辑关系

Meta_SchemaChangeLog → 元数据修改记录，用于程序启动时元数据版本检查

实体类图如图 3 所示。

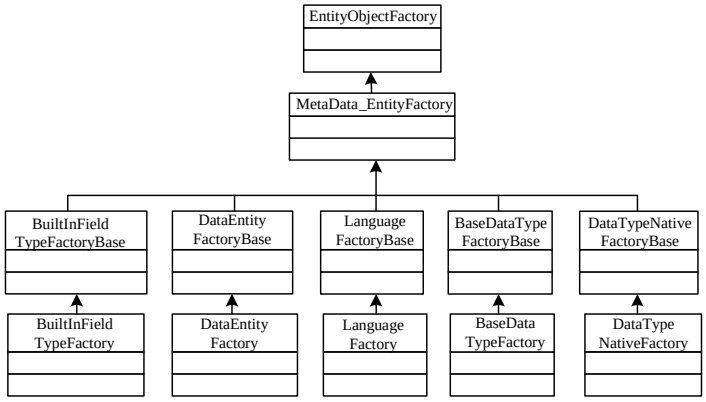

图 3　实体类图

其中，基类是实体对象，元数据实体对象从中继承出来，实体对象中主要包括了实体对象里面包括了实体的 DataRow, PrimaryKeyValue, EntityObject, ObjectHiberarchy, EntityMap, ObjectContext, EntityObjectList 等虚属性；同时还包括了对于实体操作 Delete, DeleteChildEntityObject, GetChildEntity ObjectList, GetReleatedObject, RejectChanges, GetProperty, SetProperty, SetReleatedObject, Children 等虚方法；也包括了 AddDetailEntityChildren 的具体实现。

实体对象通过增加子实体、移除实体、获取子实体列表、以及实体之间的关系等函数，实现对实体对象内部的统一操作。利用元数据的这种存储方式，可以自定义任意方式的表单。

3.2　数据驱动实现

数据驱动不仅仅实现数据的增、删、改、查和数据交互，而且实现了业务对象和业务逻辑、业务规则的封装，图 2 说明了数据在系统层次间的交互，图 4 说明了数据驱动的组成

部分和实现方法。

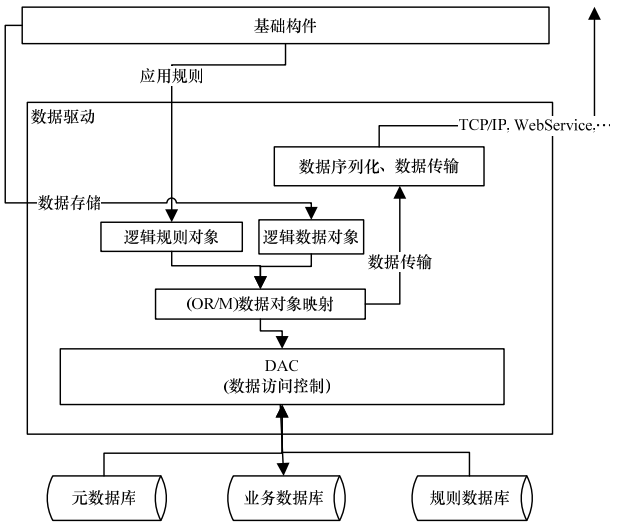

图 4　数据驱动组成描述

3.3　基础构件实现

基础构件包括文本输入框、表单、编号框、选择框、编辑框、组合框、按钮、图表框、数据列表、树状控件、子表单、OLE 对象等编辑控件。基于开发效率的考虑，采用 C# 开发，每个控件增加标签属性、数据对象属性、权限属性等，从而使基础构件本身就是数据的一部分。可以和业务数据库直接绑定，编辑框中增加数据验证功能等。图 5 是部分 Web 控件的类图。

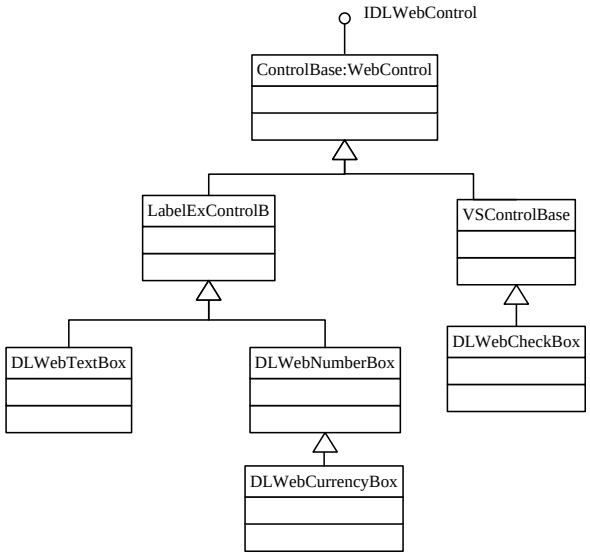

图 5　部分 Web 控件类图

在 ControlBase 中，集成了权限设置，这样的权限可以具体到每个控件，其中权限部分的代码如下：

```
public bool HasRight(AccessFactory factory, string access, string number, out string noRightTip)
//factory 是从 ObjectEntity 集成的 access 对象工厂
//access 是以“；”分割的权限字符串
//number 是权限的位置
//noRightTip 是当发现没有权限的时候，输出字符串
        {
            bool flag;
            if (string.IsNullOrEmpty(access) || (access == "0"))
            {
                noRightTip = "";
                return true;
            }
            Hashtable rightMessages = RightMessages;
    //RightMessages 是没有有权限的信息提示，由于访问频繁，需要缓存
//要缓存
            if (rightMessages != null)
            {
                flag = access.IndexOf(";" + number + ";") > -1;
//找到标志
                noRightTip = rightMessages[number]. ToString();
//返回信息
                return flag;
            }
            flag = false;
            noRightTip = "权限代码未找到";
            return flag;
        }
```

3.4　系统启动更新过程

该系统的基础构件层和数据驱动层基于效率因素的考虑，采用 C++开发，其他模块层使用 C#语言实现，当程序启动时需要检查本地元数据及其版本情况，如果不是最新版本的元数据数据，需要下载更新。

启动流程如图 6 所示。

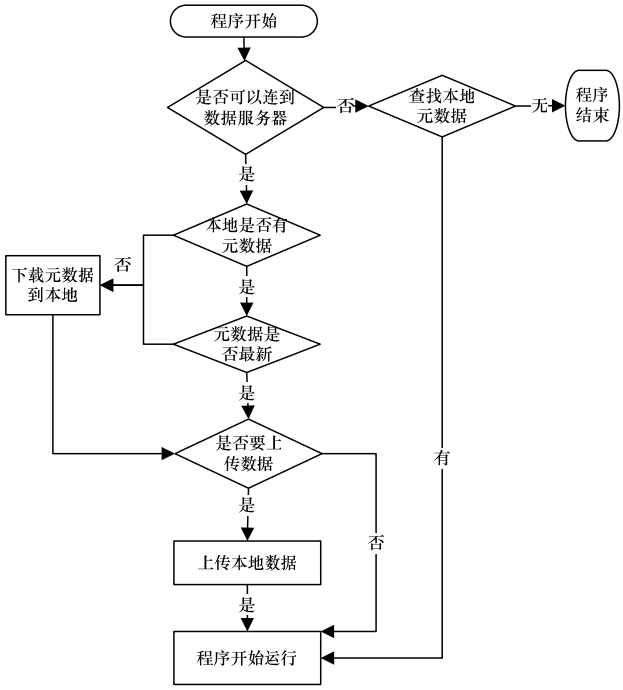

图 6　系统启动流程

4　应用效果

该系统是一个以元数据为基础的可视化快速开发平台，当前已开发出了 OA 建模平台和 HIS(医院系统)建模平台，但是在开发的过程中，已经显示了其相对于一般开发工具的优越性和广泛的适应性。

系统优越的快速构建方式决定了它具有低成本、高效率、功能强大的技术优势和竞争优势，使其他开发系统难以抗争。如表 1 所示，同样开发一个大型系统，系统构建方式与常规方式相比，无论是从市场角度还是开发角度，都占有绝对的优势。

（下转第 65 页）