# 领域变化性分类与控制机制研究

张世琨, 胡文蕙, 陈兆良, 王立福
(北京大学信息科学技术学院, 北京 100871)

摘 要: 客观世界存在的变化性以及用户的个性化需求, 要求软件系统具有足够的灵活性. 本文提出了一个开放的变化性控制框架, 从组织机构、功能、数据、表示和过程连接等五个维度讨论了变化性出现的根源、变化性表现形式、变化性控制机制和实现技术, 提供了一个系统化的变化性解决方案. 并以商业领域应用软件开发平台的实践为例, 研究变化性控制在特定领域的应用.

关键词: 领域变化性; 变化性维度; 变化性控制框架; 工作流引擎
中图分类号: TP311 文献标识码: A 文章编号: 0372-2112 (2004) 03-0446-06

# Study on Classification and Control Mechanism of Domain Variability

ZHANG Shi-kun, HU Wen-hui, CHEN Zhao-liang, WANG Li-fu
(School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

Abstract: Variability existing in real world and customized requirements of users, require adequate flexibility of software systems. This paper proposes an open variability control framework, which discusses origin of variability, representative forms, control mechanisms and implementation technologies from five dimensions of organization, functionality, data, presentation and process connection. This framework provides a systematic variability solution. As an example, practices of variability control in business domain application development platform are also discussed.

Key words: domain variability; variability dimension; variability control framework; workflow engine

## 1 引言

客观世界存在的变化性以及用户的个性化需求, 要求软件系统具有足够的灵活性. 对于变化性的研究, 目前大多集中在特定开发范型和实现技术方面, 例如, 文[1]中讨论了面向对象范型提供的机制, 针对不同变化性情况提供了解决能力和解决方案; 文[2]中尝试性地提出了一个变化性术语和概念框架, 对变化性情况和处理技术进行了探讨; 文[3]中提出的面向侧面的编程(Aspect-Oriented Programming, AOP), 支持程序员明确地分离构件和侧面, 同时提供集成的机制, 使得可以把它们组装成为一个完整的系统. 与 AOP 类似的变化性控制技术还有: 面向主题的编程(Subject-Oriented Programming)[4], 目标编程(Intentional Programming)[5], 变化的面向对象编程(Variational Object-Oriented Programming)[6]等. 这些设计和实现技术在变化性控制原理、机制和采取的策略等方面各不相同, 所能解决的变化性情况、适用范围和控制阶段也不一样, 目前仍然缺乏系统化的控制变化性的手段.

本文从变化性维度、产生的根源、表现形式、控制机制和实现技术等方面提出一个比较完整的变化性控制框架, 并对框架内包含的各类元素进行了讨论, 然后以商业领域应用软件开发平台的实践为例, 研究变化性控制在特定领域的应用.

## 2 变化性控制框架

领域变化性主要源自需求、实现技术和环境等的变化. 德国 Scheer 教授提出的一套完整的企业过程设计、过程管理的概念和方法))) ARIS (Architecture of Integrated Information Systems). 企业的业务过程涉及活动、职责以及相关的基础设施. 为了清楚地描述企业的业务过程, ARIS 模型包括五个不同的视图: 组织视图、数据视图、功能视图、过程视图和产品/服务视图, 从而降低了描述业务过程的复杂性. 在一个软件系统中, 除了上述视图以外, 还有表达人机界面的/表示(Presentation)视图0, 这是用户需求的一个方面, 同时也是软件系统的一个组成部分.

从信息加工和流动的角度, 软件系统可看作是一个信息处理过程, 包括活动和活动之间的关系, 后者由连接件(Connector)表达, 常见的连接模式可参见文[8]中的相关内容. 活动可以进一步由该活动所完成的功能、输入输出数据、实施活动的角色, 以及呈现给用户的表示等元素组成; 连接件用于表达活动之间的控制转移, 其中包括了引发转移的事件和转移中的动作. 因此, 过程和活动可定义如下:

收稿日期: 2002-10-24; 修回日期: 2003-03-17
基金项目: 国家"九五"重点科技攻关项目(No. 98-780-01-01); 国家 863 高科技发展计划(No. 2001AA113171); 教育部博士点基金(No. 20020001016)

过程J = {活动, 连接件}

活动J = 3功能, 数据, 角色, 表示4

图 1 是表达一个软件系统需求的模型片断. 从过程驱动的角度, 功能、数据、角色、表示和连接件形成了对问题域成分的一个划分, 是表达软件系统需求的基本要素, 它们在刻画软件系统需求方面是完备的. 因此, 从这五个维度出发对变化性进行分析和处理也是完备的, 它们之间是/ 正交0的.

基于以上分析, 本文提出了一个系统化的变化性控制框架, 如表 1. 该框架以领域变化性的五个维度为出发点, 通过分析问题域( 或业务领域) 中引起变化产生的根源, 以及在应用系统中的各种表现形式, 提出相应的变化性控制机制; 并采用合适的技术来实现这些控制机制, 以适应变化性的需要.

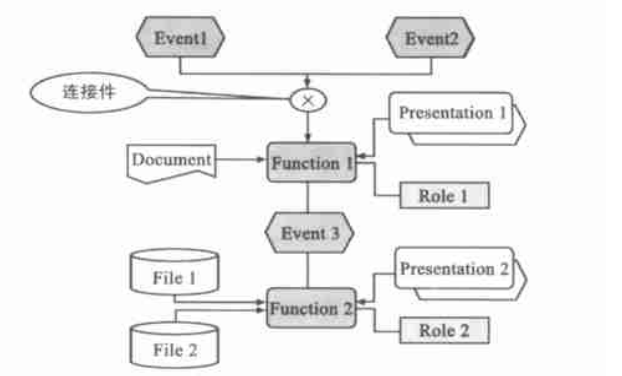


图 1　一个软件系统需求的模型片断

表1　变化性控制框架

| 变化性维度 | 变化性根源( 问题域) | 变化性表现形式( 应用系统) | 变化性控制机制 | 变化性控制实现技术 |
|---|---|---|---|---|
| 组织机构 | ¹ 组织机构差异<br>º 机构变革和人事调整 | ¹ 业务流程的变化<br>º 业务功能及相关数据的变化<br>» 用户权限的变化 | ¹ 基于角色的控制 | ¹ 参数化( 例如, 模板/ 宏, 条件编译, 运行参数/ 配置文件等)<br>º 接口和实现相分离( 例如, 构件接口和实现, 用户- 角色- 功能等)<br>» 软件体系结构<br>¼ 软件框架<br>½ 设计模式<br>¾ 虚拟机/ 中间件<br>¿ 面向对象提供的机制: 继承、多态、聚集/ 代理等<br>À 构件技术提供的机制: 聚集/ 代理、构件集成框架( 例如, CORBA、COM、EJB、Web Service)<br>Á 对编程技术的扩展( 例如, 帧、AOP 等)<br>Â 动态绑定技术( 例如, 插件、动态链接库等) |
| 功能 | ¹ 系统业务功能/ 对外提供的服务发生变化 | ¹ 业务功能的增加和删除<br>º 业务功能的演化和修改 | ¹ 标准化的构件接口和集成机制( 例如, 各种插件, Web Service 等)<br>º 脚本语言 | |
| 数据 | ¹ 系统业务功能/ 对外提供的服务发生变化<br>º 实现技术/ 运行环境发生变化( 不同数据库管理系统)<br>» 异构平台/ 系统之间的数据交换 | ¹ 数据结构/ 内容/ 格式的变化<br>º 数据物理存储环境的变化<br>» 数据加工和数据约束, 包括数据项本身以及数据项之间 | ¹ 数据内容和形式相分离( 例如, 元数据库与业务数据库, XML 的内容和形式等)<br>º 数据和操作相分离( 例如, 三层/ 多层体系结构)<br>» 数据库中间件( 包括标准数据库访问接口, 例如 ODBC)<br>¼ 标准数据表示和访问协议 XML<br>½ 公式( 数据约束和简单的数据加工) | |
| 表示 | ¹ 输入和输出手段发生变化 | ¹ 表示方式的变化 | ¹ 数据内容、加工和形式相分离( 例如, 三层/ 多层体系结构)<br>º 界面定制 | |
| 过程连接 | ¹ 组织机构、人员职责变化º 业务流程本身的改变 | ¹ 业务流程的变化 | ¹ 工作流引擎 | |

## 3　组织机构变化性控制

表 1 中列举了组织机构变化性的原因和表现, 下面用一个例子来说明这种变化性为企业带来的不便. 在电子政务的公文流转过程中, 假设 X 申请报告应该递交到 Y 局的负责人 A 手中, 那么就应该将申请报告以电子邮件( 或其他合适的方式) 发送到负责人 A 的邮箱中. 当 Y 局的负责人由于人事变动更换为 B 时, 需要修改程序将发送地址改为目前负责人 B 的邮箱. 在一个组织中, 由于人员流动、员工的职务变迁等原因引起人事变动是随时都有可能发生的事情, 将导致频繁地修改代码, 为系统维护带来了很大的不便.

采用基于角色的功能、数据和权限控制, 可以有效地解决这类变化性问题.

基于角色的控制涉及三个概念: 角色是指一个组织或任务中的工作或位置, 它代表了一种资格、权利和责任; 用户是一个可以独立访问计算机系统的功能、数据以及其他用数据表示的资源的主体, 一般情况下指的是人; 权限是对计算机系统的功能、数据以及其他用数据表示的资源进行访问的许可.

基于角色的权限控制模型如图 2, Users、Roles、Permissions 分别代表用户、角色和权限的集合, 这里有两个二元关系:

角色指派: Users 与 Roles 之间的二元关系, 用( u, r) 表示用户 u 被指派了一个角色 r.

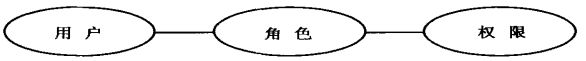


图 2 基于角色的权限控制示意

权限分配: Roles 与 Permissions 之间的二元关系, 用( r, p) 来表示角色 r 拥有一个权限 p.

通过指派和取消角色来完成用户权限的授予和取消. 系统管理员根据需要定义各种角色, 并设置合适的访问权限, 而用户根据其责任和资历再被指派为不同的角色. 这样, 整个访问控制过程就分成两个部分, 即访问权限与角色相关联, 角色再与用户关联, 从而实现了用户与访问权限的逻辑分离, 可以有效地应对组织结构的变化, 方便组织机构的调整.

实践表明, 角色/ 权限之间的变化比用户/ 权限之间的变化相对要慢得多, 并且给用户指派角色很容易. 基于角色的控制方法可以方便权限管理, 而且很好地描述了角色层次关系, 实现最少权限原则和职责分离的原则.

如图 3 所示, 采用基于角色控制方法的系统应该包括以下一些主要模块: 用户定义、角色定义、权限定义、角色指派、权限分配、访问控制核心和角色管理数据. 用户使用系统的时候, 由访问控制核心决定用户的权限, 提供定制的用户界面.

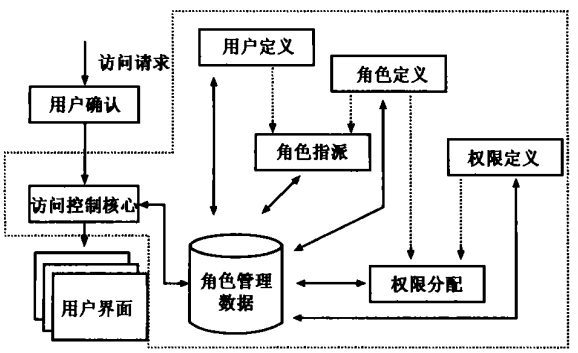


图 3 基于角色控制的系统框架

## 4 功能变化性控制

功能是为达到一个或多个业务目标而作用在对象( 信息) 上的一组操作. 功能是应用系统中易变的部分, 同时也是领域变化性中最不易控制的部分. 功能变化性的原因体现为业务功能需求和系统对外提供的服务之间的差异. 在应用系统中表现为系统功能的增加、修改或者删除. 控制功能变化的机制可以采用标准化的构件接口和集成机制, 以及可动态解释执行的脚本语言等.

### 411 标准化的构件接口和集成机制

系统的功能是由组成系统的构件和构件之间的协作完成的. 标准化的构件接口和灵活的集成机制为控制系统的变化性提供了有效的手段, 这类机制包括插件( Plug2ins) , 以及 CORBA、DCOM、EJB 和 Web Service 等构件互操作标准和规范.

插件是扩展应用系统功能的一种比较好的方式, 开发者或者最终用户按照预先规定的插件接口的要求编写插件, 并动态集成到应用系统中, 为应用系统扩充功能. 插件在部署的时候, 向应用系统登记它的功能和服务. 在需要用到插件所提供的功能的时候, 应用系统会调用相应的插件. 插件大多是与特定的应用相关联的, 例如 Microsoft Internet Explorer 的插件, Netscape Navigator 的插件等.

CORBA、DCOM、EJB、Web Service 等为相应的构件提供了集成机制, 这类机制可以归结为构件集成框架( Com2 ponent Integration Framework, CIF) . CIF 为构件的集成提供了宿主环境, 构件之间一般并不直接相互调用, 而是通过 CIF 来进行. 构件向 CIF 发布它的服务, 而需要服务的构件则向 CIF 询问. CIF 提供了构件的集成机制、对构件进行管理( 构件存储、检索等) 的构件库系统, 以及其他一些公共服务, 如图 4 所示.

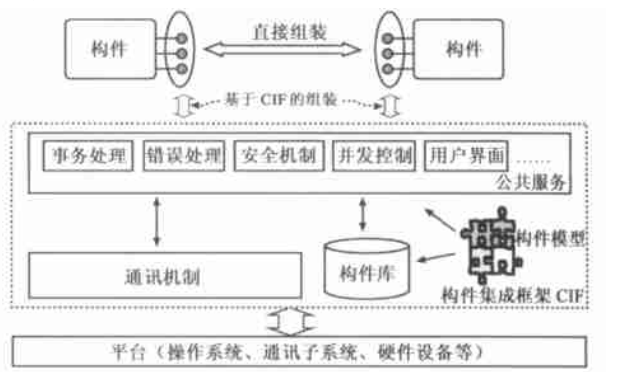


图 4 构件集成框架

CIF 通过构件接口与构件实现的分离, 利用构件动态集成机制, 为应用系统装载/ 卸载不同的构件实现, 从而支持系统功能的变化性. 使用 CIF 的构件库机制, 检索构件以及构件的注册信息, 获取构件的接口和位置信息, 进行构件的动态集成, 支持系统功能的动态演化.

### 412 脚本语言

脚本语言是解释性语言, 通常比高级程序设计语言简单和易于使用. 常见的脚本语言有: JavaScript、VBScript、Perl、MS Office 套件中的 VBA 等, 通常这些语言都是与某些应用一起使用的. 变化性实体是脚本语言书写的代码, 这些代码可以由用户根据需要编写和更改, 由系统解释执行, 支持运行时刻的系统功能绑定, 从而实现了用户编程. 脚本语言的缺点是解释执行效率较低, 处理的数据类型有限, 不适合于构造复杂的系统. 公式是一种特殊的脚本语言, 例如 Microsoft Excel 和 Lotus Notes 中的公式等. 公式机制可以分为公式语法和解释器两个组成部分, 可代替程序代码完成相对简单、有限的操作, 特别是针对有限的数据类型和基于数据库表的操作, 包括数据加工和数据约束.

## 5 数据变化性控制

数据是信息的载体, 是应用系统中最重要的成分之一. 表 1 列举了导致数据变化性产生的主要原因和数据变化性在应用系统中的体现. 其中, 系统中数据结构、内容和格式的变化, 将导致数据库表的变化, 如数据库表项的增、删、改等, 甚至还可能需要对整个数据库进行重构. 尤其在系统投入运行后, 数据库的重构需要进行新库表的构造, 原有数据的迁移等工作, 工作量大且容易出错. 因此, 需要灵活的机制来支持数据的定制, 以适应这种数据的变化性.

可以综合采用以下几种机制来控制数据变化性:

### 511 数据内容和形式相分离

通过将应用系统的数据分为元数据和业务数据, 使得业务数据的形式和内容相分离. 元数据用于对业务数据进行描述, 包括业务数据类型、格式、约束等, 这些信息被用来生成业务数据库的表结构. 通过将业务数据的描述/ 显式化0, 从而支持数据的灵活定制. 例如在商业应用中, 元数据定义了字典、单据、帐簿和报表等业务数据集合的要素, 要素的数据类型、存储精度、显示格式和约束条件, 以及字典、单据、账簿和报表中各数据元素之间的关系等. 依元数据建立的业务数据库, 存放应用系统运行时产生的与业务相关的具体数据, 例如在商业应用中, 业务数据包括在系统运行过程中, 实际产生的商品字典、出库单、入库单、销货小票、库房台帐、销售报表等数据.

### 512 数据和操作相分离

在三层/ 多层体系结构中, 表示层、业务逻辑层和数据层在逻辑上是分离的. 表示层的构件负责与用户的交互, 并调用业务逻辑层的构件来请求应用服务; 应用构件执行业务逻辑, 向数据层构件请求服务; 数据层构件负责与具体的数据库打交道. 数据库可能是不同数据库厂商的产品、文件系统、电子邮件和其他数据存储设备. 通过将数据和业务操作相分离, 使得数据库的变化只会影响到数据层构件, 从而保证数据结构的变化和数据库的更换比较方便.

### 513 数据库中间件

使用数据库中间件解决应用系统可能会使用不同数据库的变化性. 数据库中间件为应用提供一套标准的、简单的、易于理解和使用的数据库接口, 屏蔽分布式数据库的网络通讯协议、数据库访问的细节, 从而适应应用系统中数据的物理存储的变化性. 如图 5 所示.

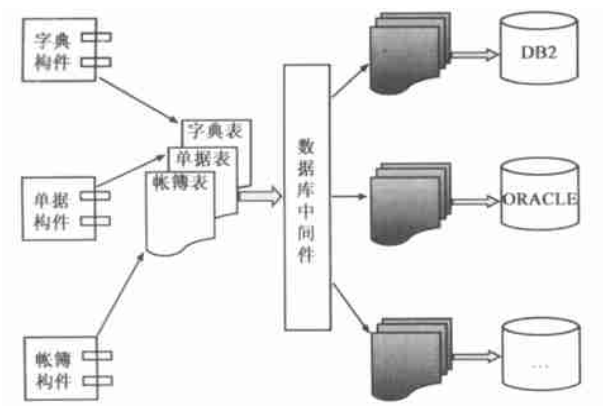


图 5　数据库中间件

标准数据库访问接口是比较常见的数据库中间件, 如 ODBC、OLE DB、ADO、JDBC 等, 提供了通用的数据访问编程模型, 屏蔽了不同物理数据库存储和访问的细节.

### 514 标准数据表示和访问协议 XML

XML 是标准的数据表示和访问协议, 可以解决跨平台的不同数据库之间的数据交换问题, 将一个系统中的某种类型数据库文件用 XML 导出, 然后导入到另外一个系统中的其他类型的数据库中. 例如, 将 Oracle 数据库中的数据发送给 SQL Server 数据库, 如果不使用 XML, 可能中间的接口会非常复杂, 若需要交换数据的数据库种类繁多, 接口的数量将按照指数级增长. 使用 XML, 将一个数据库系统, 如 Oracle 中的数据封装在一个 XML 文件中, 就可以被另外的数据库系统, 如 SQL Server 读入. 由此可见, 标准数据表示和访问协议 XML 是一种较好地处理数据变化的技术.

### 515 公式机制

前面提到的公式机制可用来控制简单数据加工和数据约束的变化性. 使用公式定义一些简单的数据加工, 例如定制系统所需的业务数据由哪些域组成, 以及这些域中数据的来源, 即是由用户输入、查询所得、或者通过一些数据项计算得到, 若是后两种情况, 则定义查询或者计算公式, 执行的时候直接调用公式的计算服务来完成. 此外, 还可以使用公式定义数据之间的约束关系, 例如定制业务数据库表项本身的约束条件、表项之间约束关系等, 从而保证业务数据的一致性和完整性.

## 6 表示变化性控制

系统接口, 特别是用户接口也是应用系统中易变的成分, 这方面的变化性称为表示变化性. 领域中不同系统在处理相同业务时可能采用不同的界面、同一系统中不同业务过程会采用不同界面、同一系统在不同的操作系统平台下也可能要求不同的界面等. 例如, 商业领域应用系统要处理种类繁多的单据、帐簿和报表等表格, 特定系统对单据、帐簿和报表的种类和表现有着各自特殊的需求, 而且随着应用系统的演化, 表格的格式也会不断变化. 随着实现技术的日益进步, 应用的解决方案也在不断的发生变化, 例如用户界面从命令行方式、Win32 的 GUI 方式, 发展到现在的浏览器方式. 表示变化性在应用系统中体现为: ¹ 系统表示方式因为需求的变化而变化; º 运行环境发生了变化. 用户往往要求应用以多种可供选择的、灵活的方式把功能和数据呈现出来. 控制表示变化的机制主要是采取数据表示和数据内容、数据加工相分离的方法, 包括采用三层/ 多层体系结构和用户界面定制等机制.

### 611 三层/ 多层体系结构

一般情况下, 界面只进行与输入和输出数据有关的处理, 而真正的业务逻辑处理都留给业务过程或者功能构件来实施. 这样做实现了业务逻辑层和表示层的分离, 各层的构件可以独立地更新、维护, 而不会影响到其他的部分, 从而移植、维护和升级的工作量大大减少. 业务逻辑层和表示层的分离也为运行环境的改变提供了方便. 只需要修改符合新的环境要求的界面表示层、替换少数的业务构件即可, 大部分可以直接复用, 开发的工作量会有一定程度的减少.

### 612 用户界面定制机制

可视化的用户界面定制是提高人机交互灵活性的重要手段. 系统开发者或用户可以使用可视化编辑器定制各种表单的显示和打印格式, 同脚本语言或公式机制相结合, 可以方便地定义其中的数据加工逻辑和数据约束条件, 从而支持可视化的系统开发和灵活定制, 例如, JetForm 公司的 FormFlow, Actuate 公司的 FormulaOne 等. 此外, 还可以通过提供常用的界面模板, 进一步方便界面定制.

## 7 过程连接变化性控制

功能构件提供了特定领域应用系统所需的大部分功能, 但是这些功能是孤立的, 它们之间没有形成逻辑关系, 在利用这些构件搭建应用系统时, 需要建立这些功能构件的逻辑关系, 形成有逻辑意义的业务过程. 领域中不同应用系统的业务过程会有所不同, 引起过程连接变化的原因有: ¹ 组织机构、人员职责变化导致的业务过程变化; ° 业务发展引起的业务过程本身的改变, 如电子政务领域里, 审批流程的变化等. 在应用系统里就直接表现为系统业务过程连接的变化.

控制过程连接的变化性可以采用工作流引擎.

国际工作流管理联盟提出的工作流参考模型如图 6 所示, 包括以下主要成分: 流程定义工具、工作流引擎、被激活的应用接口、工作流客户端应用和流程监控工具[9]. 工作流引擎是其中的核心部件, 把模型中其他的各个部分连接起来, 为工作流实例提供运行时刻的执行环境. 工作流引擎主要提供以下一些功能: 解释过程定义; 控制过程实例))) 创建、激活、挂起、终止等; 在过程中各个活动之间进行导航; 维持工作流控制数据和相关数据; 提供控制、监控和审计的手段.

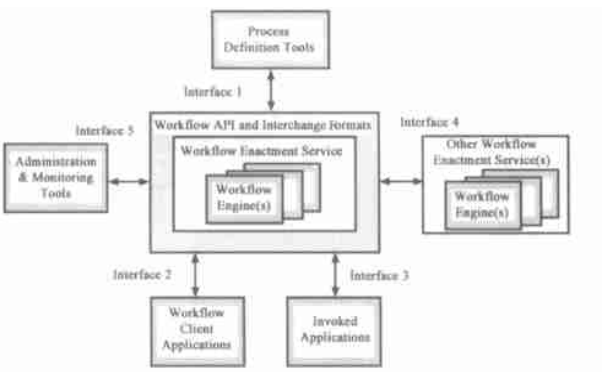


图 6 工作流参考模型

## 8 实例研究))) 商业领域应用软件开发平台

为了对不同商业业态、业务模式下的系统开发提供全面的支持, 在文献[10]中所述的商业领域应用软件开发平台的构造中(如图 7 所示), 实现了上面讨论的变化性控制机制.

### 811 数据

商业业务的基础数据即是保存在字典、单据和帐簿中的信息, 而对于不同业态的应用系统来说, 这些基础数据的组成、约束和形态是不同的. 商业领域开发平台提供的基础构件可以用来灵活定制应用系统的基础数据. 例如, 应用系统开发者通过字典构件为不同的应用系统定义不同的字典, 同类字典在不同系统中可以由不同的要素组成, 而且支持开发者指定字典中各要素的数据类型、存储精度、显示格式和约束条件等属性的定义, 还可以通过定义字典之间的关系来保证字典数据的一致性和完整性. 开发者利用单据构件和帐簿构件来定制系统所需的单据和帐簿由哪些域组成, 以及这些域中数据的来源, 即是由用户输入、或者从某一个字典中查询得到、或者由一些数据项通过计算得到, 若是后两种情况, 则定义查询或者计算的公式, 执行的时候直接调用公式构件的计算服务来完成.

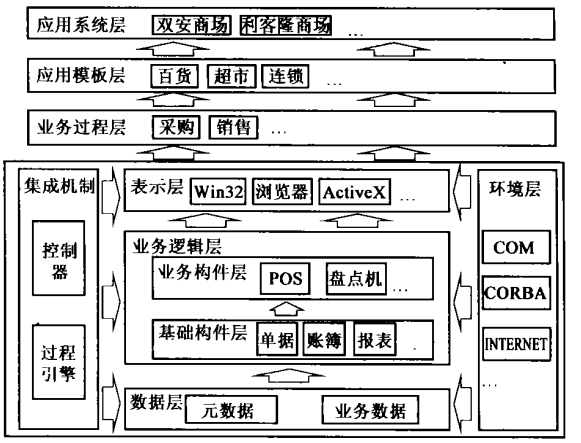


图 7 商业领域应用软件开发平台的体系结构

### 812 功能

在商业软件开发平台中, 用功能构件(包括字典、单据、帐簿、报表和公式等基础构件和 POS 机、盘点机、电子称等业务构件) 和系统功能集成工具(在这里我们称之为/ 控制器0) 来控制系统功能的变化性. 控制器相当于一个构件容器, 提供友好的界面引导应用系统开发人员将功能构件实例化并且集成到应用系统中, 由于构件通过接口对外提供功能, 所以向控制器添加构件就是为应用系统添加功能的过程.

控制器引导应用系统开发者根据自己的实际需要对功能构件进行实例化, 得到字典、单据、帐簿和报表等具体的构件实例, 通过调用构件接口将构件提供的功能集成到应用系统中去. 例如, 应用系统需要提供出库/ 入库的功能, 开发者调用单据构件定制单据的方法, 根据实际需要定制出库单和入库单的内容、格式等, 然后控制器根据定制单据的信息为入库单/ 出库单生成实际的物理表, 并且在应用系统中注册, 通知应用系统该单据的存在, 最后控制器在应用系统的功能菜单上添加入库/ 出库, 并建立入库/ 出库菜单项与单据构件接口之间的对应关系, 这样就完成了出库/ 入库功能的添加.

控制器同样支持由业务构件提供的功能的扩展. 例如某家商场需要使用盘点机清点货物, 则首先开发提供记录盘点情况的业务构件, 然后同样运用上面的方法在应用系统中扩展盘点的功能.

### 813 表示

平台通过两种机制控制表示的变化性. 一方面, 平台支持三层结构应用系统的开发, 使表示层、业务逻辑层和数据层相分离. 另一方面, 通过提供灵活定制单据、帐簿和报表等各种表单界面的功能, 来控制界面的变化性. 应用系统开发者可以使用平台提供的可视化编辑器定制各种表单显示、打印格式, 如域的排列顺序、行宽、列宽、字体、打印时是否分页以及每页的行数等. 此外, 还提供了常用的界面模板, 一般情况下, 应用系统开发者只需对模板进行微小的改动即可适应需求.

### 814 过程连接

平台中的业务构件提供了应用系统所需的大部分功能,

但是这些功能是孤立的, 它们之间没有形成逻辑关系. 例如, 被单据构件实例化的入库单提供了入库的功能, 被帐簿构件实例化的库存帐提供了记帐的功能, 但入库单的录入无法激发库存帐记帐的功能. 工作流引擎将支持系统开发者定制这样的业务逻辑.

平台中工作流引擎的实现采用了微软 COM+ 技术提供的事件服务和队列机制. 工作流引擎作为灵活定制业务逻辑的机制, 需要知道在某一事件发生后如何通知对该事件感兴趣的构件. COM+ 技术提供了事件服务来解决这个问题. 在事件机制中, 给其他构件提供变化通知的构件称为发布者, 而从发布者那里接收并处理请求的构件称为订阅者. COM+ 事件服务用于处理发布者和订阅者之间的匹配和连接, 解决了发布者与订阅者之间的通讯问题. COM+ 队列构件(Queue Component, 简称 QC) 解决构件之间调用的异步传输.

工作流引擎通过将事件服务与队列构件结合起来支持业务流程的灵活定制, 结构如图 8 所示. 当两者结合在一起使用时, 将事件类本身设置为队列构件, 发布者使用队列名字创建事件对象(3), 当发布者向订阅者激发事件(4), 而且事件服务找到所有订阅了此接口和方法的订阅者(5) 后, 事件服务连接到 QC 记录器而不是连接到实际的订阅者(6), 所有对事件接口的调用都被记录下来. 当订阅者连接到网络上开始运行时, 所有被记录下来的、对事件类的调用都被重新发送回事件机器(7), 并且在事件机器上重新播放出来(8), 这时候事件才被真正激发到订阅者处.

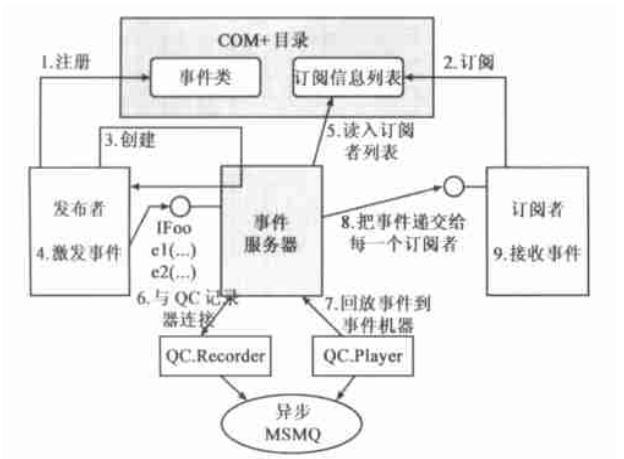


图 8　工作流引擎的一种实现

业务逻辑定制者告诉系统哪些事件需要发布、哪些构件和构件的哪些方法接口对事件感兴趣, 并建立它们之间的对应关系, 至于如何发布事件、如何激活接口和方法以及它们之间的异步通讯问题都由工作流引擎来解决, 这不仅解决了支持开发者对业务逻辑灵活定制的问题, 而且使系统开发者的主要精力集中在业务逻辑上而不需要考虑如何实现的问题.

## 9　结论

一个应用系统只有能够适应不断变化的需求, 才能具有很强的生命力, 因此变化性的研究具有重要的理论价值和现实意义. 本文提出了一个开放的变化性控制框架, 从组织机构、功能、数据、表示和过程连接等五个维度讨论了变化性出现的根源、变化性表现形式、变化性控制机制和实现技术, 提供了一个系统化的变化性解决方案. 随着软件技术的发展和人们对变化性研究的不断深入, 新的控制机制和实现技术将不断出现, 并进一步充实该框架的内容.

## 参考文献:

[1] Sharp D C. Containing and facilitating change via object oriented tailoring techniques[A]. Proceedings of The First Software Product Line Conference[C]. Denver, Colorado, 2000. 141-174.

[2] Gurp J Van, Bosch J, Svahnberg M. On the notion of variability in software product lines[A]. Working IEEE/IFIP Conference on Software Architecture[C]. Amsterdam, Netherlands, 2001. 45-54.

[3] Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C V, Loingtier J M, Irwin J. Aspect2oriented programming[A]. Proceedings of the European Conference on Object2Oriented Programming (ECOOP)[C]. Finland: Springer2Verlag LNCS 1241, 1997.

[4] Homepage of the Subject2oriented Research Project[DB/OL]. IBM Thomas J Watson Research Center, Yorktown Heights, NY, http://www.research.ibm.com/sop.

[5] Homepage of Oxford University Computing Laboratory[DB/OL]. Programming Tools Group, Intentional Programming Project, http://web.comlab.ox.ac.uk/oucl/research/areas/progtools/intentional.htm.

[6] Mira Mezini. Variational Object Oriented Programming[D]. Germany: University of Siegen, 1997. 39-42.

[7] Scheer A W. Aris2Business Process Frameworks 3rd edition[M]. New York: Springer Verlag, 2000.

[8] W M P van der Aalst, A H M ter Hofstede, B Kiepuszewski, A P Barros. Advanced workflow patterns[A]. O Etzion, P Scheuermann. 7th International Conference on Cooperative Information Systems (CoopIS 2000)[C]. Volume 1901 of Lecture Notes in Computer Science. Berlin: Springer2Verlag, 2000. 18-29.

[9] Workflow Management Coalition, The Workflow Reference Model[S]. 1995. http://www.wfmc.org.

[10] 陈兆良, 张世琨, 王立福. 基于构件的商业领域软件开发平台的构造[J]. 软件学报, 2002, 13(9): 50-57.

## 作者简介:

张世琨　男, 1969 年 10 月生于河北省沙河市, 于 2000 年 7 月在北京大学获理学博士学位, 现为北京大学信息科学技术学院副教授, 主要研究领域: 软件工程、软件体系结构和工作流技术.

胡文蕙　女, 1977 年 6 月生于山西省五寨县, 于 2001 年在北京大学获理学硕士学位, 现为北京大学信息科学技术学院博士生, 主要研究领域: 软件工程、软件框架和语义网.