基于目标树的领域建模及映射技术

焦 锋

(北京科技大学信息工程学院 北京 100083) (河北工程大学信息与电气工程学院 邯郸 056038)

摘 要 针对工程领域业务过程复杂、需求易变、数据分布、异构等特点,实现不同时期开发、采用不同技术、具有不同功能的领域应用软件集成,具有重要实际意义和研究价值。基于面向服务架构和模型驱动技术,结合虚拟石油钻井平台的设计,提出了基于目标树的领域建模技术和基于SCA集成规范的抽象服务模型及映射技术。石油钻井工程领域的建模实例表明了本建模技术能够快速、方便地适应业务扩展和技术变化的需要。

关键词 领域模型,服务模型,业务构件

中图法分类号 TP311 **文献标识码** A

Domain Modeling and Mapping Technique Based on Goal-tree

JIAO Feng

(School of Information Engineering, University of Science and Technology Beijing, Beijing 100083, China)

(School of Information Engineering, Hebei University of Engineering, Handan 056038, China)

Abstract Domain-specific software is characterized with heterogeneous data and resources, complicated business processes and changing business requirements from users. It is of practical significance and great value to find out a technique for integrating these software which may be developed by different people using different languages and technologies, and provides very different functionalities. Considering the characteristics of oil-drilling engineering, a goal-tree based domain-specific modeling technique and a SCA based platform independent modeling technique were suggested, which SOA and MDA are used for reference to. Finally, a real case, Simulation System of Oil-Drilling Engineering, shows the modeling techniques are practical to integrate domain-specific software and can meet the requirements to flexible extension and configuration of business modules.

Keywords Domain model, Service-oriented model, Business component

领域应用软件提高了诸如石油钻井等工程领域的现代化管理和生产水平。但随着领域业务不断深化和软件技术不断更新,必然需要不同时期、不同功能软件的综合集成。如何实现一系列功能重叠、资源重复的孤立系统的集成和演化,是挑战性问题。

钻井工程领域的应用集成问题主要为:

(1)业务可扩展性和灵活性。业务复杂性和易变性是石油钻井等工程领域的特点,随着自身工艺和信息技术的发展,必将不断产生新业务和改进已有业务。这些升级改造后的业务模块需要能够平滑加入到系统平台,所以需要提供业务可扩展性和灵活性。

(2)模块互操作性和兼容性。随着在工程领域软件应用地域的扩大,软件规模和复杂性增加,需要提供开放和可灵活配置的技术框架来实现异构模块互操作。

(3)界面可配置性和可用性。工程领域有大量人机交互功能,因此实现可配置和易操作的界面能够增加用户体验,提高系统可用性。

应用集成问题比较复杂,集成方法多种多样。按集成要素可分为业务集成、数据集成、控制集成、界面集成等多个方面;按集成层次可分为业务、业务过程、应用、技术、中间件等多个层次[1]。解决应用集成问题需要从技术层次和业务层次两方面考虑。

(1)从技术方面,需要提供集成异构平台的能力,以实现异构平台互通和信息共享。典型的技术有CORBA,DCOM,RMI等,其中Web Service具有较强的平台无关性,已经成为应用集成的主要技术平台,文献[2,3]提出使用Web Service技术建立一种灵活可伸缩的软件体系结构。

(2)从业务方面,应用集成关注离散的业务过程和功能的集成,通过领域模型整合新系统和遗留系统。典型的集成方法就是模型驱动方法及其变种。文献[4]提出了分层的模型驱动方法,即建立对应于表现层、业务层、数据层的3个模型及其相互关系,并各自从抽象到具体得到最终目标系统。方法细化了模型,提高了目标系统的可扩展性和灵活性。文献[5]针对智能油田领域给出了基于构件的模型驱动应用集成框架。

目前基于模型驱动的架构(MDA)和面向服务的架构

到稿日期:2010-05-15 返修日期:2010-11-12 本文受国家科技支撑计划项目(2006038027015),国家高技术研究发展计划(863)(2008AA01Z109),国家自然基金(61075053)资助。

焦 锋(1970—),男,博士生,主要研究方向为领域软件工程与数据工程等,E-mail:jiaof70@163.com。

(SOA)的企业应用集成已成为研究热点。MDA是OMG(Object Management Group)组织提出的一种模型驱动的软件开发规范,广泛用于实现企业模型的集成和软件开发。它分离了业务功能和实现细节,提供了对应于不同抽象层次的多个模型及其转换规则,使得领域业务模型等高层抽象模型可以重用。SOA是一种用于建立松耦合软件的架构风格,它提出将一系列粗粒度的业务功能封装成具有标准接口的服务,企业通过调用并装配服务实现应用集成。MDA和SOA的结合可以更好地解决应用集成问题。文献[6]给出了基于企业Web Service的模型驱动开发框架,文献[7]则提出了自己的服务概念及相应的服务模型。

服务组件架构(Service Component Architecture,SCA)是针对SOA提供的具体实施标准,是面向服务的集成规范,正被广泛采用和推广。

另一方面,MDA的业务和实现分离建模的思想已被广泛采用且可以提供业务扩展性和灵活性。但是,MDA自身提供的通用建模方法不能完全满足石油钻井领域应用集成的需要,同时MDA没有提供针对SCA的建模工具。

本文根据MDA建模思想并利用SCA集成规范提出了一种针对石油钻井工程领域的应用集成方法,用于解决诸如石油、地质等相似领域的应用集成问题。本文针对钻井工程领域,提出了基于目标树的领域建模方法建立高层的领域模型DSM(Domain-Specific Model for oil-dilling),即将施工目标逐层分解并包装为一系列特殊的对象,形成基于对象节点的目标树领域模型,通过目标树模型实现业务解耦合,便于业务扩展和调整;针对系统实现,提出了基于SCA规范的建模方法建立平台无关模型ASM(Abstract Service Model based on SCA),即依据SCA模型建立抽象的实现模型,用于将领域模型映射到目标系统,SCA的开放性和配置灵活性有利于屏蔽底层实现,集成异构平台,同时与领域模型的相似性有利于保持业务功能的完整性。

本文第1节结合石油钻井工程给出了领域模型的定义、建模过程和描述手段;第2节提出了基于SCA的实现模型,给出了定义、映射规则和描述手段。

1 领域模型

1.1 模型定义

领域模型(Domain-specific Model,DSM)由一系列工程对象组成,其元模型为图1所示的一个类层次图。DSM是一个二元组,表示为$DSM=\{E,R\}$,其中

(1)E是工程对象的集合。有3种不同粒度的工程对象:主对象(MasterObject)、设计对象(DesignObject)、原子对象(AtomicObject)。

MasterObject是钻井工程领域内的主要实施对象,主对象只有一个。例如石油钻井工程领域的所有工程活动都是围绕井眼进行的,那么井眼就是主对象。所有业务过程和业务状态都是主对象的内部组成元素。

DesignObject是各个工程过程实施的目标对象,由主对象分解得到。如石油钻井工程领域的主对象(即井眼)可分解为井眼轨迹、井身结构、钻具组合等多个设计目标,每一个目标就是一个设计对象。

AtomicObject是模型中粒度最小的工程对象,由设计对象分解得到。作为工程过程中工程活动的实施目标,原子对象具有不可再分性。

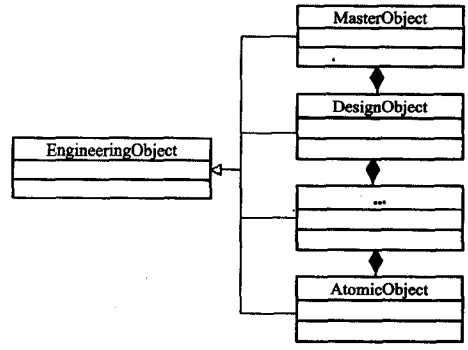

图1 领域元模型

工程对象表示为$O::=\langle oid,layer,P^S,S^S,desc\rangle$,其中

oid为对象的唯一标识符;

$layer$为对象的层级,最高层为MasterObject,最低层为AtomicObject;

P^S是属性对象集合;

S^S是对象的服务集合;

$desc$是工程对象的描述。

P^S中的属性对象p是与工程对象有关的属性,描述对象的静态结构,表示为$p::=\langle did,t_d,A^s,desc\rangle$,其中

did为属性对象的唯一标识符;

t_d为属性对象类型,$t_d::=\langle private|\ public\rangle$,分别表示私有或公开;

A^s为属性对象的内部特性集合,使用关系模式描述;

$desc$为属性对象的描述。

S^S中的服务s为工程对象的行为描述,是具有一定业务逻辑的实现,表示为$s::=\langle sid,t_s,R^s,i^{fs},imp^s,exp^s,desc\rangle$,其中

sid为服务的唯一标识符;

t_s为服务类型,$t_s::=\langle private|public|humantask\rangle$,即私有服务、公开服务、人工任务;

R^s为业务规则集合;

i^{fs}为需要外界提供的服务集合,集合中的元素表示为$oid::sid$(对象标识::服务标识);

imp^s为需要的属性对象集合,集合中的元素表示为$oid::did$(对象标识::属性对象标识);

exp^s为输出的属性对象集,表示与imp^s相同;

$desc$为服务的描述。

(2)R为工程对象的关系集合。工程对象之间的关系是组合和依赖。上层对象是下层的父对象,下层对象是上层的子对象。组合有两种表示方式:$hasChild(x,y)$和$hasParent(y,z)$,分别表示x是y的子对象,x是z的父对象。同一个父对象的子对象之间是依赖关系。与传统对象概念不同,工程对象除具有封装的特点外,不存在继承和多态,所以结构复杂性较低,容易替换和扩展。

模型中对象之间的调用关系如图2所示,模型的性质为:

(1)如果$hasChild(O_n,O_m)$,则O_m中的服务允许访问父对象O_n的公开属性对象;同时O_n可以调用O_m的公开服务。

(2)如果O_n和O_m同层且同属一个父对象,则O_n可以调用O_m的公开服务。

(3)完备性:如果主对象的服务满足系统目标,则继续判断设计对象的服务是否满足主对象的服务需求,依次逐层递归判断,直至最后的判断都为真值,则说明模型是完备的,否则需要补充功能。

(4)相交性:两个子对象的 imp^s 的交集即为相交性。如果相交性大于设定的阈值,则两对象的联系过于紧密,应考虑分解父对象的属性对象或合并两个子对象。

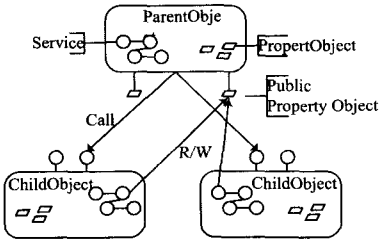

图 2 父子对象之间的关系

DSM 的特性分析:

DSM 是问题空间的客观抽象,符合人们认识工程问题的客观规律,具有较强的稳定性和灵活性。

(1)钻井工程领域元模型不会随着业务扩展和改变而变化,保持了较强的稳定性;

(2)工程对象是业务概念的抽象和对象化,封装了业务数据(即属性)和业务过程(即操作)。因此,业务扩展和改变可映射为领域模型中相应工程对象的增加、更新、替换。工程对象的独立性、封闭性等对象特性及粒度划分原则使变化限制在对象范围内,因此模型具有较强的灵活性。

1.2 建模过程

钻井工程领域模型的建模过程主要由初步建立目标树、确定工程对象的属性、确定工程对象的服务、优化模型 4 步组成,具体描述如下:

(1)初步建立目标树。分析问题域,识别各级工程对象,建立树状结构。用类 java 伪代码(java-like pseudocode)描述为:

```
class EngieeringObject{   //工程对象类的定义
    private string name;   //工程对象的名称
    private short layer;   //工程对象的层级
    private Set<string> operations;//业务功能
    private Set <Service> service;//工程对象的服务
    private Set <PropertyObject> propertyobject;
    private Set <string> childname;//子对象的名字
    private Set <EngieeringObject> childobject;//子对象集合
    …}
```

其中,类 EngieeringObject 的属性 operations 表示诸如过程、活动、步骤等业务功能,用于获取标识工程对象的 property-objects 和 services。

表 1 类 ObjectFactory 中的各类方法说明

关键词	说明
create	创建一个工程对象
findOperation	查找业务功能,即 business operations
findPropertyObject	从 operations 中标识属性对象 PropertyObject
classifying	分类 property-objects
identifying	标识子对象

如表 1 所列,类 ObjectFactory 用于生成工程对象,主要包括 4 个步骤:针对业务目标找出业务功能,即 operations;从 operations 中识别属性对象 property-objects 并作为候选;归类这些属性对象,并标识出工程对象;继续标识子对象。

```
class ObjectFactory(){
    public ObjectFactory(){}
    public void create(string objectname,short layer){
    EngieeringObject eo=new EngieeringObject();
    eo. setName(objectname);
    eo. setLayer(layer);
    Set(string) ops=findOperation(eo);
    eo. setOperation(ops);
    Set (PropertyObject) pos=new Set (PropertyObject)();
    Iterate iter=ops. iterate()
    While(ops. hasNext()){
      PropertyObject po =
                    findPropertyObject(ops. get(iter. next()));
      pos. add(po);}
      eo. setPropertyObject(pos);
    //标识子对象名字
    Set(string) childobjectname =new Set(string)();
    childobjectname=identifying(classfying(pos));
    eo. setChildName(childobjectname);
    return eo;}
}

class DSMStructBuilding{ //生成目标树
  public static void main(string[] args)
  { ObjectFactory objectFactory=new ObjectFactory();
    short layer=0;
    //标识主对象
    EngieeringObject mo=
                    objectFactory. create("masterobject",layer);
  //递归调用,依次标识每一层的工程对象
  nodeBuilding(mo,layer);
//依次检查每一次层中对象划分是否正确,并滤掉重复对象
    filterDSMStruct(mo);  }
  public static void nodeBuilding(EngieeringObject eo,short layer){
    short ls_layer =layer+1;
    Set(string) coname=eo. getChildName();
    if(null == coname or Layer. AtomicLevel == ls_layer)
        return;
    Iterate iter=new eo. iterate();
      While(iter. hasNext()){
        //创建下一级工程对象
        EngieeringObject ceo=
          objectFactory. create(iter. get(iter. next()),ls_layer);
        eo. setChildObject(ceo);//建立对象间的父子关系
        nodeBuilding(ceo,ls_layer);}
    return;}
```

以石油钻井工程中的井身结构为例,说明其树状结构的建立过程。

找出所有针对井身结构的操作。

将操作中的信息字段列为候选的属性对象:

岩性信息、压力梯度曲线、必封段信息、套管个数、套管直径、钻头直径、套管长度、井身设计系数、水泥封固段、设计报告。

分析归纳属性对象：

地质信息(岩性信息、压力梯度曲线、必封段信息)、套管个数和长度(套管个数、套管长度)、设计系数、水泥封固段、套管和钻头直径、设计报告。

谁	操作	信息
设计人员	查看	岩性信息
	查看	压力梯度曲线
	查看	必封段信息
	确定	套管个数
	确定	套管类型和长度
	确定	套管和钻头直径
	给出	井身设计系数
系统进行	安全	校核套管层次
系统	计算	水泥封固段
设计人员	绘制	井身结构图
设计人员	编写	设计报告
……		

确定子对象：

分析可知，地质信息、设计系数、设计报告显然不属于井身结构的设计目标，仅是辅助信息，因此不应列为子对象。经过过滤可得井身结构的子对象为套管个数和长度、水泥封固段、套管和钻头直径。

(2)确定工程对象的属性。经过步骤(1)已得到初步的属性对象集合。依次获取每一层对象的属性对象集合，检查属性的拥有者是否合理，调整不合理的属性，直至满足需求。

(3)确定工程对象的服务。分析各级对象，找出各自的服务，并划分公开和私有类型。

遍历所有工程对象，并做如下步骤：

获取对象的操作集合 $O.opreation^s$；

从操作集合中抽取由外部对象提供的操作，并分类作为公开服务，存入临时集合 LS 中；

从操作集合中抽取由外部对象提供的属性对象，并分类存入临时集合 LP 中；

遍历 LS，找到元素所属的工程对象，并为该对象建立对应的公开服务；

遍历 LP，找到元素所属的工程对象，并将其对应的属性对象类型设为公开；

对象中剩余的操作分类归纳为私有服务，剩余的属性归纳为私有属性；

将操作集合 $O.opreation^s$ 中的操作分类划分到各服务的规则集合中，即获取 $O.S.R^s$。

(4)完善对象的服务信息。遍历所有服务，从所属的规则集合中，抽取服务的组成元素 if^s，imp^s，exp^s。

(5)检查对象及对象关系的相交性和完备性，补充完善模型。

(6)转步骤(2)迭代，直至满足相交性和完备性的最优解。

以工程对象"套管个数和长度"为例，经过步骤(2)－步骤(6)后可以得到对象的完整信息。

对象标识：WellBody(井身结构)

层级：1

属性对象：

WellBodyInfo(套管信息)＝{…}

服务：

服务标识：getBodyInfo(获取井身的信息)

服务类型：public

……

服务标识：compuCasingNum&Length(计算套管个数和长度)

服务类型：private

需要提供的属性对象：WellEye. WellEyeObject(井眼标识)，PressGradient(压力梯度数据)

输出的属性对象：CasingLengthList，WellBodyInfo

需要提供的服务：WellTrail. getPiecewisePoint，CasingNum-Length. safetyCheck()

对象标识：CasingNumLength(套管个数和长度)

层级：2

属性对象：

CasingInfo(套管长度信息)＝{编号，套管名称，起始深度，终止深度，井深，垂深，类型}

CasingLengthList(套管长度信息集合)＝{CasingInfo}

DesignParam(设计参数)＝{…}

服务：

服务标识：safetyCheck(安全校核)

服务类型：private

需要提供的属性对象：WellBody. WellEyeObject(井眼标识)，DesignParam，WellBody. PressGradient(压力梯度数据)

输出的属性对象：CasingLengthList

需要提供的服务：WellTrail. getPiecewisePoint()

服务标识：getDesignParam(获取设计参数)

服务类型：private

需要提供的属性对象：WellBody. WellEye(井眼标识)

输出的属性对象：DesignParam

需要提供的服务：propFactory. create(调用属性工厂的服务创建设计参数对象)

服务标识：setDesignParam(设置参数)

服务类型：humantask

需要提供的属性对象：WellBody. WellEye(井眼标识)

输出的属性对象：DesignParam

需要提供的服务：getDesignParam

Human. formDesignParam(人工填写表单)

……

2 抽象服务模型

SCA 的核心思想是构造一系列不同粒度的服务，并由服务组成系统，强调松耦合，注重重用性和扩展性。工程对象是业务功能包装体，同 SCA 的服务构件一样，也具有独立性、封装性、模块化等特征，便于转换和映射，有助于保持业务概念一致性。基于 SCA 规范，本文针对领域模型提出了平台无关模型，即 ASM。ASM 是一个抽象的实现模型，是领域模型和实现系统之间的中间模型。

2.1 模型定义

抽象服务模型(ASM)由用户界面层(UI Layer)、服务层(Service Layer)两个逻辑层组成。如图 3 所示，用户界面层 UI Layer 包含一系列客户组件(ClientComponent)，负责与服务层交互和界面显示；服务层 Service Layer 包含一系列构件(Composite，即复合组件)，负责响应 Client Component 的请求和业务逻辑处理。在服务层中有一类特殊的构件，称为数据构件(Data Composite)，负责处理来自其他构件的数据请求及数据对象的持久化。服务层的模型也是一个类层次图，由域(Domain)、构件(Composite)和组件(Component) 3 种元素组成。

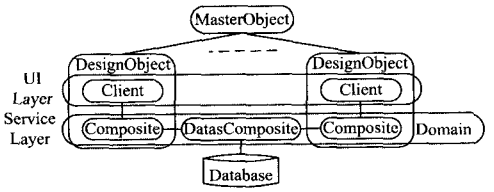

图 3　DSM 和 ASM 之间的关系

定义 1(数据对象,DataObject)　是 ASM 中一种特殊的对象,是数据及其结构的封装体。数据对象中的数据类型分为简单类型和对象类型。

定义 2(组件,Component)　是 ASM 的基本组成元素,表示为 Component::=〈name, interface, reference, dataobject, implementation〉。其中,

name 为服务组件惟一标识名;

interface 为接口,包含对外提供的服务,表示为 interface::=〈name, op^s, *desc*〉。其中 op^s 为操作集合,表示为 *op*(name, imp^s, exp^s),name 为 interface 的惟一标识名,imp^s 为输入的数据对象集合,exp^s 为输出的数据对象;

reference 为需要提供的服务,表示为 reference::=〈name, target〉。其中 name 为 reference 的惟一标识名,target 为其他 Component 的 interface 名;

dataobject 为数据对象集合,表示为〈name, A^s, *desc*〉。其中 name 为 dataobject 的惟一标识名,A^s 为数据对象的内部特性集合,使用关系模式描述;

implementation 为内部逻辑的实现,表示为 implementation::=〈L^s〉。其中 L^s 是实现模块的集合。

定义 3(构件,Composite)　是 Component 的复合体,表示为 composite::=〈name, component, interface, reference, dataobject〉。其中 component 为原子组件集合,其余元素含义与 component 相同。

定义 4(客户组件,ClientComponent)　是用户界面的组成单元,表示为 ClientComponent::=〈name, UI^S, reference, implemetation〉。其中 name 为客户组件惟一标识名;UI^S 为用户界面组成元素的集合;reference 和 implementation 的定义同 Component。

ASM 的特性分析:

如图 3 所示,ASM 对领域模型实施了横向切割,划分出 ClientComponent 层、Service 层,分别封装了客户端功能、业务逻辑功能和数据功能。同时,ASM 划分出多个逻辑模块(由 ClientComponent 和 Composite 组成),对系统实施纵向切割,每个逻辑模块对应设计对象 DesignObject。

(1)ASM 与 DSM 的相似性保证了业务在不同阶段的一致性和完整性,同时 ASM 的开放性保证了模块兼容性和互操作性。

(2)领域需要大量的人机交互功能,然而 SCA 仅考虑了业务层的集成和重用,未考虑用户界面层。稳重的 ASM 模型兼顾了业务层和用户界面层(ClientComponent)两部分,保持了应用系统的完整性。

(3)遗留系统的相应功能被包装成 Component 或 Compsite 集成到目标系统中。

2.2　从 DSM 到 ASM 的映射

对象约束语言 OCL 是 OMG 组织(对象管理组织)制订的 UML 语言(统一建模语言)规范的一部分,也被用于描述转换规则。本文对其进行简单的扩充,用于描述从 DSM 到 ASM 的映射规则。OCL 扩充说明如表 2 所列。

表 2　OCL2.0 扩充说明

关键词	说明	格式
transrule	定义一个规则	transrule〈identifier〉{statement}
source	定义源对象,即转换前的对象	source 〈identifier〉:〈scope〉::〈elmenet〉
destination	定义目标对象,即转换后的对象	destination〈identifier〉:〈scope〉::〈elmenet〉
mapping	一系列转换语句	Mapping:statement referrule
referrule	调用一个规则	(〈sourceobject〉,〈destinationobject〉,〈transrule〉)
new	实例化一个对象	new〈objecttype〉
for	定义一个循环	For (OCLexpress;OCLexpress;OCLexpress){Statement}
scope	返回所属对象或父对象	〈servicename〉)scope|〈objectname〉.scope

从 DSM 到 ASM 的映射由 3 部分组成:从 AtomicObject 到 Component 的映射、从 DesignObject 到 Compsoite 的映射、ClientComponent 的生成。映射关系如图 4 所示。

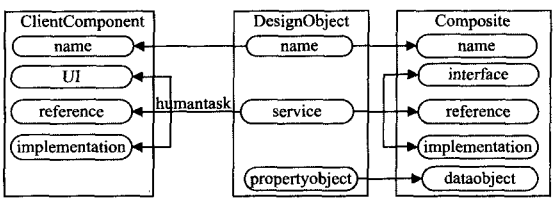

图 4　DSM 到 ASM 的映射

(1)Mapping from AtomicObject toComponent(从原子对象到服务组件的映射)

```
transrule adoToac{
  source ado:DSM::AtomicObject;
  destination ac:ASM::Component;
  mapping:
    let ac=new ASM::Component;//实例化一个新 Component
    let ac. name=ado. name;
    let ss:set(ado. service)->select(type <> "humantask");
    let num:integer=ss. size();  let ac_s:integer=1;
    let ac_i:integer=1;  let ac_r:integer=1;
    //映射每一个服务
    let ac. interface= new ASM::Component. interface;
    let ac. interface. name=ac. name+"interface"
    for(let i:integer=1;i<=num;let i++){
        //映射 interface
      if ss(i). type="public" then
        let ac. interface. op(ac_s)=new ac. interface. op;
        referrule(ss(i), ac. interface. op(ac_s++), serviceTointerface);
      endif
      //映射 implementation
      let ac. implementation(ac_i)=new ASM::Component. implementation;
      referrule(ss(i), ac. implementation (ac_i++), serviceToimplementation);
      //映射 reference,ope 表示服务所属对象
      let ss_rs:set(ss(i). referserv->select(scope <> ado));
      let rsnum:integer=ss_rs. size();
      for(let j:integer=1;j<=rsnum;let j++){
        if not ac. refernce->exists(ss_rs(i)) and
```

```
            ss_rs(i). type="public" and ss_rs(i). exists then
        let ac. reference(ac_r)=new ASM::component. refe-
                                    rence;
        let ac. reference(ac_r). name=ss_rs(j). scope+"inter-
                                    face";
        let ac. reference(ac_r). target=ac. reference(ac_r). name;
      endif}
  }
  //映射每一个属性对象
  let ps:set(ado. property);let num:integer=ps. size();
  for(let i:integer=1;i<=num;let i++){
    if not ac. dataobject. exits(ps(i)) then
      let ac. dataobject(ac_p)=new ASM::Component. dataob-
                                    ject;
      let ac. dataobject(ac_p++)=ps(i);
    endif}
  }
  //工程对象的服务到构件服务的映射
  transrule serviceTointerface{
    source s1:DSM::service;
    target o1:ASM::interface. op;
    mapping:
      let o1. name=s1. name;
      let o1. imp. name=s1. impprop. name;
      let o1. exp. name=s1. expprop. name;
  }
  //工程对象的服务到构件内部逻辑的映射
  transrule serviceToimplementation{
    source s1:DSM::service;
    target i1:ASM::implementation;
    mapping:
      let i1. name=s1. name;
      let i1. imp. name=s1. impprop. name;
      let i1. exp. name=s1. expprop. name;
      let i1. module=s1. processrule;
  }
```

(2)从 DesignObject 到 Composite 的映射

```
transrule fsdoTofcc{
  source fsdo:DSM::DesignObject;
  destination fcc:ASM::Composite;
  mapping:
    let fcc=new ASM::Composite;let fcc. name=fsdo. name;
    let sa:set(fsdo. atomicobject);let num:integer=sa. size();
    for(let i:integer=1;j<=num;let j++){
      let fcc. component=new ASM::Compoonent;
      let fcc. component. name=sa(i). name;
      let fcc. component. refer=sa(i)+". xml";
      }
      //其余元素的映射同 transrule adoToac
}
```

(3)生成用户端组件

对每一个 designobject 使用规则 serviceTocliencomponent

```
transrule serviceTocliencomponent{
  source s1:DSM::DesignObject:service;
  destination c1:ASM::clientcomponent;
  mapping:
    let s1=s1->select(type="humantask");
    if s1. exists() then
      let c1=new ASM::clientcomponent;
      let c1. name =s1. scope. name+"Client";
      for(s1 中的每一个元素){
        //processrule 映射规则同
              adoTocomponenet 中的 processrule 映射部分
      //reference 映射规则同
              adoTocomponenet 中的 reference 映射部分
      //映射输入的属性对象
      let ps:set(c1. impprop)->select(property. scope="people");
      let num:integer=ps. size();
      for(let i:integer=1;i<=num;let i++){
          //增加界面元素和实现 }
    }
  endif
}
```

结束语 本文给出了基于 MDA 思想和 SCA 规范的集成方法。MDA 通过分离关注点,建立不同抽象层次的模型,有助于高层模型的重用。SCA 集成规范的标准化和开放性有利于集成异构平台,同时解决了领域模型和实现模型的失配问题。模型在虚拟石油钻井平台中得到了实际应用。应用表明,模型高度适应领域业务扩展和异构模块互操作性的需求,提升了领域建模和应用集成能力。通过虚拟石油钻井平台的案例也为诸如地质、煤炭等具有相似特点的领域提供了一些实例参考。但本文中的领域模型的普遍性尚需更多的实践,同时增强和完善模型及其支撑工具,也将是研究组进一步的工作。

参 考 文 献

[1] Mosawi A A, Zhao Li-ping, Macaulay L. A Model Driven Architecture for Enterprise Application Integration[C]// Proceedings of the 39th Hawaii International Conference on System Sciences. Hawaii: IEEE Society Press, 2006: 181c

[2] Bagheri O R, Nasiri R, Peyravi M H, et al. Toward an elastic service based framework for Enterprise Application[C]// Fifth International Conference on Software Engineering Research. Management and Applications Integration Busan. Korea, 2007: 711-719

[3] Harikumar K, Lee A, Chia-Chu Chiang R, et al. An Event Driven Architecture for Application Integration Using Web Services[C]// IRI-2005 IEEE International Conference on Information Reuse and Integration. Las Vegas, Nevada, USA, 2005: 542-547

[4] Moreno N, Vallecillo A. A Model-based Approach for Integrating Third Party Systems with Web Applications[C]// Proc. the 5th International Conference on Web Engineering, LNCS 3579. 2005: 441-452

[5] Zhang Cong, Bakshi A, Prasanna V, et al. Towards a Model-based Application Integration Framework for Smart Oilfields [C]// IEEE International Conference on Information Reuse and Integration. Hawaii: IEEE Society Press, 2006: 545-550

[6] Yu Xiao-feng, Hu Jun, Zhang Yan, et al. A Model Driven Development Framework for Enterprise WebServices [C] // 10th IEEE International Enterprise Distributed Object Computing Conference(EDOC'06). Hong Kong: IEEE Society Press, 2006: 75-24

[7] Belhajjame K, Embury S M. A Model-driven Approach to Service Composition in Virtual Enterprises[C]// IEEE International Conference on Services Computing. Salt Lake City, Utah, USA, 2006: 214-221