ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.7, July 2007, pp.1612−1625
DOI: 10.1360/jos181612

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于 MDA 的 TRISO-Model 模型管理方法及应用*

袁 峰[1,2], 李明树[1,3+]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100080)

[2](中国科学院 研究生院,北京 100049)

[3](中国科学院 软件研究所 计算机科学国家重点实验室,北京 100080)

MDA-Based Model Management Method and Its Application for TRISO-Model

YUAN Feng[1,2], LI Ming-Shu[1,3+]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[3](State Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62661001, E-mail: mingshu@admin.iscas.ac.cn, http://www.itechs.com.cn

Yuan F, Li MS. MDA-Based model management method and its application for TRISO-model. *Journal of Software*, 2007,18(7):1612−1625. http://www.jos.org.cn/1000-9825/18/1612.htm

Abstract: TRISO-Model (tridimensional integrated software development model) is a 3-D integrated software engineering methodology proposed to deal with the problems caused by the increasing complexity and dynamic in current software development. In TRISO-model, to maintain the semantic consistency among multiple models and reuse the common model operations, there is the need for semantic-consistent model management. This paper proposes an MDA-based model management method: MDA-MMMethod (MDA based model management method). Based on the common semantic definition—MOF (meta object facility), the implementations of MDA (model driven architecture) standards can be reused in the development of model applications in TRISO-model. The corresponding supporting system, MDA-MMSystem (MDA based model management system), is developed. Based on the system, this method is implemented in SoftPM project. Compared with the traditional solutions, the development efficiency is greatly improved and the cost is reduced. Finally, one example of the model applications and model merging is provided.

Key words: TRISO-model (tridimensional integrated software development model); MDA (model driven architecture); MOF (meta object facility); metamodeling; model transformation

摘 要: TRISO-Model(tridimensional integrated software development model)是为处理软件开发的复杂性和动态性而提出的三维集成软件开发方法学,其中,多维模型之间的语义一致性维护以及对模型应用中公共操作部分的重用,提出了基于一致语义进行模型管理的需求.给出了基于 MDA(model driven architecture)进行模型管理的方法 MDA-MMMethod(MDA based model management method),应用 MDA 的 4 层模型管理结构,基于 MDA 核心标准

* Supported by the National Natural Science Foundation of China under Grant Nos.60273026, 60573082 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2002AA116060 (国家高技术研究发展计划(863))

Received 2006-02-13; Accepted 2006-05-11

MOF(meta object facility)所提供的公共语义基础管理模型和元模型,MDA-MMMethod 支持各种 MDA 模型操作标准实现在 TRSIO-model 应用中的重用.开发了相应的支持系统 MDA-MMSystem(MDA based model management system),应用于 SoftPM 的项目实践中.与传统方法相比,模型应用的开发效率得到了显著提高,同时降低了开发成本.最后,给出了模型融合的应用实例介绍.

关键词: TRISO-model(tridimensional integrated software development model); MDA(model driven architecture); MOF(meta object facility);元建模;模型转换

中图法分类号: TP311 文献标识码: A

TRISO-Model(tridimensional integrated software development model)是面向软件开发的复杂性和动态性而提出的三维软件集成开发方法学.它从过程-人(经济)-技术的三维视角出发,考察软件开发中涉及的各种因素及其相互关系,支持对应集成软件开发的系统化建模和管理[1].

TRISO-Model 方法学的应用,提出了对模型进行一致管理的需求:首先,从不同维度出发对软件开发建模得到的模型并不是相互孤立的,它们相互交织,各维度模型中都包含其他维度因素的影响,模型之间的语义一致必须保证;其次,各种模型应用中存在一些重合的操作,对这些公共操作的提取和重用,需要一致的模型访问接口的支持.例如:模型融合[2]中涉及到元建模、模型数据交换、模型转换等操作,而在模型执行等其他模型应用形式中同样也存在这些“原子”操作.基于统一接口构建这些公共操作的应用实现并重用于各种模型操作中,将有助于提高模型管理的开发效率并降低成本.

MDA(model driven architecture)提供了一个基于一致语义基础的模型管理架构[3].基于其核心标准 MOF(meta object facility)[4]所定义的公共语义标准,MDA 为其框架中模型和元模型的各种模型操作(如模型存储、模型表示、模型数据交换、模型转换等)定义了统一的接口规范.

我们将 MDA 框架应用于 TRSIO-Model 的模型管理,提出 MDA-MMMethod(MDA based model management method),支持 TRISO-Model 中模型融合和模型执行等应用实现,并开发了对应的支持系统 MDA-MMSystem (MDA based model management system).以上方法和系统被应用于中国科学院软件研究所的国家 863 重大项目 SoftPM[5],为模型的管理和执行提供了有力的支持.本文第 3 节中给出了其中模型融合的应用实例介绍.

1 基于 MDA 的模型管理方法 MDA-MMMethod

MDA-MMMethod 方法中用 MDA 的 4 层模型管理结构有效地组织模型和元模型.在 TRSIO-model 的应用中,过程、人、技术这 3 个维度上多年的研究积累了众多遗留的方法论、模型和元模型.为支持对这些研究成果的重用,MDA-MMMethod 方法中通过“MOF 规范化”操作将这些模型/元模型转换为基于 MOF 语义的表示形式,纳入到 MDA 的统一管理框架中.基于 MOF 这个一致的语义基础,MDA 的模型操作标准也被方便地重用于 TRISO-Model 的各种高级模型应用实现中.

1.1 应用MDA的4层模型结构管理模型/元模型

MDA 制定的初衷是支持业务逻辑和底层平台技术的分离,以保护业务建模的成果不受技术变迁的影响[3].为实现这个目的,OMG(object management group,国际对象管理集团)定义了核心的语义标准 MOF,并基于 MOF 制定了对框架中所有模型/元模型进行统一管理的 4 层模型管理结构.

MDA-MMMethod 方法中应用 MDA 的 4 层管理结构对模型/元模型进行组织.如图 1 所示,*M*2 层的 I-SPEM(integrated SPEM)是扩展 SPEM(software process engineering metamodel)[6]得到的覆盖软件开发 3 个维度(SE Process,SE Human 和 SE Technology)的扩展元模型[1].其中定义了对应各维度的 3 个内部集成包 Process Integration,Service Integration 和 Development Integration 以及表达维度之间关系的 3 个外部集成包 Management Integration,Data Integration 和 Use Integration.*M*1 层是 *M*2 层元模型的实例,如过程领域的 CMMI (capability maturity model integration)模型、人(经济)领域的 COCOMOII(constructive cost model)模型以及技术

领域的瀑布模型等.如图 1 所示,*M*2 层模型/子模型(I-SPEM 中对应各维度的集成包可以视为领域子模型)都基于 MOF 定义得到,这保证了它们之间的相互理解、相互映射以及互操作和元数据的交换.即,MOF 为 TRISO-Model 中的模型/子模型提供了统一的语义基础.在此基础上,MDA 的各种模型操作标准可以在 TRISO-Model 的模型应用中重用.

我们将 TRISO-Model 中的模型操作分为两类:高级模型操作和原子模型操作.前者对应模型的各种高级应用形式,如模型融合、模型检测\验证、模型执行等;后者则是易于重用的可直接被 MDA 模型操作标准支持的各种底层模型操作,如模型表示、模型交换和模型转换等.高级模型操作通常包含原子模型操作的组合.

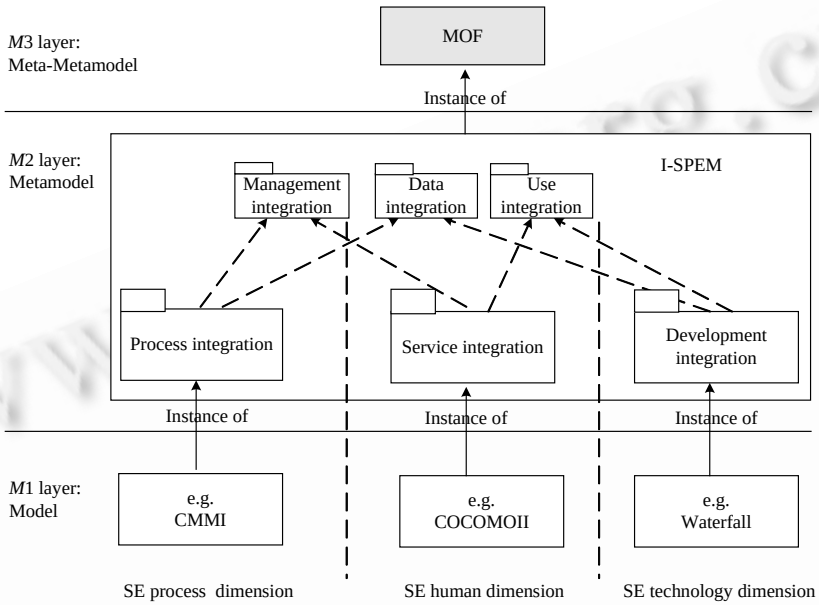

Fig.1 The model/metamodel management in TRISO-model applications

图 1 TRISO-Model 应用中的模型/元模型管理

如图 2 所示,MDA 中定义了对应各种原子模型操作的标准规约.其中,基于 MOF 定义的元模型支持特定的模型表示和存储,XMI(XML metadata interchange)[7]定义基于 XML 的模型数据交换标准,QVT(query/view/transformation)[8]定义模型转换的规则描述语言.JMI(Java metamodel interface)[9]是 JCP(Java community process)定义的对 MDA 框架中模型/元模型进行访问操作的 Java 接口.

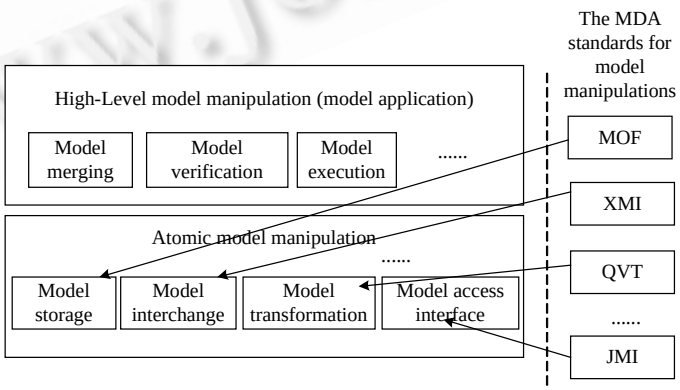

Fig.2 Model manipulations in TRISO-model and related MDA standards

图 2 TRISO-Model 中的模型操作及 MDA 中的相关标准

此外,更多的标准还在制定之中.需要指出的是,利用 MDA 的标准必须满足一个前提条件:待操作的模型必须是“符合 MOF(MOF-compliant)”的.这一点是模型语义一致的保证.

1.2 形式化定义

相关概念的形式化定义如下:

定义 **1**. 用 $m(s)/mm$ 表示基于元模型 mm 对系统 s 建模得到的模型 m.例如:应用 SPEM 对 RUP(rational unified process)建模得到的模型为 $m(RUP)/SPEM$;应用 UML(unified modeling language)建模得到的则是模型 $m'(RUP)/UML$.

定义 **2**. 模型 m 也可以表示为二元组 $m=(E,R)$,其中:E 为模型元素 e_i 的集合,$E=\{e_1,e_2,\ldots,e_n|n\geq 1\}$;$R$ 为元素之间的关系集合,$R=\{r_1,r_2,\ldots,r_m|m>0\}$,$R=E\times E$.

定义 **3**. 将模型元素之间的实例化关系记作

$$insof:E\times E\rightarrow\{\text{true,false}\}.$$

例如,$insof(e,f)$=true 表示模型元素 e 为 f 的实例化.

引理 1. $insof$ 函数具有传递性,$insof(e,f)\wedge insof(f,g)\Rightarrow insof(e,g)$.

即如果 e 可以表示为 f 的实例,f 又可以表示为 g 的实例,则 e 可以表示为 g 的实例.

定义 4. 将模型之间的符合关系记作

$$mc:\{m\}\times\{m\}\rightarrow\{\text{true,false}\}.$$

设模型 $m1=(E1,R1)$,模型 $m2=(E2,R2)$,$mc(m1,m2)$=true 必须满足以下条件:

$\forall e1\in E1,\exists e2\in E2$,使得 $insif(e1,e2)$=ture

并且

$\forall r1\in R1,\exists e2\in E2$,使得 $insif(r1,e2)$=ture

mc 表示的是两个模型在语法上的符合关系.如果 $mc(m1,m2)$=true,同时 $m1$ 还满足 $m2$ 定义的语义约束,则 $m2$ 是 $m1$ 的元模型;反之,如果模型 $m2$ 是 $m1$ 的元模型,则以上条件必然满足.

定义 **5**. 模型 $m2$ 是模型 $m1$ 的元模型,记作 $mm(m1)=m2$.

定义 **6**. 模型 m 符合 MOF,记作 $mof(m)$=true,也即,$mc(m,MOF)$=true.

MOF 模型是自描述的,即,$mof(MOF)$=true,$mm(MOF)=MOF$.

定理 **1**. 如果模型 m 的元模型 mm 符合 MOF,则模型 m 符合 MOF.即

$$mc(m,mm)\wedge mof(mm)\Rightarrow mof(m).$$

证明:

令 $m=(E1,R1)$

$mm=(E2,R2)$

由 $mc(m,mm)$=ture,根据定义 4,

$\forall e1\in E1,\exists e2\in E2,insif(e1,e2)$=ture;

由 $mof(mm)$=ture,即 $mc(mm,MOF)$=true,对 $e2,\exists e3\in E2,insif(e2,e3)$=ture;

根据引理 1,

$e1.insif(e3)$=ture

同理

$\forall r1\in R1,\exists e2\in E2,insif(r1,e2)$=ture;

由 $mof(mm)$=ture,即 $mc(mm,MOF)$=true,对 $e2,\exists e3\in E2,insif(e2,e3)$=ture;

根据引理 1,

$r1.insif(e3)$=ture

根据定义 4,有 $mc(m,MOF)$=true,即 $mof(m)$=ture.

定理得证. □

定义 **7**. 将元模型 $m1(E1,R1)$和 $m2(E2,R2)$之间的转换规则记作 r_{m2}^{m1}.

$r_{m2}^{m1}=\{r_1,r_2,\ldots,r_k|k>0\}$,

$r=(e1\times e2)\vee(r1\times r2),e1\in E1,e2\in E2,r1\in R1,r2\in R2$.

r_{m2}^{m1} 也是一个模型,其元模型为 QVT,$mc(r_{m2}^{m1},QVT)$=true;同时,$mof(QVT)$=true,因此有 $mof(r_{m2}^{m1})$=true.

推论 **1**. 对模型 $m(s)$,要应用 MDA 的各种标准实现,必须满足前提条件 $mof(m(s))$=true.将 s 用元模型 mm 表示,如果 $mof(mm)$=true,则对 $m'(s)/mm$,条件满足.

根据推论 1,在 MDA-MMMethod 方法中,我们通过重建模或模型转换的途径,用“符合 MOF”的元模型表示待操作的模型,从而使待操作的模型满足“符合 MOF”的前提条件.然后,在对这些模型的高级操作中,可以重用 MDA 的标准实现.以 TRISO-Model 应用中的高级模型操作——模型融合为例,其形式化定义如下:

定义 **8**. 将模型之间的语义融合记为 $mg:\{m\}\times\{m\}\rightarrow\{m\}$.

对模型 $m1(s1)/mm1$ 和模型 $m2(s2)/mm2$:

$$mg(m1(s1)/mm1,m2(s2)/mm2)=m3(s3/mm3).$$

$mm3$ 可以是各种元模型.通常而言,选择 $mm1$ 或者 $mm2$ 作为融合后模型的元模型 $mm3$.

定义 **9**. 如果待融合的模型基于不同的元模型,则将它们的模型融合记为异构模型融合 $exmg$.

$$exmg(m1(s1)/mm1,m2(s2)/mm2)=m3(s3)/mm3|mm1\neq mm2.$$

定义 **10**. 如果待融合的模型基于相同的元模型,则将它们的模型融合记为同构模型融合 $enmg$.

$$enmg(m1(s1)/mm1,m2(s2)/mm2)=m3(s3)/mm3|mm1=mm2.$$

第 3 节的应用研究部分将通过实例介绍以 MOF 规范化操作将一个异构模型融合问题简化为同构模型融合的方法.

1.3 MOF规范化

“MOF 规范化”方法对于不符合 MOF 的模型/元模型,通过重建模或模型转换的途径,基于 MOF 这个共同的语义基础进行统一管理,以支持后续的模型操作.MOF 规范化的算法如下:

输入:TRISO-Model 中的模型 $m(s)/mm1$,$mof(mm1)\neq true$,$mof(m)\neq$true.

输出:与 m 语义一致的 $m'(s)/mm2$,$mof(mm2)$=true,$mof(m')$=true.

1) if $!\exists mm2$,满足 $mof(mm2)$=true 而且适合于表达 m 所在领域的抽象概念

1.1) 基于 MOF 进行元建模,得到 $mm2$;

2) 通过重新建模或者模型转换得到 $m2$;

2.1) Case 1 //通过重建模来实现

基于 $mm2$ 建模,得到 $m'(s)/mm2$;

2.2) Case 2 //通过模型转换来实现

2.2.1) if $!\exists r_{mm2}^{mm1}$

基于 QVT 制定转换规则 r_{mm2}^{mm1}/QVT;

2.2.2) 基于 r_{mm2}^{mm1} 进行模型转换,得到 $m'(s)/mm2$;

3) 算法结束;

其中,首先判断 MDA 框架 $M2$ 层中是否有适用的元模型 $mm2$.如果没有,则基于 MOF 进行元建模,得到 $mm2$.然后,可以选择两种方式实现 MOF 规范化:

1) 由建模人员使用 $mm2$ 对 m 进行重新建模表示,得到 $m'(s)/mm2$;

2) 将 m 转换为 $mm2$ 表示的模型形式.模型转换过程中,需要对应的转换规则 r_{mm2}^{mm1}/QVT.如果过去没有定义,则需要使用 QVT 进行描述得到.基于 r_{mm2}^{mm1}, QVT 的标准实现(即 QVT 引擎),可以将 $m(s)/mm1$ 转换为 $m'(s)/mm2$.由 $mof(mm2)$=true 可知,转换得到的模型 $m'(s)/mm2$ 是符合 MOF 的:$mof(m')$=true.

如图 3 所示,通过 MOF 规范化,位于 MDA 框架之外的模型被纳入到 MDA 框架的统一管理中.

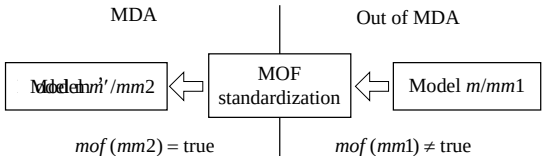

Fig.3 MOF standardization

图 3 MOF 规范化

算法中涉及 MDA 中的 3 种原子模型操作:

1) 基于 MOF 的元建模

基于 MOF 的元建模是根据 MOF 定义的建模结构和语义约束,对目标领域的领域知识进行提取和抽象的过程.MOF 为领域建模提供了通用的抽象语法,包括 5 个主要的语法结构{Class,Association,DataType,Package,Constraints},分别用于描述元类(概念)、关系、数据类型、包(组织形式)和约束.为精确描述约束语义,OMG 定义了 OCL(object constraint language)标准[10]作为统一的约束描述语言.

作为示例,图 4 所示的一个简单的过程元模型 SimPM 可描述为

Class{Actor,Activity,Artifact}

Association{perform,Actor,Activity*}

Association{input,Activity*,Artifact*}

Association{output,Activity*,Artifact*}

Association{responsible for,Actor,Artifact*}

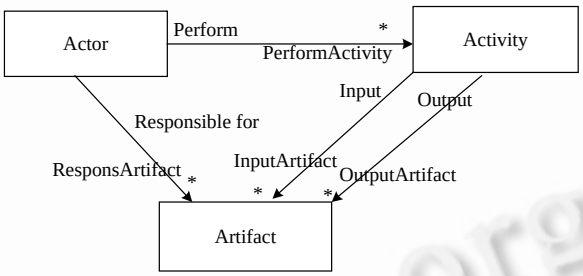

Fig.4 Process metamodel: SimPM

图 4 过程元模型 SimPM

该例子中没有涉及 DataType 和 Pakcage.其中的*表示多重关系,如一个角色(actor)可以执行(perform)多个活动(activity),一个活动可以有多个输入/输出的工件(artifact).元模型中的语义约束用 OCL 描述,例如 Actor 负责(responsible for)的 Artifact 必须是其执行的某一个活动的输出(output),用不变式表示如下:

context Actor inv:

Self. performActivity.ouputArtifact→includesAll (self. responsArtifact)

这些语义约束将在基于该元模型进行建模时保证模型的语义正确,具体地,由建模人员或者工具来维护.元建模所面向的领域的粒度变化范围较大,既可以是面向对象分析设计(eg:UML)、软件过程(eg:SPEM)等较大的领域,也可以面向具体的项目和工具,如文献[11]中,Jean 为 Ms-Project 定义了基于 MOF 的元模型.

2) 基于符合 MOF 的元模型的建模

得到符合 MOF 的元模型 *SimPM*(*mof*(*SimPM*)=true)后,基于 SimPM 提供的语法结构和语义(自然语言的说明或形式化的语义约束)进行建模,得到模型 *proM*(*s*)/*SimPM*.图 5 所示为 proM 的内容片断,也可以表示为

Actor{Requirement engineer}

Activity{Requirement acquisition}

Artifact{SRS}

perform{Requirement engineer,Requirement acquisition }

output{Requirement acquisition,SRS}

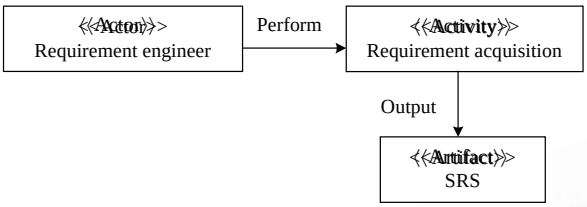

Fig.5 Process model: *prom*(*s*)/*SimPM* (partial)

图 5 过程模型 *proM*(*s*)/*SimPM* 内容片断

可以看到,模型内容满足第 1.2 节中的定义 4,每个元素和关系都是 SimPM 中元素的实例化.

由(*mc*(*proM*,*SimPM*)=true)∧(*mof*(*SimPM*)=true),根据定理 1 可得 *mof*(*proM*)=true,*proM* 模型也是符合 MOF 的.

为支持对满足 *mof*(*m*)=true 的模型 *m* 的动态访问和操作,MOF 定义了反射接口[4]:

(1) Reflective::RefBaseObject //获取所有的 MOF 对象.

(2) Reflective::RefObject //获取所有对象(对应 MOF 中 classifier 的对象).

(3) Reflective::RefAssociation //获取所有 Association 对象.

(4) Reflective::RefPackage //获取所有 Package 对象.

基于以上反射接口,MOF 的标准实现可以在运行时利用 MOF 中定义的反射接口动态读取 *m* 的元模型,获得元模型的抽象语法,并依此提供建模和表示支持.例如,对模型 proM,通用的 MOF 实现可以在运行时读取 SimPM 元模型,并通过下面的操作来分别显示 proM 中的 Actor,Activity,Artifact 对象列表,并进行相关操作.

```
Collection theModelClass=theModel.refAllClasses();     // theModel. refAllClasses()实现了 RefObject 接口
Iterator i=theModelClass.iterator();
while (i.hasNext()) {
    RefClassImpl m2class=(RefClassImpl) (i.next());
    Collection m1objects=m2class.refAllOfClass();
    showClass(m1objects)                                //列表输出所有对象实例;
    …                                                   //进行其他操作;
}
```

同样的方式,基于 MOF 的其他模型操作的标准实现也支持对该模型的操作.例如,可以利用 XMI 输出 XML 形式的数据文件,在不同应用程序之间交换模型数据,也可以通过 QVT 标准实现进行模型转换.

3) 基于 QVT 的模型转换

对于元模型 *mm*1 和 *mm*2,如果满足(*mof*(*mm*1)=true)∧(*mof*(*mm*2)=true),则可以使用 QVT 描述它们之间的转换规则,并使用 QVT 的标准实现(QVT 引擎)来实现从 *m*1(*s*)/*mm*1 到 *m*2(*s*)/*mm*2 的转换.

继续前面的例子,如果要将基于 simPM 的过程模型用 UML Profile 的形式表示,如图 6 所示,用 Use Case 图来表示 Actor 和 Activity 之间的关系,用类图表示 Actor 和 Artifact 之间的关系.以对应 proM 中 Activity 对象的转换为例,QVT 表示的转换规则为

```
Rule Actor2Actor&Class()
Source
    a: SimPro::Actor;
```

Target

ac: UML::Actor;

cs: UML::Class *stereotype=Actor*;

Mapping

a. name ⟨~⟩ ac.name;

a. name ⟨~⟩ cs.name

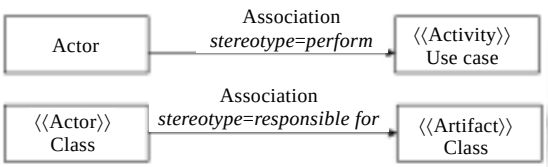

Fig.6 UML profile for transformation target metamodel

图 6 转换目标元模型对应的 UML profile

proM 中的 Actor 对象被转换为 UML 表示的同名 Actor 对象以及版型(stereotype)为“Actor”的同名 Class 对象.基于以上规则定义,QVT 引擎转换 proM 的结果如图 7 所示.

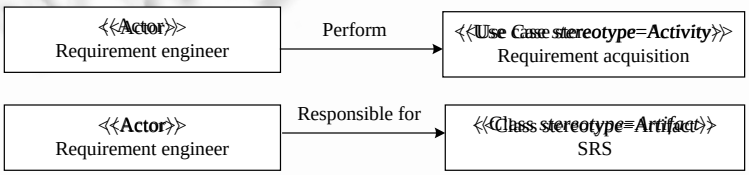

Fig.7 The proM model expressed by UML profile

图 7 UML profile 形式表达的 proM 模型

在经过 MOF 规范化操作之后,TRISO-model 中的所有模型都被纳入到基于 MOF 的 MDA 统一管理框架中.基于 MOF 这个统一的语义基础,MDA 的各种标准实现都可以在 TRISO-model 的模型应用中进行重用.为了支持 MDA 标准实现的重用,我们开发了相应的支持系统 MDA-MMSystem.

2 基于 MDA 的模型管理系统 MDA-MMSystem

MDA-MMMethod 方法中通过对模型的统一管理,支持在 TRISO-model 的模型应用中重用 MDA 的标准实现.MDA-MMSystem 中集成了主要的 MDA 模型操作标准的实现,为 TRISO-model 的应用实现提供了强大的支持.图 8 所示为 MDA-MMSystem 的结构示意.经过 MOF 规范化之后,其中所有的元模型和模型 m 都满足条件 $mof(m)$=true,因此可以通过 MOF 模型仓库对它们进行统一的存储管理.对这些模型/元模型的操作有两种途径:

1) 通过 XMI 实现读取/写入 XML 形式的模型数据文件;

2) 通过 JMI 实现使用 Java APIs 直接访问.

在此之上,是基于 OMG 制定的标准接口的各种模型操作的标准实现:QVT 实现、MOF 实现、UML 实现和 OCL(object constraint language)实现等,对应 TRISO-model 中的各种原子模型操作.基于这些公共的标准实现,最上一层是与业务相关的高级模型操作,如模型融合、模型检查\验证和模型执行等.

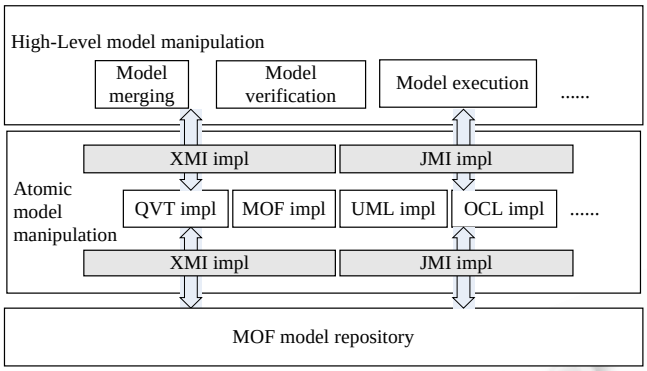

Fig.8 Structure of MDA-MMSystem

图 8 MDA-MMSystem 结构示意

对应图 8 的结构示意,MDA-MMSystem 中主要包括 4 个模块:

1) 元建模工作区 MW(metamodeling workspace).
2) 通用模型编辑工作区 GMW(general modeling workspace).
3) 转换规则编辑工作区 TEW(transformation rule edit workspace).
4) 模型转换引擎 MTE(model transformation engine).

近年来,MDA 标准的实现工作进展迅速,其中的核心标准如 MOF,JMI,XMI 和 QVT 方面都出现了众多优秀的开源和商业软件.MDA-MMSystem 中重用了一些开源的标准实现,如在 MOF 仓库部分重用了 Unisys 的 CIM (computation independent model);在 JMI 和 XMI 方面,重用了 nsUML 的实现模块;QVT 的情况比较特殊,到本文写作之时,最终标准尚未公布,但相关的研究和提案的实现已有不少.我们基于法国 Lip6 小组的 Modfact 项目[12],选择 SimpleTRL 语言作为转换规则的定义语言,并在 Modfact 项目的基础上进行改进,开发了支持 SimpleTRL 的转换引擎 MTE.其他部分,通用模型编辑工作区 GMW 基于 JMI 实现开发得到;由于受人力和时间的限制,使用 UML 的 CASE(computer aided software engineering)工具 Rational 加上 Unisys 的 xmi import/export plug-in 作为案例研究部分的元建模工作区 MW,另外使用了普通的文本编辑工具作为 TEW 来编辑转换规则.

基于对标准实现的重用,MDA-MMSystem 的使用极大地降低了开发成本,并缩短了开发周期.同时,基于标准的实现还给 MDA-MME 带来了开放和可扩展的优点,更新的标准实现可以基于统一的接口快速集成进来,与业务相关的高级模型操作对应的实现模块也可以快速集成,在 MDA-MMSystem 的基础上做二次开发进行支持.

第 3 节中,我们通过模型融合的应用实例介绍 MDA-MMMethod 方法和 MDA-MMSystem 工具在 SoftPM 项目中的应用.

3 应用实例

SoftPM 是支持 TRISO-model 的软件质量管理平台[5],其中涉及过程、人、技术 3 个维度上不同模型之间的建模和应用.下面以模型融合这种模型应用形式为例,介绍 MDA-MMMethod 方法在 SoftPM 中的应用.

3.1 CMM与EPM的融合

CMM 是近年来人们广为接受的过程评价和改进模型,用以指导组织的过程改进.通常来说,软件组织已经有了成型的软件开发过程,CMM(capability maturity model)实施的过程是由专家发现已有过程中与 CMM 不相符合的地方,指导改进的方向.CMM 侧重过程维和管理相关的概念,与企业具体的过程执行环境无关;而 SoftPM 中原有的企业过程模型 EPM(enterprise process model)侧重技术维相关的概念,并包含了过程执行的环境特征信息.CMM*m* 和 EPM 的融合将为 CMM 的实施提供有效的支持.

图 9 所示为融合前的 EPM 模型片断.其中,文档化的用例规约(UseCase Doc,对应 CMM 中的需求文档)并没有经过评审就纳入了基线管理.这一点不符合 CMM 的要求,而是为了考察模型融合的效果而特意设计的.

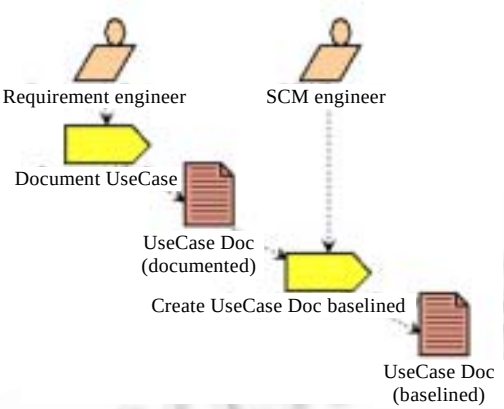

Fig.9 EPM before model merging

图 9 融合前的 EPM

3.2 应用MDA-MMMethod方法的融合实现

EPM 是基于 SPEM 建模得到的,SPEM 是 OMG 制定的软件过程工程元模型标准[6],它是符合 MOF 的,由

$(mc(EPM,SPEM)=\text{true})\wedge(mof(SPEM)=\text{true})$

$\Rightarrow(mof(EPM)=\text{true})$

可知,EPM 符合 MOF.按照 MOF 规范化算法,首先要基于 SPM-CMM 进行元建模得到 CMM*m*.应用 MDA-MMMethod 方法,CMM*m* 和 EPM 的融合步骤如图 10 所示.首先是基于 MOF 的 CM 元建模,在文献[13]中我们定义了 SPM-CMM,$mof(SPM\text{-}CMM)=\text{true}$,因此,用它建模得到的 CMM 过程模型 CMM*m* 也是符合 MOF 的,$mof(CMMm)=\text{true}$.

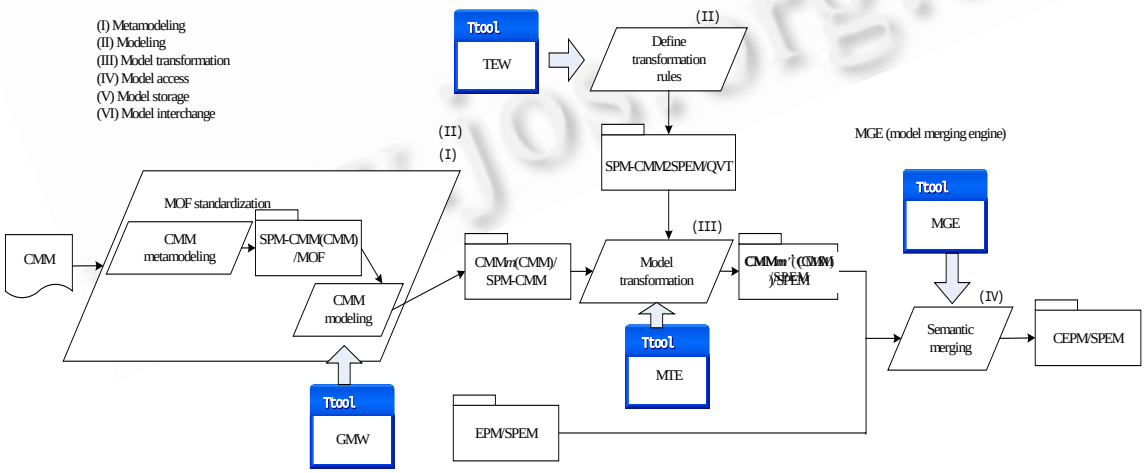

Fig.10 The CMM*m*+EPM merging steps and corresponding tools

图 10 CMM*m* 和 EPM 的融合步骤和对应的工具

CMM*m* 和 SPEM 都是基于 MOF 这个统一语义基础的待操作模型,因此,模型融合的实现中可以重用 MDA 框架提供的各种原子模型操作实现.图 10 标注出重用元建模、建模、模型转换、模型访问等原子模型操作的各个步骤.另外,基本上,每个环节中都涉及到了模型存储和模型数据交换这两种原子模型操作,因此没有特别标注.对应地,图 10 中也标注出了对应原子模型操作的 MDA-MMSystem 各模块在其中的重用.

CMMm 实质上是对 CMM 模型的符合 MOF 语义基础形式的重新组织,目的是支持后续的 MDA 模型操作.由于 *mc*(*SPM-CMM*)=true,因此可以在通用模型编辑工作区 GMW 进行建模.如图 11 所示,选择特定的元模型(SPM-CMM)后,可以在 GMW 中根据 SPM-CMM 中定义的模型结构对 CMM*m* 进行添加/删除/编辑/查询等基本操作.由于元模型的特定领域知识是运行时动态读入的,GMW 并没有对应的特定语义,因此对所有的元数据,GMW 中的 GUI(graphical user interface)界面都是相同的.文献[13]中构建了对应 CMM2 级的 CMM*m* 模型,因受篇幅所限,本文不再介绍相关细节.

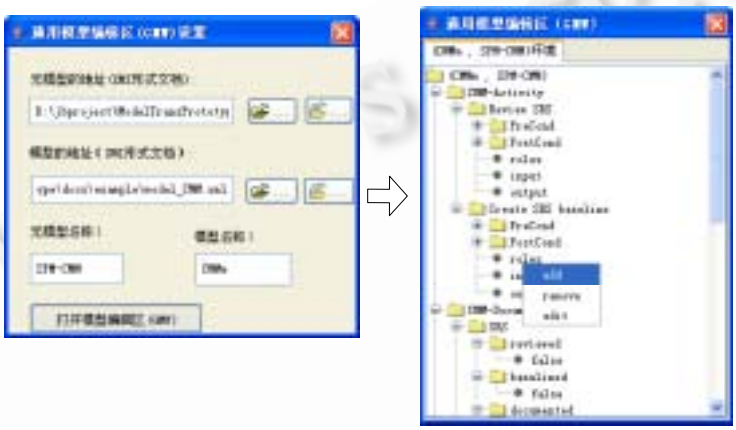

Fig.11 Modeling in GMW

图 11 在 GMW 中进行建模

如图 10 所示,接下来要将 CMM*m*(CMM)/SPM-CMM 转换为用 SPEM 元模型表示的形式.因此,首先需要定义元模型 SPM-CMM 和 SPEM 之间的转换规则.

我们基于转换语言 SimpleTRL 在 TEW 中编辑转换规则 SPM-CMM2SPEM,包括模型元素之间的对应以及关系之间对应的转换规则.作为示例,CMM*m* 中的 CMM-WorkProduct 对象转换到 CMM*m*′中的 WorkProduct 对象的规则如下所示:

```
Rule CMM-WorkProduct2WorkProduct()
creates WorkProduct
return #result{
    #result.name=#context.name    //对应为 CMMm′中同名的 WorkProduct 对象;
    if (#context.reviewed=true)
        //如果 CMMm 中 CMM-WorkProduct 对象的 reviewed 属性为 true,则在 CMMm′中创建对应状
        态机;
        {#result.state.add(context.reviewed.CMM-WorkProductStateReviewAttribute2StateMachine())}
    ……
}
```

转换引擎实现 MTE 以 XMI 形式读入转换前后的元模型以及转换规则模型,图 12 所示为模型转换引擎 MTE 的运行界面.转换得到 SPEM 元模型形式表示的 CMM 过程模型:CMM*m*′(CMM)/SPEM,由

$$(mc(CMMm',SPEM)=\text{true})\wedge(mof(SPEM)=\text{true})\Rightarrow(mof(CMMm')=\text{true})$$

同样,CMM*m*′也可以在 GMW 中进行读写和编辑.

至此,待融合模型(CMM*m* 和 EPM)已经被转换为基于同样元模型(SPEM)的表达形式(CMM*m*′和 EPM).接下来的工作是对这两个模型的语义进行融合,其中使用了基于匹配和人工干预相结合的融合算法[2],对 EPM 中与 CMM 语义冲突的部分进行相应编辑.在这种高级的模型操作实现中,也涉及到对模型/元模型的访问和操作、XMI 形式的模型数据交换等.在实现中,基于第 2 节中介绍的开放架构,这部分模块 MGE(model merging engine)被方便地集成到 MDA-MMSystem 中来.

Fig.12 Transformation engine MTE

图 12 转换引擎 MTE

模型融合的结果得到符合 CMM 要求的 CEPM(CMM-compliant EPM)模型,它也是基于 SPEM 元模型的.如图 13 所示,WorkDefinition:Create UseCase Baseline 前增加了需求文档评审这一环节.修正后的 CEPM 满足了 CMM 的要求,达到了模型融合的预期目的.

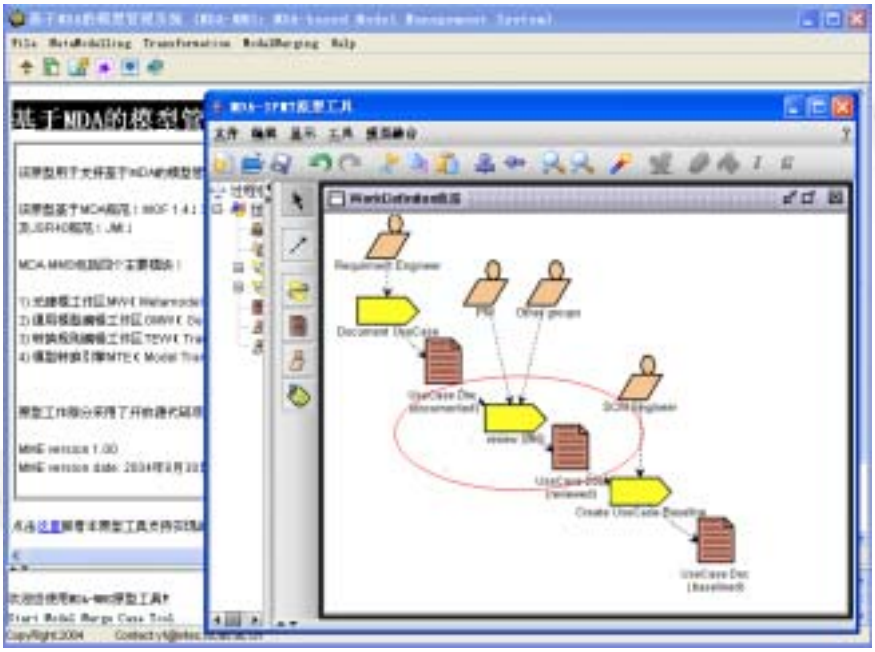

Fig.13 Resulted model CEPM after transformation in MDA-MMSystem

图 13 MDA-MMSystem 中转换后的 CEPM 模型

在该实例研究中,MDA-MMMethod 方法的优势得到了证明.在模型融合的操作实现中,基于 MDA-MMSystem 充分重用了模型存储、表达、数据交换、模型转换等方面已有的程序实现.与具体融合业务相关的

融合模块 MGE 可以方便地整合到整个系统中.与传统方法相比,实现效率得到极大的提高.

4 结束语

三维的集成软件开发方法学 TRISO-model 从不同维度对软件开发进行建模得到了多个相互交织的领域模型,对这些模型的语义一致性维护以及对模型的进一步操作导致了对基于一致语义基础的支持底层模型操作实现的重用的公共框架的需求.

MDA 中定义了各种模型操作的标准化接口,并且为模型的统一管理提供了良好的框架.基于 MDA 对模型进行集成管理开始引起研究人员的关注,也先后出现了很多 MDA 的实现框架.其中著名的有 IBM Eclipse 项目下的 EMF、美国 Vanderbilt 大学研发的 GME 以及德国 Paderborn 大学研发的 FUJABA.但是,由于标准制定时间以及实现上的原因,大多数框架并没有完全遵循 MOF 标准.例如 EMF 实现中的核心元模型并不是 MOF,而是对应 MOF API 核心子集的一个高效 Java 实现——Ecore.这使得 MDA 的标准定义失去了原有的意义,不同标准实现之间的交互和重用难以实现.另外,OMG 自己也提出了基于 MDA 支持模型集成计算的计划,在元模型层次上支持各种工具的集成[14],但目前还处于起步阶段.法国南特大学的 Jean 教授提出了将 MDA 标准应用于软件过程模型,支持过程驱动的模型工程的想法[15,16].Jean 指出了基于 MDA 的模型工程的众多优势和潜在的应用形式,不过并没有给出具体的模型应用实例和工具.

我们将 MDA 框架应用于 TROSO-model 的模型管理,给出了 MDA-MMMethod 方法.对于遗留的不符合 MDA 统一语义基础 MOF 的模型/元模型,MDA-MMMethod 中给出了 MOF 规范化算法.为支持 MDA-MMMethod 方法,支持对各种 MDA 标准实现的重用,我们开发了 MDA-MMSystem 系统.方法和系统被成功应用在 SoftPM 项目中,为提高模型应用实现的效率和降低实现成本起到了重要的作用.

目前,MDA 的几个标准如 QVT 还在制定和完善之中.随着标准的进一步完善和更多优秀标准实现的涌现,下一步的工作之一将是同步更新和完善 MDA-MMSystem 系统.另外,将考虑支持更多的高级模型操作实现,例如对基于 SPEM 的过程模型的执行支持等.在实现这些模型应用的同时,也将对 MDA-MMSystem 系统提出更多要求并进行相应的改进.

References:

[1] Li MS. Expanding the horizons of software development processes: A 3-D integrated methodology. In: Li MS, Boehm BW, Osterweil LJ, eds. Unifying the Software Process Spectrum, Int'l Software Process Workshop (SPW 2005). Beijing: Springer-Verlag, 2005. 54−67.

[2] Yuan F, Li J. A MDA based approach for merging CMM and EPM. In: Li MS, Boehm BW, Osterweil LJ, eds. Unifying the Software Process Spectrum, Int'l Software Process Workshop (SPW 2005). Beijing: Springer-Verlag, 2005.

[3] Miller J, Mukerji J, eds. Model driven architecture. Ormsc/2001-07-01: Needham, Object Management Group. 2001. 1−31. http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

[4] Object Management Group. Meta object facility (MOF) specification. Version1.4, formal/02-04-03: Needham, Object Management Group. 2002. http://doc.omg.org/formal/02-04-03

[5] Wang Q, Li MS. Software process management practices in China. In: Li MS, Boehm BW, Osterweil LJ, eds. Unifying the Software Process Spectrum, Int'l Software Process Workshop (SPW 2005). Beijing: Springer-Verlag, 2005. 317−331.

[6] Object Management Group. Software process engineering metamodel specification. Version 1.0, Formal/02-11-14: Needham, Object Management Group. 2002. 1−98. http://www.omg.org/cgi-bin/doc?formal/2002-11-14

[7] Object Management Group. XML metadata interchange (XMI) specification. Formal/03-05-02: Needham, Object Management Group. 2003. http://www.omg.org/cgi-bin/doc?formal/2003-05-02

[8] Object Management Group. MOF 2.0 query/views/transformations RFP. Ad/02-04-10: Needham, Object Management Group. 2002. http://www.omg.org/cgi-bin/doc?ad/2002-04-10

[9] Java Community Process. Java metadata interface (JMI) specification. Version 1.0, JSR40, 2002. http://jcp.org/aboutJava/communityprocess/final/jsr040/

[10] Bézivin J, Breton E. Applying the basic principles of model engineering to the field of process engineering. UPGRADE, 2004,5(5): 27−33. http://www.upgrade-cepis.org/issues/2004/5/upgrade-vV-5.html

[11] Object Management Group. UML 2.0 object constraint language specification. Version 2.0, Ptc/03-10-14: Needham, Object Management Group. 2003. http://www.omg.org/cgi-bin/doc?formal/2003-10-14

[12] http://modfact.lip6.fr

[13] Li J, Li MS, Wu ZC, Wang Q. A SPEM-based software process metamodel for CMM. Journal of Software, 2005,16(8):1366−1377 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/16/1366.htm

[14] http://mic.omg.org

[15] Jean B. From object composition to model transformation with the MDA. In: Proc.of the 39th Int'l Conf. and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39). 2001. http://www.sciences.univ-nantes.fr/info/lrsg/Recherche/mda/TOOLS.USA.pdf

[16] Breton E, Bézivin J. Process-Centered model engineering. In: Wang GJ, Lupu CE, Wegmann A, eds. Proc. of the 15th IEEE Int'l Enterprise Distributed Object Computing Conf. Washington: IEEE Computer Society, 2001. 179−182.

附中文参考文献:

[13] 李娟,李明树,武占春,王青.基于 SPEM 的 CMM 软件过程元模型.软件学报,2005,16(8):1366−1377. http://www.jos.org.cn/1000-9825/16/1366.htm

袁峰(1977－),男,湖北咸宁人,博士生,主要研究领域为模型驱动架构,过程建模.

李明树(1966－),博士,研究员,博士生导师,CCF 高级会员,主要研究领域为智能软件工程,实时系统.