一个以Ada为核心的面向对象的开发系统

陈辉　王振宇（中国舰船研究院第七〇九所，武汉430074）

摘　要

本文将逐步求精的思想融合于面向对象的方法，形成了逐步求精的面向对象的方法学SROOM，提出了对象的三级抽象；给出了三级对象范型统一的表达及其生成算法。

一、现行面向对象方法学的困难与局限

面向对象的方法学(OOM)有时又叫面向对象的设计方法学[1]，分三步：第一步是理解问题的结构；第二步是用自然语言描述解决问题的策略；第三步分四个阶段：先识别对象及对象类型；再识别对象上的操作；然后为每个对象建立接口；最后用算法实现各个操作。

显然，这个方法学的前提是：非形式化策略易于根据客观对象进行描述且易于理解，这个假设对于一些实验性的小系统来说是成立的。然而，对于一个实用复杂的大系统，开发一个非形式化的策略是极其困难的，而且这种一次性的面面具到的描述势必复杂且免不了存在各种矛盾或遗漏，给后续的理解工作带来很大困难。其次是对象的识别问题。由于G.Booch将所有的对象放在同一个平面来考虑问题，试图一次性地识别出系统中的所有对象、对象之间关系及对象的有关属性，但一个复杂的大系统中所涉及的对象数量庞大且关系复杂而对象的属性(内涵)丰富又模糊；在系统的设计和实现时往往还要产生许多新对象，因此OOM到目前为止尚未解决对象的识别问题。另外，G.Booch也没有给出必要的验证与测试方法。

正是这些原因使得OOM难于超越实验范围。为了降低系统求解的复杂性，我们利用分治策略，将逐步求精的思想融合于面向对象的方法学OOM，形成了SROOM (Stepwise Refining OOM)。SROOM对系统描述的分割和次层化降低了描述难度与描述复杂性，减少了描述矛盾及描述冗余；系统中相关对象的尽量局部化简化了对象识别，而且验证与测试构成了SROOM的重要内容。

二、已有的工作及其与本系统的关系

我们从"七.五"期间开始对Ada语言与面向对象方法学相结合的研究，完成了一个Ada软件开发系统Ada-GOODS[8,9]，其特色如下：①是基于Ada语言的，②支持面向对象方法学，③支持图式的结构编辑，④支持从详细设计到编码的平滑过度。但是对初步设计及需求分析的支持还较薄弱，因此我们目前的工作是拓展Ada-GOODS对软件开发全生存期的支持，如图1所示：

三、SROOM与对象的三级抽象

用SROOM进行软件开发分三个阶段：

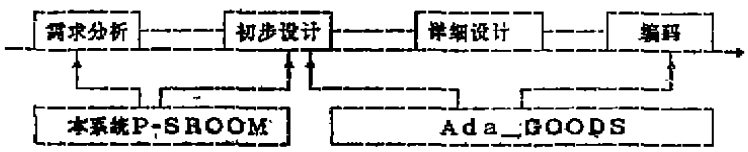

图1.P-SROOM与Ada-GOODS共同支持全生存期软件开发

面向对象的需求分析OORA、面向对象的设计OOD、面向对象的实现OOI，各个阶段中的软件开发活动都以对象为核心，从OORA到OOD到OOI三个阶段的对象无论在内容上还是在形式上都是不一样的。为了研究面向对象软件开发生存期中各级对象之间的映射关系及其转换方法，从而找到一套连贯的对象表达形式[3]及其转换算法，我们将对象抽象成三个层次：PMO对象、CMO对象、SMO对象(图2)。

定义1 PMO对象(Problem Model Objects)是需求分析OORA的结果，是对问题空间中的实体及其关系的抽象；一般采用非形式的表达。

定义2 CMO对象(Computing Model Objects)是设计阶段OOD的结果，是对PMO对象的设计求精。CMO对象完备地描述了系统中包含的所有类对象、它们之间的关系以及它们所具有的处理能力；一般使用精确的数学形式表达。

定义3 SMO对象(Software Model Objects)是实现阶段OOI的结果，是对CMO对象的软件实现；一般采用高级语言来表达。

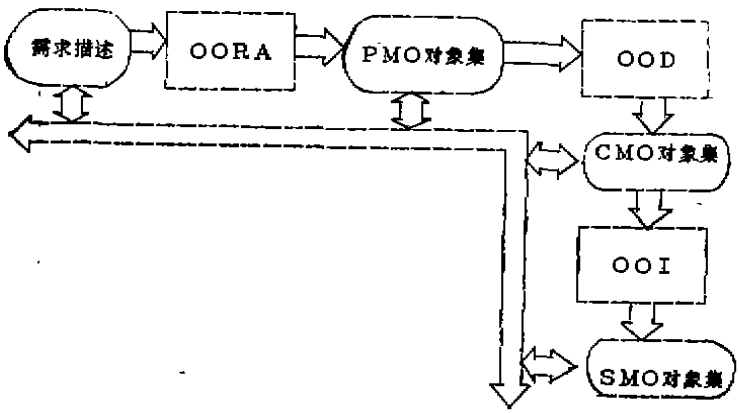

图2.对象的三级抽象与SROOM的生存期

SROOM不再把所有对象放在一个平面，而是把对象抽象为三个层次，面向对象的软件开发实质上就是使上述三级抽象平滑过渡，逐步求精；而且在上述各阶段都可提供相应的验证与测试手段。这样就可解决OOM的复杂性与验证等问题，至于对象的识别问题我们通过对功能的迭代求精来解决。

四、CMO对象的表达与面向对象的设计语言OODL

OODL由一个个的Class描述构成，其语法结构见附录1。一个CMO对象的表达应由以下几个方面组成[4]：

1.外部关系表达

外部关系是指一个类与另一个类之间的关系，我们对概念语义联系模型SAM进行改进，将对象之间的关系分成动态关系和静态关系：①动态关系是指class之间的动态引用与服务关系或者是信息生产与消费的关系，用A Serving B表示对象A使用对象B。②静态关系是指class之间所具有的内在关系，包括继承关系A-Kind-Of、结构关系A-Part-Of(Consists-Of)、成员关系A-Member-Of、约束关系Constraint。

2.内部属性表达

内部属性是指class中的状态属性和操作属性。一个class的状态属性由两个部分组成：①class的主抽象数据类型。我们采用的方法与Smalltalk相似，即对其结构是隐藏的：每一个class隐含地对应一个数据类型，其具体表示延迟到OOI阶段给出；②class的从抽象数据类型，即输入抽象数据类型。实质上是充当主抽象数据类型的参数角色。我们用"Abs-data-type"：<type-list>来表达class所含的从抽象数据类型。

消息协议是对使用与实现的共同约束和协议。我们用"Operations"：<operation-list>来表达该class的消息协议。

3.实现表达

我们用等式逻辑[7]来描述class中各个操作之间的语义关系，以描述这些操作是做什

么的。

五、OODL的实现——SMO对象的表达与生成

用户使用结构编辑器SDE-OODL，在OODL的语法制导下对系统中的类进行描述。再使用Relation-Valid对系统中的类进行关系检测，若语义关系不正确，则返回到SDE-OODL进行修正；否则，调用Trans2_Ⅰ自动生成各个对象的Ada表示的规格说明。然后在Trans2_Ⅱ下调用Ada-GOODS系统对各个对象进行实现求精，最后生成完整的SMO对象(图3)。

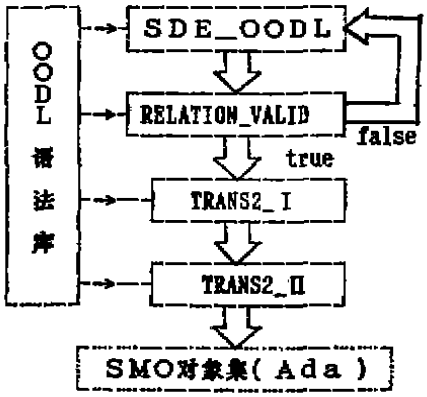

图3．CMO对象向SMO对象转换

1.SDE-OODL的设计与实现

总控程序Main-control调用OODL-EDIT对系统中的类对象进行描述编辑，在这个阶段随时可以求助于菜单选择MENU以提供所需的语法单元；用户输入的命令都是在命令窗COMM的监控下进行的。OODL-EDIT的每一个动作都是针对内部树Internal-Tree进行的，它存储了关于系统中所有类对象的所有信息。OODL的所有语法知识都存放在OODL语法库中(图4)。

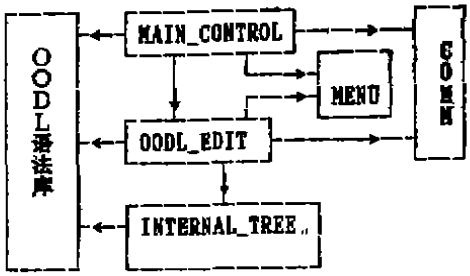

图4．SDE_OODL的模块结构图

2．Relation_Valid:系统对象的关系检测

关系检测主要是对系统生成的四条关系链进行检测，我们对以下几个方面进行了检测。

2.1 **系统未定义的class**(图5) class B在系统中没有定义，应取消这一描述，或者在系统中增加class B的描述。

2.2 **描述冗余**(图6) 冗余错误有两种情况：(1)C A_Kind_Of B and C A_Kind_Of A and B A_Kind_Of A显然C A_Kind_Of A冗余；(2) C A_Part_Of A and C A_Part_Of B AND B A_Part_Of A与上述情况相同。解决办法是求关系集的传递闭包，然后删除冗余[4]。

2.3 **循环圈错误**(图7) 循环圈错误有三种情况：(1) A A_Kind_Of C and C A_Kind_Of B and B_Kind_Of A；(2) A Consists_Of C and C Consists_Of B and B Consists_Of A；(3) 对于Serving关系，与上述情况相同。解决方法是解开循环圈；取消A、B、C之间的任意一个关系。

2.4 **多继承限制**(图8) 由于Ada_GO-

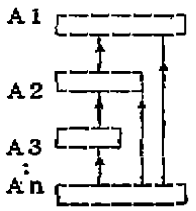

图6．冗余错误

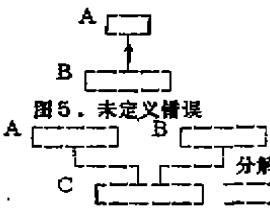

图5．未定义错误

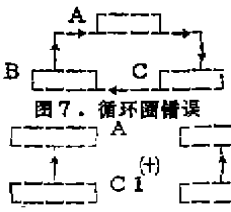

图7．循环圈错误

图8．多继承分解成单继承

```
task T is
   entry PUT(P:in PERSON);
end T;
task body T is
  ......
end T;
```

图9. 一个Ada任务T

```
package P__T is
  type PERSON is private;--ADT声明
  procedure PUT(P:PERSON);
  private PERSON is  ......
end P__T;
package body P__T is
   task T is
      entry PUT(P:in PERSON);
   end T;
   task body T is
     ......
  end T;
   procedure PUT(P:in PERSON) is
   begin
       T.PUT(P);
   end PUT;
end P__T;
```

图10. 任务T相应的任务类型程序包

ODS只能提供单继承机制，因此应对多继承关系描述进行进一步分解。其分解规律是以数据为核心。

3. Trans2_I：由CMO对象自动生成SMO对象的规格说明

3.1 程序单元的选择与映射 Ada的程序单元package、generic、procedure、task可用来模拟SMO对象类。有一点需要说明的是对任务的处理方法，由于任务实质上是一种特殊的类型，它不能表达一种抽象数据类型，而且不能作为库单元；为了既保持它的并行特性又能表示SMO对象的抽象数据类型，我们对它进行了必要的处理：将它封装到一个程序包或类属程序包中，随后通过上下文中的with子句就可以使它间接可见。我们称上述程序包、类属程序包分别为：任务型程序包及任务型类属程序包。例如：设有任务T（图9）我们可以将它变换成对应的任务型程序包P__T（图10）。这样就有package、generic package、procedure、任务类型package、任务类型generic package、generic procedure七种结构单元供选择。我们主要从四个方面进行考察：①该class动态引用其他class的数目BeServed；②该class被动态为其他class服务的数目ServeFor，③该class中非预定义的从抽象数据类型数目UDT；④该class是否由PMO对象归纳得到的FromPMO（若是为真）。我们主要使用以下几条规则来选择：

①若not FromPMO　则转到⑥，否则：

②若ServeFor=0　则转⑨；否则：

③若BeServed=0　则转到⑤；否则：

④若UDT=0，则选程序包；否则选择类属程序包。转⑩。

⑤若UDT=0，则选择任务类型PACKAGE，否则选择任务类型类属程序包。转⑩

⑥若BeServed/=0　则转到⑧；否则：

⑦若UDT=0，则选择任务类型PACKAGE，否则选择任务类型的类属程序包。转⑩

⑧若UDT=0，则选择PACKAGE；否则选择类属程序包。转⑩

⑨若UDT=0，则选择procedure，否则选择类属子程序。

⑩停止。

3.2 状态属性表达的映射 对于主抽象数据类型的表达可以简单地映射为：type <CLASS_NAME> is private；如果选择的程序单元是package，则在实现其体的时候给出具体表示；如果是类属单元，则在声明对象时动态给出。对于从抽象数据类型的表达，若是预定义的类型则直接转换成Ada的类型描述，否则将它转换成Ada的私有类型：type T is private。

3.3 消息接口表达的映射 OP：Ti1，Ti2…，Tik，…，Tim→Tik，…，Tim，Tj1，Tj2，…，Tjn转换成Ada的子程序规格：procedure OP(V1：Ti1，…，Vik：in out Tik，…，Vj1：out Tj1；…，Vjn：out Tjn)；

3.4 关系属性表达的映射

①A_Kind_Of关系的映射：A_Kind_of ⟹ Ada_GOODS的Super关系子句；

②Consist_Of关系的映射：Consist_Of ⟹ Ada的嵌套关系；

③Serving关系的映射：Serving⟹ Ada的with关系子句；

④A_Member_Of关系的映射：A_Member_Of⟹Ada的is new关系子句。

根据上述规则我们可以自动实现Trans2_Ⅰ。

4. Trans2_Ⅱ：由CMO对象生成SMO对象的体

Ada_Goods对详细设计及编码的强有力的支持使得求精SMO对象的体非常容易[9]，可望今后能自动生成SMO的体。

六、CMO对象空间S_cmo与SMO对象空间S_smo之间的同构证明

定理：CMO对象空间S_cmo与SMO对象空间S_smo之间同构

证明：设两个代数系统：$DS1=\langle S\_cmo; K, C, S, M\rangle$、$DS2=\langle S_smo; super, embed, with, is_new\rangle$其中：S_cmo、S_smo表示CMO对象空间、SMO对象空间(Ada的程序空间）的结构集。K、C、S、M分别表示CMO空间结构之间的关系：A_Kind_Of、Consists_Of、Serving、A_Member_Of关系。super、embed、with、is new是SMO对象空间中的结构之间的关系（Ada库单元之间的关系）。

对任意的P_1、$P_2\in S\_cmo$，都有：P_1 K $P_2\in S\_cmo$，P_1 C $P_2\in S\_cmo$，P_1 S $P_2\in S\_cmo$，P_1 M $P_2\in S\_cmo$，所以S_cmo封闭。同样可知：S_smo封闭。

在第五节中的程序单元选择规则①⟶⑩实质上建立了一个映射函数，定义它为：

$$H: S\_cmo \longrightarrow S\_smo$$

对任意的CMO对象P_1、$P_2\in S\text{-}cmo$，都有：if P_1 K P_2 then $H(P_1)$Super $H(P_2)$；if P_1 C P_2 then $H(P_1)$ embed $H(P_2)$；if P_1 S P_2 then $H(P_1)$ with $H(P_2)$；if P_1 M P_2 then $H(P_1)$is_new $H(P_2)$；因此DS1与DS2是同态关系。为了求得H的逆函数H′，我们先求得H的类Ada程序$\phi(P, P')$(详见〔4〕)，那么有：

$$H(P)=\{P\in S\_cmo\}\phi(P, P')\{P'\in S\_smo\}$$

因为ϕ中只有选择语句和赋值语句，我们可以形式化地求出$\phi(P', P)$的逆程序ϕ'(详见〔4〕)，因而得到H的逆$H'(P')=\{P'\in S_smo\}\phi'(P', P)\{P\in S_cmo\}$。显然在Ada的库单元一级逆函数$H'$成立，即H是一个满射。因此代数系统DS1与DS2是同构关系，定理得证。

七、结束语与致谢

对于Ada是不是面向对象的语言虽然尚存争议，但是我们的工作表明它对OO的支持是充分的。目前我们已完成了一个逐步求精的面向对象开发系统的原型。今后我们将在已有的理论研究及实现经验基础上，在下列方面做进一步的实现工作：

①生成PMO对象的交互式实现；②PMO到CMO的半自动转换；③CMO到SMO的自动转换。

衷心感谢梁先忠、马晓东两位同志，在系统实现与本文写作过程中都曾得益于他们的帮助，他们的热忱合作和新颖独创的思想是构成本文的源泉。

参考文献

〔1〕Grady Booch, Object_Oriented Development, IEEE Transactions on Software, vol.SE_12, No.12, Feb,1986, PP211_221

〔2〕Wang Zhenyu,Monolingual:One Way Towards the Integrated Software Development Environment, Journal Of Computer Science & Technology,4 (1989), PP184_187

〔3〕王振宇、梁先忠、马晓东，Ada软件开发系统：Ada_GOODS,计算机杂志，1 (1991)

〔4〕陈 辉，SROOM：逐步求精的面向对象方法学，中国舰船研究院硕士论文，1991

〔5〕王振宇，Ada语言程序设计，武汉大学出版社，1987。

〔6〕马晓东、王振宇，面向对象的需求分析，计算机杂志，1(1991)

[7] D.Sannella&A.Tralecki, On Observational equivalence & algebraic specification, Journal of Computer & system science, vol.34, No.2/3, April/-june 1987

[8] 屈延文，形式语义学基础与形式说明，科学出版社，1989

[9] Liang XianZhong,Wang Zhenyu, Ada_Based Support for Abstraction,Encapsulation and Unit Hierarchy,the Proceeding of Tri_Ada'91, 21-25, OCT. 1991, san jose, U.S.A.

附录1：OODL的语法结构

```
1. <OODL> ::= <CLASS><OODL> | <CLASS>
2. <CLASS> ::= "Class" <IDENT>  "is" ["Serving_Class:"<name_list>][<Relation_
feature>":"<Relation_Class>]* ["Abs_Data_Type:" <Type_list>]["Operations:"
<operation_list>]["Variables:"  <Variable_list>]["Equations:"<equation_list>]
3. <Relation_feature> ::= "Consist_Of"|"A_Kind_Of"|"A_Member_Of"]"Constraint"
4.  <Relation_Class>::=<Ident>
5.  <Name_list> ::= <IDENT>"," <name_list> | <IDENT>";"
6.  <Type_list> ::= <Type_name>","<Type_list> | <Type_name>";"
7.  <Operation_list> ::= <Op_declare><Operation_list> | <Op_declare>";"
9.  <Index> ::= "(generator)" | ∈
10. <Parameter_list> ::= <Type_list>
11. <Type_name> ::= <IDENT>          12. <Op_name> ::= <IDENT>
13. <Variable_list> ::= <Var_decl>";"<Variable_list> | <Var_decl>";"
14. <Var_decl> ::= <var_form>":"<Type_name>[ArrayVariableLength]
15. <Var_form> ::= <var_name>","<var_form> | <var_name>
16. <Var_name> ::= <IDENT>
17. <ArrayVariableLength>::="(" <Number> ")"
18. <Number>::=<Digital> | <Digital><Number>
19. <Equation_list> ::= <equation>";"<equation_list> | <equation>";"
20. <Equation> ::= <Op_exp>  "=" <expression>["IF" <bool_exp>]
21. <Op_exp> ::= <op_name>(<parameter_list>)
22. <Parameter_list> ::= <parameter>","<parameter_list> | <parameter>
23. <Parameter> ::= <Op_name>"("parameter")" | <IDENT>
24. <Expression>:=<IDENT> | <OP_name> ( <parameter_list> ) | (<Expression>)
25. <Bool_exp> ::= <Simple_exp><Relation_exp><Simple_exp> | "Not" <Simple_exp>
26.<Simple_exp>::=(<Simple>)|<Simple_exp><Math_op><Simple_exp>|Not<Simple_exp>
27  <simple>::=<Ada简单表达式>
28. <Relation_op> ::="<"|">="|"<"|"<="|"="|"/="|"not in" |"in"|"and" | "or"
29. <Math_op> ::= <Relation_op> | '+' | '-' | '*' | '/'
30. <Ident>::=<Ident><Letter_Digit> | <Letter>
31. <Letter_Digit>::=<Letter> | <Digit>
32. <Number>::=<Digit> | <Digit><Number>
33. <Digit>::='0'..'9'                  34. <Letter>::='a'..'z' | 'A'..'Z'
```