第32卷 第21期 计 算 机 工 程 2006年11月
Vol.32 No.21 Computer Engineering November 2006
·软件技术与数据库· 文章编号：1000—3428(2006)21—0079—03 文献标识码：A 中图分类号：TP311.5

实现 UML 二元关系的规范化方法

王 鑫，袁晓洁
(南开大学信息技术科学学院，天津 300071)

摘 要：由于建模语言和程序设计语言之间缺乏准确的映射机制，UML 二元关系的存在为设计到实现的自动转换造成了许多困难。解决问题的关键在于如何定义一种规范化的方法，用于在程序设计语言中准确地实现 UML 二元关系。该文围绕 UML 二元关系的定义问题，从 4 种基本属性出发，提出关联、聚合与组合关系的形式化定义，并给出在 C++中实现这些关系的规范化模式。
关键词：二元关系；关联；聚合；组合；UML

Normalization Method to Implement UML Binary Relationships

WANG Xin, YUAN Xiaojie
(School of Information Technical Science, Nankai University, Tianjin 300071)

【Abstract】 Since there is no precise mapping mechanism between modeling languages and programming languages, the existence of UML binary relationships causes many difficulties in transforming automatically from design into implementation. The key part of the solution to this problem lies in how to define a normalization method to implement UML binary relationships precisely in programming languages. This paper centers on the definition problem of UML binary relationships. Based on four fundamental attributes, it presents formalization definitions of association, aggregation and composition relationships. Normalization patterns to implement these relationships in C++ are also provided.
【Key words】 Binary relationships; Association; Aggregation; Composition; UML

1 概述

UML 自从成为建模语言的工业标准以来，已经被广泛地应用于面向对象分析与设计领域。然而在软件实现和维护阶段，UML 所发挥的作用仍然相当有限。要做到软件设计与实现之间的自动转换，就必须准确地定义建模语言和程序设计语言之间的映射机制。

虽然 UML 与通用的程序设计语言之间存在着若干相似性；但是 UML 为了建模的需要，提出了许多更加高级的抽象概念，通用的程序设计语言一般不直接支持这些概念。UML 二元关系就是一个典型的例子。在 UML 中，两个类之间的关系可以表现为：关联 (association)，聚合 (aggregation) 或组合 (composition)，然而在 C++或 Java 中这些概念是不存在的，如图 1 所示。

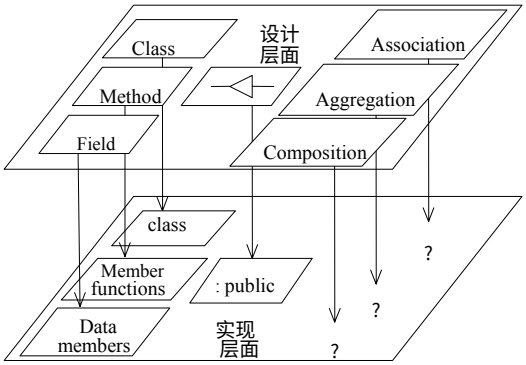

图 1 设计与实现之间的映射问题

文献[1]指出，正是由于 UML 二元关系的存在，才给设计与实现之间的自动转换造成了许多困难。如何将 UML 二元关系准确地映射到程序设计语言，不仅会影响到正向工程中代码的生成，而且还关系着逆向工程中模型的恢复。作为定义 UML 的正式规范，文献[2,3]应该给出关于二元关系的无二义性描述，并提供在程序设计语言中实现二元关系的几种推荐模式。但是这些标准规范文本使用非形式化的自然语言定义 UML 二元关系，并且没有给出任何指导实现的提示信息。

要解决模型与实现之间的映射问题，第 1 种方法是：定义新的程序设计语言，或者对已有的语言进行必要的扩展，使它们直接支持二元关系。但是这种方法失去了 UML 作为通用建模语言的意义。文献[2]指出：“UML 是一种通用的建模语言，它可以适用于所有的对象和组件建模方法，同时可以应用在各种不同的领域和实现平台”。

解决该问题的第 2 种方法是：依靠现有程序设计语言中提供的概念，给出一套实现 UML 二元关系的规范化模式；无论从设计角度，还是从实现角度，都对 UML 二元关系进行准确的、无二义性的定义。本文将使用该方法，给出在 C++中实现 UML 二元关系的规范化模式。

2 二元关系的描述性定义

在面向对象分析与设计领域，不同的研究者对于二元关系的定义和认识也不尽相同。Rumbaugh 在文献[4]中强调了二元关系对于面向对象建模的重要性，然而他没有区分聚合与组合关系，也没有给出各种二元关系的严格定义。Booch

基金项目：天津市自然科学基金资助项目(033600411)
作者简介：王 鑫(1981－)，男，硕士生，主研方向：软件工程，数据库技术；袁晓洁，博士、教授、博导
收稿日期：2006-01-06 **E-mail：**wxnk@eyou.com

在文献[5]中指出，类之间的关系体现了某种语义连接，但他没有提出组合的概念，也没有从形式化的角度定义各种二元关系。

在UML2.0 规范文本[2,3]中，二元关系的定义是：二元关系指两个类之间存在的某种语义联系；聚合是二元关系的一种特殊形式，它表明两个类具有整体和部分的语义联系；组合是一种特殊的聚合，在某个给定的时刻，一个部分对象只能隶属于一个组合对象，组合对象要负责部分对象的创建和销毁工作。UML2.0 规范没有提及任何关于如何实现二元关系的细节问题。

在定义 UML 二元关系时，不仅要从设计和建模的角度出发，还要充分考虑到实现方法。下面首先从设计角度给出关联、聚合和组合的描述性定义。

2.1 关联关系

关联关系是两个类之间在某种概念上的连接。每个类都可以有多个实例参与到一个关联关系中。如果类 A 与类 B 之间存在着关联关系，那么类 A 的实例就具有向类 B 的实例发送消息的能力。

2.2 聚合关系

聚合关系是一种特殊的关联关系，它为两个类赋予了更多的语义，即一个类代表整体，另一个类代表部分。聚合关系具有反对称性，如果类 A 与类 B 之间存在聚合关系，那么类 B 与类 A 之间就不能存在聚合关系。

聚合关系具有反对称性是它区别于关联关系的重要特征。它表明整体可以包含部分，但部分却不能包含整体。实际上，如果允许聚合关系具有对称性，那么它就会退化为关联关系。

2.3 组合关系

组合关系是一种特殊的聚合关系。在组合关系中，整体对象控制着部分对象的生命期。在某个给定的时刻，一个部分对象只能隶属于一个整体对象。

组合关系具有两种特性，使它区别于聚合关系：(1)生命期，整体对象包含着部分对象，整体对象销毁之前，部分对象必须首先销毁；(2)所有权，整体对象拥有部分对象的排他性所有权，一旦部分对象隶属于某个整体对象，它就不能再隶属于其他整体对象。

3 二元关系的属性

3.1 排他性（Exclusivity）

在某个给定时刻，如果类 Y 的一个实例与类 X 的一个实例之间存在关系 R，那么在同一时刻，类 Y 的这个实例就不能再与其他实例之间存在关系。这种属性称为类 X 对类 Y 具有排他性。设 B 表示集合{true,false}，Class 表示类的集合，则排他性可以定义为

$$EX：Class\times Class\rightarrow B$$

对于任意给定的两个类 X 和 Y，若 EX(X,Y)＝true，则表明只要类 Y 的一个实例已经与类 X 的一个实例建立了某种关系，就不能再与另外的实例（包括类 X 的或其他类的）建立关系。若 EX(X,Y)＝false，则表明类 Y 的一个实例不但可以与类 X 的一个实例建立某种关系，而且还可以在同一时刻与其他实例建立关系。

3.2 发送消息（Message Send）

两个类之间存在着某种二元关系，通常意味着一方可以向另一方发送消息，从而引发特定操作的执行。在一般情况下，类可以通过数据成员、成员函数的参数或成员函数中的局部变量发送消息。若 mode 是消息发送方式的集合，则

$$mode＝\{member,parameter,local\}$$

其中，member 表示数据成员，parameter 表示成员函数的参数，local 表示成员函数中的局部变量。两个类之间发送消息的属性定义为

$$MS：Class\times Class\subseteq mode$$

对于任意给定的两个类 X 和 Y，MS(X,Y)的值表明类 X 的实例通过哪种方式向类 Y 的实例发送消息。例如，MS(X,Y)＝member 表示类 X 将类 Y 的对象作为数据成员，并通过该成员向类 Y 发送消息；而 MS(X,Y)＝∅ 表示类 X 无法向类 Y 发送消息。

3.3 生命期（Lifetime）

生命期是指一个类的实例从创建到销毁所持续的时间。类 X 的实例与类 Y 的实例之间可能存在着生命期上的依赖关系。二元关系的生命期属性定义为

$$LT：Class\times Class\rightarrow when$$

Class 表示类的集合，when＝{before,after}。对于任意给定的两个类 X 和 Y，如果在销毁类 X 的实例之前，要首先销毁类 Y 的实例，则 LT(X,Y)＝before；如果在类 X 的实例被销毁之后才能销毁类 Y 的实例，则 LT(X,Y)＝after。如果类 X 与类 Y 的实例不存在生命期上的依赖关系，则记作 LT(X,Y)∈when。

3.4 多重性（Multiplicity）

对于二元关系来说，多重性是指类 X 的一个实例可以在一个关系中对应于多少个类 Y 的实例。二元关系的多重性定义为

$$MU：Class\times Class\subset N$$

其中，N＝{0,1,2...}。为简单起见，可以使用一个区间来表示多重性。例如，在“汽车–轮胎”关系中，一辆汽车（Car）通常有 4 条轮胎（Tyre），有时还可能配有一条备用轮胎，因此 MU(Car,Tyre)＝[4,5]。

4 二元关系的形式化定义

4.1 关联关系

类 X 与类 Y 之间存在的关联关系 AS(X,Y)定义为

$$\begin{aligned}AS(X,Y)＝\ &(EX(X,Y)\in B)\wedge(EX(Y,X)\in B)\wedge\\ &(MS(X,Y)＝mode)\wedge(MS(Y,X)＝\varnothing\ )\wedge\\ &(LT(X,Y)\in when)\wedge(LT(Y,X)\in when)\wedge\\ &(MU(X,Y)＝[0,+\infty])\wedge\\ &(MU(Y,X)＝[0,+\infty])\end{aligned}$$

只要类 X 的实例能够通过某种方式向类 Y 的实例发送消息，类 X 与类 Y 之间就存在关联。为简单起见，仅考虑单向二元关系，在定义中限制类 Y 的实例向类 X 的实例发送消息。关联关系对于排他性、生命期和多重性没有特殊要求。以下代码段给出了使用 C++实现关联关系的一种参考模式。

```
class Y{
public:
 void g(){}  //MS(Y,X)＝∅
};
class X{
public:
 void f(Y& y){
 //MS(X,Y)＝parameter⊂ modes
  y.g();
 }
```

```
};
int main(){
 //对象 x 和 y 具有显式的名称，
 //可以在其他位置被引用。
 //EX(X,Y)＝EX(Y,X)∈B
 X x;   Y y;
 //对象 x 和 y 之间不存在生命期上的依赖。
 //LT(X,Y)＝LT(Y,X)∈when
 //MU(X,Y)＝MU(Y,X)＝[0, + ∞]
 x.f(y);
 y.g();
}
```

4.2 聚合关系

类 X 与类 Y 之间存在的聚合关系 AG(X,Y)定义为

AG(X,Y)＝(EX(X,Y)∈B)∧(EX(Y,X)∈B)∧
(MS(X,Y)＝member)∧(MS(Y,X)＝∅)∧
(LT(X,Y)∈when)∧(LT(Y,X)∈when)∧
(MU(X,Y)＝[0,+∞])∧
(MU(Y,X)＝[1,+∞])

聚合是特殊的关联关系，它不仅要求一个类的实例可以向另一个类的实例发送消息，而且要求两个类之间应该体现出整体和部分的语义联系。在 C++中，实现聚合关系的一种常见模式为：整体类通过数据成员向部分类发送消息。例如，在以下代码段中，整体类 X 将指向部分类 Y 的指针定义为其数据成员，从而建立起聚合关系 AG(X,Y)。

```
    class Y{
public:
 void g(){}  //MS(Y,X)＝∅
};
class X{
public:
 //MU(Y,X)＝[1,1]⊂ [1, + ∞]
 X(Y* y=0):y_(y){}
 Y* gety() const {return y_;}
 void sety(Y* y){y_=y;}
 void f(){
  y_->g();   //MS(X,Y)＝member
 }
private:
 //MU(X,Y)＝[0,1]⊂ [0, + ∞]
 Y* y_;
};
int main(){
 Y y;        //EX(Y,X)∈B
 X x(&y);    //EX(X,Y)∈B
 x.f();      //LT(X,Y)∈when
 y.g();      //LT(Y,X)∈when
}
```

4.3 组合关系

类 X 与类 Y 之间存在的组合关系 CO(X,Y)定义为

CO(X,Y)＝(EX(X,Y)∈true)∧(EX(Y,X)∈false)∧
(MS(X,Y)＝member)∧(MS(Y,X)＝∅)∧
(LT(X,Y)∈before)∧(LT(Y,X)∈after)∧
(MU(X,Y)＝[1,+∞])∧(MU(Y,X)＝[1,1])

组合关系体现出更加严格的整体和部分语义，因此它是一种增强型的聚合关系。组合除了要满足聚合的所有属性之外，还要满足整体类对部分类在所有权上的排他性，以及部分类对整体类在生命期上的依赖性。以下代码段给出了使用 C++实现组合关系的一种参考模式。

```
    class Y{
public:
 void g(){}  //MS(Y,X)＝∅
};
class X{
public:
 //MU(Y,X)＝[1,1]
 X(Y* y):y_(y){}
 void f() {
  y_->g();   //MS(X,Y)＝member
 }
 //LT(X,Y)＝before∧LT(Y,X)＝after
 ~X(){delete y_;}
private:
 //MU(X,Y)＝[1,1]⊂ [1, + ∞]
 Y* y_;
};
int main(){
 //EX(X,Y)＝true∧EX(Y,X)＝false
 X x(new Y);
 x.f();
}
```

一方面，整体类 X 在析构函数中释放部分类 Y 的对象所占据的存储空间，这样就能确保在整体对象 x 销毁之前，部分对象 y 首先销毁；另一方面，在 main 函数中使用语句“X x(new Y);”建立组合关系 CO(X,Y)，目的是使部分类 Y 的实例不具有显式的名称，其他对象无法引用该实例，从而可以确保它只隶属于实例 x。

5 总结

本文介绍了用于描述 UML 二元关系的 4 种基本属性，重点论述了关联、聚合与组合关系的形式化定义，并给出了在 C++中实现二元关系的规范化参考模式。

使用规范化方法实现 UML 二元关系将有利于完善建模语言和程序设计语言之间的映射机制，从而做到设计与实现之间的自动转换，使 UML 在实现和维护阶段发挥出其应有的作用。

参考文献

1 Guéhéneuc Y G, Hervé A A, Rémi D, et al. Bridging the Gap Between Modeling and Programming Languages[R]. Computer Science Department, École des Mines de Nantes. Technical Report: 02/09/INFO, 2002.

2 Object Management Group. UML 2.0 Infrastructure Specification[Z]. OMG Adopted Specification, 2003-09-15.

3 Object Management Group. UML 2.0 Superstructure Specification[Z]. OMG Adopted Specification, 2003-08-02.

4 Rumbaugh J, Blaha M, Premerlani W, et al. Object-oriented Modeling and Design[M]. Englewood Cliffs, New Jersey: Prentice-Hall, 1991.

5 Grady B. Object-oriented Analysis and Design with Applications[M]. Redwood City, CA: Benjamin Cummings, 1993.