计算机科学1997 Vol. 24 №. 5

Fusion 方法导论

An Introduction to Fusion Method

肖金声　黄思曾　　陈仲驹
（中山大学计算中心　广州510275）（广东省计算中心）

摘　要　This paper introduces a second-generation object-oriented development method- -Fusion method.

关键词　Object-oriented, Development method, Analysis, Design

继第一代 OO 开发方法[1-10]之后，94年出现了头一个第二代 OO 开发方法——Fusion 方法[11]。同以往的许多开发方法一样，Fusion 方法也把开发过程划分为分析、设计和实现三个阶段，但它的基本思路与众不同。本文介绍其分析与设计部分，特色尽在其中。我们假定读者熟悉面向对象技术中的基本概念和术语。

一、分析

Fusion 的分析阶段产生两类模型：①对象模型，被用来捕捉问题域中存在的概念以及它们之间的关系，以确定系统中信息的静态结构。②界面模型，用于表达系统的行为。

1.1　对象模型

对象模型表达为应用域中类(class)的联系图，联系的方式有三种：关系、聚集和一般与特殊。类的图形表示为：

其中类名标识符以大写字母开头。

1.1.1　关系　是对象的元组。二元组是二元关系，多元组是多元关系。关系应有关系名，以小写字母开头。关系以菱形表示，如图1。关系也可以有属性(如果必要)。

为了限定关系中各方对象的数量，Fusion 采用四种形式的基数(cardinality)表示：数(如6)，范围(如1..4)，星号 *（表0或多）和加号 +（表示1或多）。

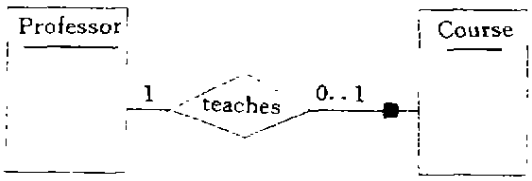

图1　关系的图形表示

在涉及关系的类中，如果每个对象都必须参予该关系，可用弧上的全体符号 ■ 来指明。例如图中的 ■ 表明每门课程均需有人数。

有时，对象模型中有些特殊的特性不能用基数等有限地符号表达，可能需要进行一定的文字注释，若用谓词的形式写成不变式，则较为简洁。

在某些情况下，同一类中的对象会具有某种关系，例如 Person 类中的家长关系，也可以同样用关系图表示，但在关系两边应该注以大写字母开头的角色名。如图2所示：

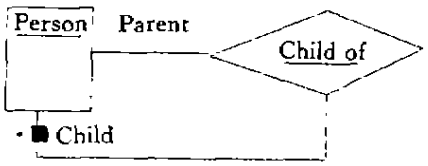

图2　同类中对象的关系

三元或多元关系也类似表示，不赘述。

1.1.2　聚集　是一种构造机制，用以由成分类构造聚集类。聚集的图形表示如图3。成分类的框嵌套在聚集类的框内，成分类的基数写在成分框的外边，其缺省值为1。

*）国家自然科学基金资助项目

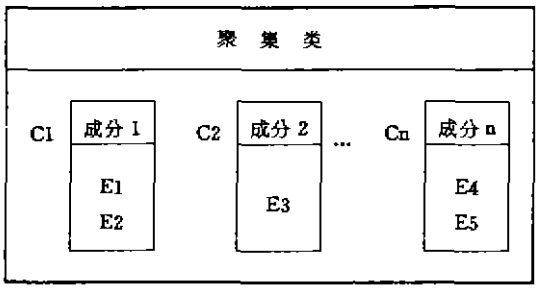

图3 聚集的图形表示

1.1.3 一般与特殊 一般化允许从 n 个子类(subtype)抽取公共特性而形成超类(supertype);特化则相反,由超类确定更特别的版本而形成新子类。

超类的属性和关系被其所有子类继承。每个子类可以有另外的属性,及参预另外的关系。一般化的重要特性是,子类的所有对象都属于超类。一般化的记号是联系超类和子类的空三角,如图4。

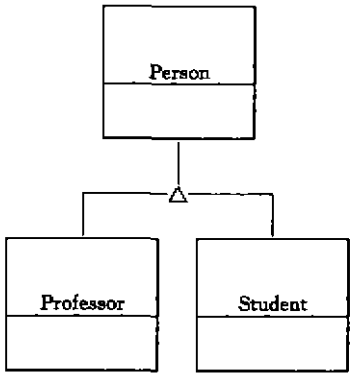

图4 超类与子类

这里有一种特殊情况:子类分割超类,亦即各子类无公共对象,且所有子类对象的并就是超类的对象集。为表达这种特殊情况,只需将空三角改为实三角。

Fusion 规定,子类只能扩展而不能改动超类的特性,因而子类的对象才总是超类的对象。许多 OOPL 都不符合这一要求,可见,一般与特殊对应于 OOPL 中继承的受限使用。

把应用域中的所有对象类按以上三种联系方式连接起来就形成对象模型。如果对象模型的图形表示大而乱,允许适当地破分为若干子图,或者省略嵌套中的某些内容,而对象模型恰是所有子图的并。

1.1.4 系统对象模型 对象模型中的类和关系同时指明了属于系统环境和系统本身的概念。系统对象模型则是由对象模型排除属于环境的所有类和关系而余下的子集。这种边界圈定必须保持系统对象模型仍是个完好模型,即满足:若某个关系属系统对象模型,则涉及该关系的类也必须属于系统对象模型。

1.2 界面模型

界面模型描述系统的行为。无论是人还是别的硬软件系统,所有能与系统通讯的活动实体都被称为 agent。系统的环境就是与系统通讯的 agent 集。

系统与其环境通讯的原子单位是事件(event)。输入事件由 agent 发送给系统;输出事件由系统发送给 agent。这种通讯是异步的,发送方不等待事件被接受。发送方可能随事件提供数据值和对象。

当系统接受事件时,会引发系统状况的改变和发送输出事件,其效果由随之提供的值和系统当时的状况决定。一个输入事件连同其效果被称为一个系统操作。

系统的界面是它能接受的系统操作的集合和它能输出的事件的集合。

一般地说,一个系统不能以任意的模式同环境交互,可允许的通讯序列称为系统的生命周期。

为了捕获系统行为的不同方面,界面模型分两种:操作模型和生命周期模型。

1.2.1 操作模型 以系统状况的改变和发送的输出事件来说明系统操作的行为。一个系统操作可能会:创建新的类实例;改变现存对象的属性值;给某关系加入或删去某个对象元组;发送事件给某 agent。

操作模型由一系列模式来表达。每个系统操作至少有一个模式。模式用结构化的文字表示:

Operation:	名字
Description:	文字
Reads:	项表
Changes:	项表
Sends:	agent 与事件表
Assumes:	条件
Result:	条件

操作名是自由格式的标识符;文字是系统操作非形式的简要描述;Reads 的项表列出操作可能访问但不改变的所有量。标识符前若加关键字 Supplied,则为参数;Changes 的项表列出该操作可能改变的量。标识符前若有关键字 New,表明新创造一个对象;Sends 子句分组列出该操作可能发送的输出事件和接受 agent;Assumes 子句的条件是用 and 连接的逻辑谓词,表示操作时必须为真的前置条件,缺省值

是 true；Result 子句中的条件是确定后置条件的谓词，操作完成时必须为真。可见，操作模式是对操作的黑盒描述。

1.2.2 生命周期模型 生命周期表达式用于确定系统在其生命期内可能参预交互的允许序列，亦即确定通讯模式。若某一时刻遇到了不允许的输入事件，系统将拒绝该事件，保持状况不改变。生命周期表达式的语法和语义为：

·原子式 输入或输出事件的名字，输出事件前缀以#。

·运算符 若 x 和 y 是生命周期表达式，则 x·y，表 x 后紧接 y；x|y，表 x 或 y 之一出现；x*，表 x 的零次或多次出现；x+，表 x 的一次或多次出现；[x]，表 x 是可选的；x||y，意指任意夹杂 x 和 y 的元素。这些也是生命周期表达式。

·替换 可以用名字来替换表达式：Name＝生命周期表达式。Name 可用于其他表达式，但替换不能是递归的。

·运算符优先性 按递减的次序，优先性是：[ ]，*，+，·，|，||。可以用圆括号改变运算符优先性。生命周期被实现为有穷自动机。

1.2.3 两个模型的互补 系统的行为由生命周期模型和操作模型共同描述。操作模型描述单个操作的有关数据和效果，生命周期模型则决定操作的合法次序。

生命周期决定事件的可接受性，而前置条件则决定系统操作的效果。例如，没有在银行开户，就不能存取款，这由生命周期描述；取款时透支则属前置条件不满足。

一般来说，一个操作只有在生命周期允许且前置条件为真时才能正常执行，参阅下表。

系统反应 / 前置条件	true	false
生命周期		
接受 拒绝	行使操作 不理	不确定 不理

1.3 数据字典

在 Fusion 的各开发阶段都需要构造或使用数据字典。它是定义术语和概念的中心资源库。在交叉检查模型的完备性和一致性时，数据字典起着核心作用。当然，它还能帮助非开发者理解系统。数据字典取以下格式：

名 字	种 类	描 述
条目名称	类，系统操作，agent，事件等	定义或解释该条目的文字

构造数据字典应遵从命名易理解、避免别名、不应重复模型内已含信息。

1.4 分析步骤

系统分析的起点是系统用户用文字写出的非正式需求文件。分析者据此展开调查和分析工作。分析工作的步骤如下：

1)为应用域开发对象模型。

2)决定系统边界。①识别 agent，系统操作和事件。向对象模型加入边界而产生系统对象模型。

3)开发界面模型。①开发生命周期模型。②开发操作模型。

1.4.1 对象模型的开发 开发工作围绕识别对象类及其关系进行。候选类的所有可能来源是问题域中的物理对象、人或组织、抽象概念。识别对象类当然还包括确定属性。

关系是一种语义关联。其可能来源包括通讯、物理相关、包含和动作。识别关系时，应同时考虑到有关类的基数和可能的不变式。然后，聚集、一般与特殊的结构关联也必须注意识别。

把这些识别的结果用约定的记号以图形的形式表达出来，就得到对象模型。子图的划分依赖于分析员的经验。所有的类和关系都应写进数据字典。

1.4.2 系统界面的确定 确定系统边界在于确定 agent、系统操作和事件。关注问题域行事习惯的脚本(scenario)是一种有效方法。脚本是为某种目的，在 agent 和系统之间流动的事件序列。脚本被表示成时序图，以表明系统操作和流向 agent 的事件的时间顺序。时序图的形成与 OMT 中的相同。

为系统要执行的所有任务都写出了脚本并画出时序图之后，agent 和事件都显得十分清楚。由此，输入事件、输出事件和系统操作也容易确定。这时，系统的边界也就自然地显现出来。记住，系统对象模型由对象模型提炼出来。它本身也是个完好的对象模型。

1.4.3 界面模型的开发 生命周期和操作模型的开发顺序虽不必固定，但以先开发生命周期模型为好。

1)生命周期模型

生命周期模型通过开发生命周期表达式而形成。一个生命周期表达式能确定一组脚本，不象时序图只能表明单个脚本。形成步骤是：①综合各脚本，

以形成命名的生命周期表达式。②联合各生命周期表达式，以形成生命周期模型。

根据语法，单独的输入或输出事件本身就是个生命周期表达式。把单个脚本用表达式来表达也不困难，关键在于综合，使得最后的总表达式(即生命周期模型)既能概括所有脚本，又没有多余的东西。这有赖于分析者的经验。

2)操作模型

操作模型确定每个系统操作的语义。每个操作模式中的 Assumes 和 Results 子句应确实表明系统客户对该操作的需求。分析者还应保证最后产生的操作模式是可满足的，即对满足 Assumes 子句的所有初始值，操作的最后值均满足 Results 子句。操作模式的开发步骤是：

A. 开发 Assumes 和 Results 子句

·以 Results 的子句分开描述结果的每个方面。

·用生命周期模型去寻找 Results 中的输出事件。

·Results 不允许不想要的值，需检查。

·给 Assumes 和 Results 加进贴切的不变式。

·保证 Assumes 和 Results 是可满足的。

·将系统操作和事件列入数据字典。

B. 从 Results 和 Assumes 抽取 Sends，Reads 和 Changes。

1.5 分析模型的检查

检查分析模型有两个方面：完备性和一致性。模型是完备的，如果它捕捉了问题中全部有意义的抽象。模型集是一致的，如果这些模型明里暗里彼此均不矛盾。

1)以需求为背景的完备。仔细重读需求文档，同客户议决突出的论点，澄清客户对系统要求的东西，然后检查。

·是否所有可能的脚本都被生命周期模型复盖了。

·是否所有系统操作都用操作模式确定了。

·是否所有静态信息都被系统对象模型捕获了。

·数据字典中的任何其他信息(如技术定义和不变式)。

2)简单一致性。这些检查涉及各分析模型间的相迭部分。

·操作模型中提到的所有类、关系和属性是否都出现在系统对象模型中。所有其他概念(如谓词)都必须在数据字典或其他引用源中定义。

·系统对象模型的边界同生命周期模型给出的系统界面一致。

·生命周期模型中的所有系统操作是否都有操作模式。

·所有模型中的标识符在数据字典中是否都有相应条目。

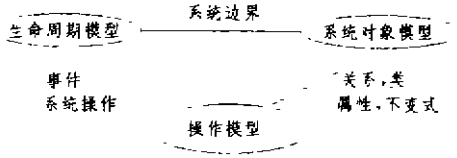

图5 检查模型示意

3)语义一致性。这些检查试图保证各模型的含义一致。

·生命周期模型与操作模型中的输出事件必须一致。

·操作模型必须维持系统对象模型的不变式。

·用操作模式人工检查脚本。

选择脚本的用例，并确定应当出现的每次状况改变。然后"执行"脚本，用操作模式去确定每个系统操作的行为，检查结果是所期望的。

二、设计

分析阶段的系统对象模型抽象地描述了系统中的类和类间拥有的关系。设计阶段的相应输出是含有同样信息并维持同样关系的面向对象软件结构。出自分析员的操作模型确定用户预期的系统功能。设计的相应输出是实现操作模型的交互对象组合。Fusion 方法要求在设计阶段开发四个模型：

·对象交互图 描述系统运行时对象如何交互，以支持操作模型所定义的功能。

·可视图 描述对象的通讯路径。

·类描述 提供系统中所有类的接口、数据属性、对象引用属性和方法的定义。

·继承图 描述类/子类的继承结构。

此外，继续使用和/或完善数据字典。

2.1 对象交互图

每个系统操作对应一张对象交互图。它把该系统操作的功能分配给一组合作的对象而实现该操作。这时，会确定分析阶段未曾定义的对象方法接口。

2.1.1 记号 对象交互图是一组用矢线连接的框。框表示设计对象，其中有对象名及所属类名。矢线表示消息传递。每图有一个框是控制元，有一不始自任何别的框的矢线指向它；此矢线用该对象交

互图要实现的系统操作的名字作标记。图中其他框都是合作元。其余的矢线均有起始框。这些矢线用消息名作标记，可以附以适当的参数。矢线的连接使得任何框都能从控制元循一条矢线路径到达。

设计对象间的消息序列决定图中各设计对象的联合功能行为，从而确定系统操作的高层功能的实现。

每个对象交互图在数据字典中均有相关的文字说明。

设计对象是分析期间所识别的对象的延伸，应有一个方法接口，各对象通过消息进行通讯，以援引这些方法来执行任务。设计对象对应于高层设计部分，它可能需要进一步的分解，用一组对象来实现。

在对象交互图中，单个对象以实线框表示，而一组同类对象则用虚线框表示。

消息传递是有向的点到点通讯，以方法的调用来实现。矢线的起始点是消息的发送方，到达点是消息的接收方。有时也把发送方称为客户，把接收方称为供方。矢线的存在表明发送方可能发送消息，而不是一定发送消息。接收方得到消息之后，便引用其接口中对应的方法。

对一组同类对象发送消息，意味着对其中每个对象发送同样消息。这时，还可以在消息名之下加一个选择谓词，表明该消息只发送给其中使此谓词为真的对象。其缺省值为 true。

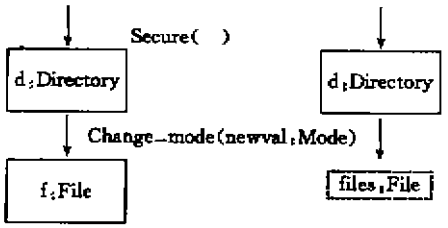

图6 对象交互图

若某对象前冠以关键字 New，表明该对象在执行中生成。就是说，首先必须有特定消息 Creat 发送给新对象，并附以适当的初始化参数。否则，该对象处于不确定状态，不能接受其它消息。

2.1.2 对象交互图的开发 设计期间，须为每个系统操作建立一个对象交互图。这涉及四个主要步骤：

1)识别操作中涉及的相关对象

2)确定每个对象的作用。①识别控制元，即对该系统操作负责的对象。②识别所涉及的合作元。

3)决定对象间传递的消息

4)在对象交互图中记录已识别的对象如何交互。

设计对象交互图的主要依据是分析产生的操作模型的模式，那里既指明了读和访问的对象，也有功能的简单描述。

有时控制元并非明显是某个对象，可能有多种选择。这时，以不同的对象作控制元，就会得到很不同的对象交互图，设计者必须权衡，作恰当的决定。

设计是个反复进行和逐级分解的过程。所得的对象交互图就提供了算法结构的直观表示，而方法也可能需要逐级分解。

对设计的结果必须进行两种基本检查：①与系统说明的一致性。检查系统对象模型中的每个类是否在至少一个对象交互图中出现过。②功能效果的核实。检查每个 Results 子句是否都满足了。

2.2 可视图

为使客户能发送消息给供方，供方对客户来说必须是可访问的(即所谓可视的)。可视图的目的就是确定系统中类的引用结构。这个任务是为每个类验明：①类实例需要引用的对象。②对引用对象的适当引用种类。

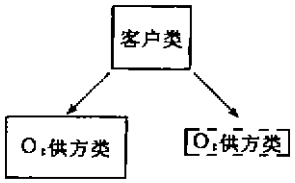

图7 可视图示意

2.2.1 记号 可视图的成分是客户框、供方框、和从客户框连到供方框的可视性矢线。客户框表示须作访问引用的类，其矩形框内含该类的名字。供方框表示被访问的对象，其矩形框内含该对象及其类的名字。客户框有四种框边：单边、双边、虚边、实边。可视矢线则有两种，实矢线和虚矢线，分别代表不同的语义。

2.2.2 可视关系 在考虑类的引用结构时，必须作出某些设计决定：

·引用期 对单个对象的动态引用(虚矢线)和对一组对象的持续引用(实矢线)

·供方可视性 排他引用(双框)与共享(单框)

·供方受囿性 若供方生命期囿于客户，在可视图上把供方框划在客户框内。

·引用的可变性 在供方名前冠以关键字 Constan 或 Variable。

2.2.3 可视图的开发 可视图由对象交互图按以下方式开发：

1)检查全部对象交互图。对于从一个对象到另一个对象的每条矢线，发送方的类要求能引用其接收对象的可视性。

2)对于每个类 C，考虑出自 C 的实例的所有矢线。以 C 为客户框，用可视矢线连向供方框。

3)为每条可视矢线确定适当的可视性信息：①引用生命期；②供方可视性；③供方受围性；④引用可变性。

此外，对象交互图矢线上的表(即方法名)必要时可能被拷贝到可视图上。

2.2.4 检查 可视图完成后，有三个重要的检查：

1)与分析模型的一致性。对于系统对象模型中的每个关系，我们期望在对应的设计类之间有一条可视路径。

2)相互一致性。检查排他的供方对象不被多于一个客户引用。

3)完备性。看看对象交互图中确定的所有消息传递是否都在可视性图中实现了。

2.3 类描述

类描述中的信息先从系统对象模型、对象交互图和可视图中整理。每个类有一个类描述。这时，每个类的方法、某些数据属性、以及对象值属性均被确定。当继承图产生之后，再在类描述中加进反映继承结构的信息。

2.3.1 记号 类描述由以下信息组成：类名、直接超类、类的属性以及该类提供的方法。描述记号是：

```
Class<类名>[isa<超类名表>]
//对每个属性
  [attribute][变动性]<属性名>:[<共享性>]
  [<受围性>]<类型>
  ⋮
//对每个方法
  [method]<方法名><参数表>[:<类型>]
  ⋮
endclass
```

类描述以关键字 class 开头，以关键字 endclass 结尾。Class 之后紧随类名。如果有直接超类，便列在关键字 isa 之后。

每个属性以可选关键字 attribute 引进。属性名与其类型之间以冒号分隔。对象属性的可视信息用适当的关键字表明。

<变动性>::=constant\variable

<共享性>::=shared\exclusive

<受围性>::=bound\unbound

三者的缺省值分别是 Variable、Shared、unbound。

每个方法以可选关键字 method 引进。方法名和参数表的列出如同一般程序语言中的过程和函数。如果有回送值，则在最后加冒号及其类型。

2.3.2 类描述的开发 类描述是编码的依据，指明类的内部状况和外部接口，必须为每个类逐个建立类描述。

1)方法得自对象交互图，该类的对象参预其中的所有对象交互图要一并考虑。对该类对象发送的消息被整理在一起，形成该类的接口，对象交互图上所标出的消息提供方法名和所需参数的类型。方法的其他参数从可视图得来。

2)数据属性的来源是系统对象模型和数据字典。系统对象模型确定类及其属性，数据字典中的方法描述也可能产生属性。

3)对象属性从该类的可视图抽取。所有的持续引用都用对象属性来实现，而类描述指明可视引用设计成哪一种，即 constant/variable，bound/unbound，shared/exclusive。

4)类描述的最后方面是继承信息，继承依赖在继承图之后记述。对于每个类，从继承图整理它的直接超类是容易的。

2.3.3 类描述的检查 尽管类描述的信息内部都是从别处抄来的，也仍要作下述检查。

1)数据属性。检查系统对象模型中的所有数据属性是否都已记述了。

2)对象属性。检查所有可视引用是否都已记述了。

3)方法和参数。检查对象交互图中的所有方法是否都已记述了。

4)继承。检查是否所有被继承的超类都已被记述了。

2.4 继承图

分析期间的一般与特殊是应用域中的语义关系，是域模型的特性，是描述性的。一般和特殊可直接用继承来实现。设计期间的继承是系统的特性，是指定的。

2.4.1 记号 对继承所用的记号同对一般与特殊所用的对象模型记号一样，不再重复。

2.4.2 继承图的开发 凡为把类结构组织成

抽象层次而引进新类时，就要给它建立继承图。这一步的输入是系统对象模型、对象交互图、可视图和类描述。

1）系统对象模型——一般与特殊。系统对象模型的一般与特殊为构造继承结构提供了明显的出发点。特殊的分析类是子类，而一般的分析类是超类。也可能为效率或代码重用的原因而引进继承结构，但须小心做到，新引进的继承不干预初始子类型语义结构。

2）对象交互图和类描述——公共功能。每个对象交互图确定一个操作的功能，并推动实现时所涉及对象的外部接口的确定。这些类接口说明记载在每个类的类描述中，交叉检查每个类描述就能发现公共功能，这些功能可以提取来建立新抽象类。

3）可视图——公共结构。以其他可视图为背景交叉检查每个可视图以提取公共结构。如果两个类有共同的引用结构，就有可能确定一个定义该共同引用的抽象类。

最后，给定一个稳定的继承图结构后，设计者应当更改类描述，以反映这些新抽象类和继承结构。

2.4.3 继承图的检查 继承图必须用下列模型来检查：

1）系统对象图。通过检查，确保分析时的一般与特殊关系都用继承机制维持下来了。

2）对象交互图。检查对象交互图中的所有类都在继承图中表示了。（这要求很高）

3）可视图。检查可视图中类的共同结构是否都确定了抽象类。

4. 类描述。检查新的类描述是否捕捉到了初始类描述的所有公共方法，以及是否考虑到继承图。

三、简评

与第一代 OO 开发方法相比，Fusion 确实进了一大步。初步比较一下，就很容易发现它的一些优点：

1. 更纯的面向对象。在 Fusion 中，不容易察觉传统方法的影子。

2. 对系统行为的描述有说服力。系统行为的描述一直是第一代 OO 开发方法的弱项，而 Fusion 以操作模型和生命周期来描述，显得自然而实际。

3. 平滑过渡。分析、设计、实现三大阶段的过渡比较平滑，每个阶段内的步骤连贯性和一致性都比较好。

但是，Fusion 也有不大令人满意之处：

1. 只重视问题域。作为软件系统的设计，有一些因系统本身的需要而产生的问题，如人机界面，用户管理……等，是不应忽视的。

2. 不尽合理。例如，系统的生命期似乎是某类对象的生命期；是否所有类都必须在继承图中出现，值得商榷；为定系统边界而使用脚本，但脚本本身是 agent 与系统之间的事件序列，有些矛盾。

3. 未涉及系统的层次结构。开发大系统时，子系统和模块的划分是不可少的。Fusion 中见不到这些，本来可视图是与模块划分有关的，但不见作者有何展开。

参 考 文 献

[1] S. C. Ballin. An object-oriented requirements specification method, CACM. 32(5)1989

[2] P. Coad and E. Youron, Object-Oriented Design, Prentice Hall, Englewood Cliffs, N. J., 1991

[3] G. Booch, Object-Oriented Design with Applications, Benjamin Cummings, Redwood City, CA. 1991

[4] J. Ivari, Object-Oriented informations systems analysis, A framework for object identification. IEEE Trans., 1991

[5] G. Kappel, Using an object-oriented diagram technique for the design of information systems, Dynamic Modeling of Information Systems. H. G. Sol et al., Eds, Elsevier Science Publishers B. V., 1991

[6] K Lieberherr et al., Object-Oriented Programming: An objective sense of style. In Proc. of OOPSLA '88 Conf., San Diego, Calif. Sept., 1988

[7] B. Meyer, Object-Oriented Software Construction. Prentice Hall, Englewood Cliffs, N. J., 1988

[8] J. Runbangh, et al., Object-oriented modeling and design, Same to [2]

[9] S. Shaer and S. J. Mellor, Object-Oriented Systems Analysis: Modeling the Words in Data, Same to [7]

[10] R. J. Wirfs-Brock, et al., Designing Object-Oriented Software, Prentice Hall, Englewood Cliffs, N. J., 1990

[11] D. Coleman et al., Object-Oriented Development. The Fusion Method, Prentice Hall, Englewood Cliffs, N. J., 1994