系统模型开发的形式化技术

冯玉琳　桂自强　丁茂顺

(中国科学技术大学)　(中国科学院软件所)

THE FORMAL TECHNIQUES FOR SYSTEM MODELING DEVELOPMENT

Feng Yulin and Gui Ziqiang

(*University of Science and Technology of China*)

Ding Maoshun

(*Software Institute, Academia Sinica*)

ABSTRACT

The paper contributes some formal techniques for the system modeling development discussed in [1]. Based on a kind of multi-sorted first order logic, the language CML is used to specify system models conceptually, and the analyser CMA is followed to check their static consistency and dynamic temporal.

摘　要

本文是[1]中内容的继续，讨论系统模型开发的形式化技术。基于多类一阶逻辑设计的概念模型语言CML可用来描述系统模型，并可用CMA对模型描述进行静态一致性和动态时序特性的检查。

1989 年 8 月 20 日收到，1989 年 11 月 20 日定稿。

§1. 系统模型开发

我们在[1] 中讨论了软件工程的模型化开发方法，模型是现实世界中存在的实在系统的抽象。软件需求是用户根据现实世界中实在系统的经验，由软件人员和用户共同制定的互相约束的说明规范。自顶向下的软件工程设计方法，例如数据流分析和设计，要求在软件开发之始，就要制定一份软件需求规格说明，这在许多情况下并不是一件容易的事。首先，用户本身对系统要求常常也不很明确，再由于软件人员和用户之间知识背景的差异，经常发生因理解不同而导致的信息转移错误。软件开发过程的深入常常要求对最初制定的软件需求规范进行修正、扩充和改善。问题暴露得愈晚，修正工作所花费的代价也就愈大。

与自顶向下的设计方法不同，软件工程的模型化开发方法要求首先建立系统模型。这样的模型是在软件人员和用户共同理解的基础上，由软件人员建立的。模型刻划所要求设计软件系统的主要特征。除了模型本身的一致性以外，模型还应该是可执行的，从而可以检查模型的时序动态特性。通过系统模型可以加深对所要求设计的系统的认识，对不恰当之处进行反复修正，直到满意为止。最后再对系统模型进行精化、扩充，在一定的实现环境下具体实现为所要求的软件系统。模型化开发过程示意如图1。

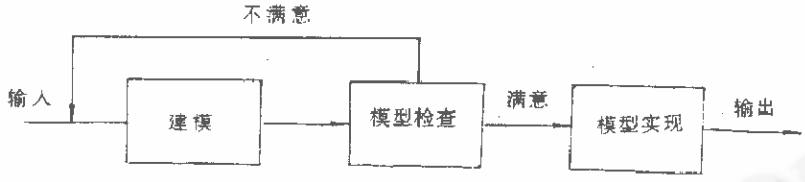

图 1

其中，建模和模型检查是整个设计过程的关键。[1] 中将建模过程归纳为三步，即基准步-、进程步和网络步。但在[1] 中，模型是以非形式化方法描述的。为了有效地进行模型检查，有必要研究形式化的模型描述。本文是[1] 的继续，讨论系统模型开发的形式化技术。

§2. 模型描述语言

概念模型是实际外界系统的特征抽象表示。由概念模型语言CML 描述的系统模型包括对象、关系、事件和进程等四部分，CML 从概念上定性地描述并发系统模型。以下仍以[1] 中的飞机订票系统(FRS) 为例，给出系统模型的CML 描述如下:

· 对象

```
Section Classes:
customer (paid, delivered, id);
order (name, flight, active, avail);
```

```
ticket (flight, allocated);
```

这部分说明了FRS 中有三类实体，即顾客customer，订单order 和机票ticket，类属性列在类名后的括号内。customer 有三个属性：paid 表示有否付款，delivered 表示有否收到机票，id 表示顾客名；类order 有四个属性：name 是订单的顾客名，flight 是飞机航班号，active 表示订单是否有效，avail 可取0, 1, 2 值，0 表示订单尚未处理，1 表示票已预订好，2 表示无票。同样，类ticket 有二个属性：flight 和allocated 分别表示航班号和订购情况。

· 关系

系统内各类属性之间常要求满足一定的限制条件，这样的静态限制条件在CML 中是用多类(multi-sorted) 一阶谓词逻辑来表示的，其中的量词约束变量明显指定其所属的类型。例如，FRS 中各类属性之间要求满足如下的限制条件：

```
Section Static Constraints:
(∀x: order )(∃y: customer) (x. name=y. id);
(∀ x: customer) (~(x. delivered ∧ ~x. paid));
```

此两个静态限制条件，前者表示任何一个订单都对应有一个登记该订单的顾客；后者表示对于任何一个顾客，不可能未付款就拿到了机票。

· 事件

在类的实体上允许执行一些元操作，称为事件。事件描述要给出参与者所属的类名和事件执行前、后的断言。在有些事件描述中，还允许出现局部变元，用作前后断言之间对存在的某一实元的引用。FRS 中的事件包括reserve(预订), cancel (取消), pay(付款), allocate (分配机票), void (通知无票), deliver (发送机票), skip (空操作) 等，它们的CML 描述如下：

```
Section Events:
  reserve        (c: customer, od: order)
    $ pre        (od. name=c. id)∧ ~od. active
    $ post       (od. name=c. id)∧ od. active ∧ (od. avail=0)
                 ∧ ~ c. paid ∧ ~ c. delivered
          ;
  cancel         (c: customer, od: order)
    $ some       (tck: ticket)
    $ pre        ( c.id=od. name) ∧ od. active ∧ ~ c. delivered
                 ∧ (( od. avail=2) ∨ (od. avail=1 ∧ od. flight=tck.flight ∧ tck.allocated))
    $ post       (c. id=od.name) ∧ ~ od. active ∧ ~ c.delivered
                 ∧ ((od. avail=2) ∨ (od. avail=1 ∧ od. flight=tck. flight ∧ ~ tck. allocated))
          ;
  pay            (c: customer, od: order)
    $ pre        ~ c.paid
    $ post       c. paid
          ;
  allocate       (od: order, tck: ticket)
```

```
    $ pre     od. active ∧ (od. avail=0)
    $ post    od. active
              ∧ ((od. avail=2) ∨
              (od. avail=1 ∧ od. flight=tck. flight ∧ tck. allocated))
        ;
  deliver     (c: customer, od: order, tck: ticket)
    $ pre     (c. id=od. name) ∧ od. active ∧ (od. avail=1)
              ∧ tck. allocated ∧ od. flight=tck. flight ∧ ~ c. delivered
    $ post    ~ od. active ∧ c. delivered
        ;
  void        (c: customer, od: order)
    $ pre     (c. id=od. name) ∧ od. active ∧ (od. avail=2)
              ∧ ~ c. delivered
    $ post    ~ od. active ∧ ~ c. delivered
        ;
  Skip ( )
    $ pre
    $ post
        ;
```

· 进程

每类实体的独立活动是一进程，表示为一串事件的序列。序列中允许出现一些控制算符，例如:

(i) 循环算符"*", 如a^* 表示a 循环执行零次或若干次。

(ii) 或算符"。", 如$(a,b)^\circ$ 表示或者执行a, 或者执行b。

(iii) 顺序算符"·", 如$a \cdot b$ 表示a,b 顺序执行。通常情况下，顺序算符可省略不写。

CML 中还引入了其它几种缩写算符，这里从略。

进程事件序列中的事件有主动、被动之分，它们的逻辑语义相同，且保持确定的时序关联。例如，reserve(c, od) 是主动事件，reserved (c, od) 是被动事件，当主动事件发生时，借助于进程间隐式通道的信息传递，相应的被动事件亦立即发生，任何一对对应的主、被动事件在时序上同时发生和完成。CML 还允许利用系统操作send和read 发送/读取信息，从而显式地确立实体并发活动的同步控制。

相应于FRS 的三种不同对象，其进程的CML 描述如下:

```
Section Processes:
  customer    (c):
    $ some    (od: order, tck: ticket)
    $ body    reserve (c, od)
              read (od) wait (od. vail ≠ 0)
              (cancel (c, od), voided (c, od),
                (pay (c, od) delivered (c, od, tck)))°
```

```
order       (od)
  $ some    (c: customer, tck: ticket)
  $ body    reserved (c, od) allocated (od, tck)
            send (c)
              (canceled (c, od), void (c, od),
                (paid (c, od) deliver (c, od, tck)))*.
      ;
ticket      (tck)
  $ some    (c: customer, od: order)
  $ body    allocated (od, tck) delivered (c, od, tck)
```

§3. 概念模型分析

对CML 所描述的系统模型的分析包括两个方面：一是静态分析，即模型描述一致性的检查；二是动态分析，即模型所应满足的时序特性的检查。

静态分析就是检查CML 描述所给出的若干静态限制本身的协调性，以及检查各类进程事件序列的合理性。为此，可利用扩充的Tableaux 方法[3,6]，从给出的谓词公式抽取出基原子集和一切可能的系统状态，再根据事件描述建立一个基模型。给出的一组静态限制公式是协调的，当且仅当存在这样的一个基模型。

基模型的系统状态是部分基原子集，状态之间的变换由事件确定。设基模型中有系统状态S_1 和S_2，若在S_1 处，事件e 的前断言为真，且在S_2 处，事件e 的后断言为真，则基模型中存在一条从S_1 到S_2 的子路径，如图2。

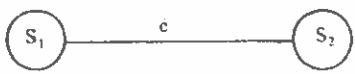

图 2

根据如此建立的基模型，容易检查每个进程事件序列的合理性。详细论证从略。

系统的时序特性与系统的初始状态有关。在进程事件序列的合理性确认之后，给定初始状态，可由基模型扩充，生成相关的系统模型，并在所生成的系统模型上检查系统的时序特性。

模型的初始状态包括给出每类实元的个数以及初始时它们的属性之间的关系，例如，设要生成每类各含两个实元的系统模型，可给出初始定义如下：

```
Customer: 2
order: 2
ticket: 2
for  i=0. .1 do
     c[i] (false, false, ID[i])
     od[i] (ID [i], NO [i], false, 0)
     tck [i] (NO [i], false)
```

系统模型的动态特性用分支时序逻辑公式描述[4], 例如

$$\forall i : (0..1) : od[i].active \rightarrow AU[true, \sim od[i].active]$$
$$\forall i : (0..1) : \sim EU[true, \sim C[i].paid \wedge C[i].delivered]$$

这里, $AU[f_1, f_2]$ 表示在模型的所有计算路径上, f_1 *until* f_2 为真; $EU[f_1, f_2]$ 表示存在某计算路径, f_1 *until* f_2 为真; 在一条计算路径上 f_1 *until* f_2 的解释与通常线性时序逻辑的解释是一样的, 因此, true until f 在路径上为真, 就表示在该路径上最终存在有一个状态, 使 f 为真。

这样, 对于如上两个分支时序逻辑公式, 前者表示在任何情况下, 订单都不会无限期的延迟而不予处理; 后者表示任何一个顾客, 不可能尚未付款而收到机票。在所生成的系统模型上, 这些时序公式的真假值可由labeling 算法来检查, 关于labeling 算法的细节可见[2]。

§4. 实现

CML/CMA 实现在USTATION 机器上, UNIX 操作系统支持, 系统的工作流程如图3, 图中各功能模块均已实现。

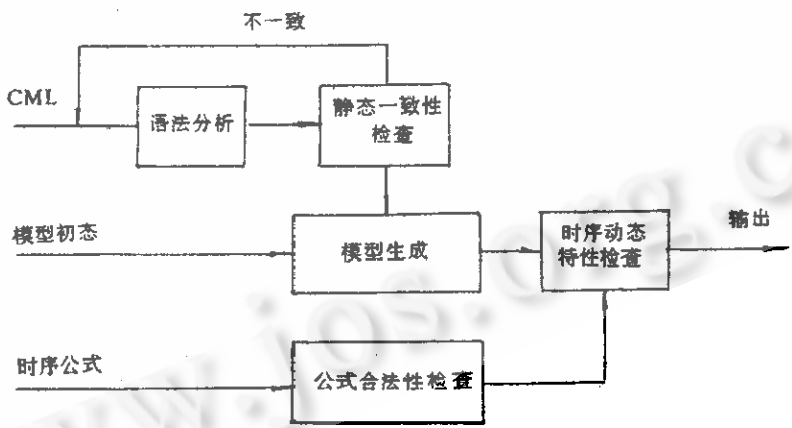

图 3

CML 描述是软件模型的一种形式化描述, 当然它不可能涉及到所要求设计的软件系统的所有细节方面, CML 只是定性地描述系统的一些主要特征方面, 并可由CMA 对之进行分析和处理。与通常所说的系统原型(prototype) 比较, 这样的模型显然是更加"粗线条" 的, 但它是形式的和严格的。系统开发的继续就是要将此CML 描述精化, 使粗线条细化, 根据具体的实现环境和策略, 建立一个实际的原型系统, 并保证实现的有效性。

§5 结论

本文将[1]中系统开发的时序模型方法形式化，设计了CML模型描述语言。CML模型描述概念简明，易写易读，并可用CMA进行静态一致性检查和时序动态特性检查，保证CML所描述的系统模型的合理性和可靠性。CML可以方便地描述系统的定性特征，现已成功地用CML设计了几个系统例子，如飞机订票系统、图书管理系统等。但是，另一方面，CML描述系统数量特征的能力很弱，这是有待进一步研究的课题。

与传统的自顶向下的软件工程设计方法比较，系统模型开发方法(无论是形式的，还是非形式的)都有着许多无可比拟的优点[1]，而形式化方法更是为软件工程设计自动化的长远目标所寻求的一种必然的技术途径。本文结果表明，概念模型的建立和验证可以有效地应用于软件工程系统设计的早期，提高系统原型设计的合理性和可靠性。

参考文献

[1] 冯玉琳，丁茂顺，“系统开发的时序模型方法”，《计算机研究与发展》，Vol. 26, No.1, 1989。

[2] 冯玉琳，赵旭东，郭端阳，“并发系统的模型和自动验证”，《计算机学报》，Vol.30 No.2, 1990。

[3] J. Bell, M. Machover, “A Course in Mathematical Logic”, North Holland, 1977.

[4] E. clarke, E. Emerson, A. Sista, “Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specification”, ACM Trans. on Prog. Languages and Systems, vol. 8, No. 2, 1986.

[5] M. Jackson, “System Development”, Prentice Hall, 1983.

[6] Gong Zhenhe, “On Conceptual Model Specification and Verification”, J. of Comp. Sci. and Technol., vol.2, No. 1, 1987.

[7] 桂自强，《CML/CMA系统用户指南》，中国科大计算机系技术报告，1989。