第 37 卷 第 2 期
Vol.37 No.2

计 算 机 工 程
Computer Engineering

2011 年 1 月
January 2011

• 软件技术与数据库 •　　文章编号：1000—3428(2011)02—0072—03　　文献标识码：A　　中图分类号：TP391

基于 TLA 的 UML 模型形式化验证

梁盟磊[a]，王小平[a]，薛小平[b]，李 刚[b]

(同济大学电子与信息工程学院 a. 计算机科学与技术系；b. 信息与通信工程系，上海 200092)

摘 要：统一建模语言(UML)不能直接对所建立模型的正确性进行形式化验证。为解决上述问题，从 UML 模型的静态结构和动态行为 2 个方面分别提出结合行为时序逻辑(TLA)的模型形式化方法，在此基础上提出将 UML 模型转化为 TLA+的形式化描述方法，并用 TLC 工具形式化检测 TLA+描述的正确性。通过实例分析证明了该方法的有效性。

关键词：形式化方法；形式化验证；统一建模语言；行为时序逻辑

Formal Verification of UML Models Based on TLA

LIANG Meng-lei[a], WANG Xiao-ping[a], XUE Xiao-ping[b], LI Gang[b]

(a. Department of Computer Science and Technology; b. Department of Information and Communication Engineering, College of Electronics and Information, Tongji University, Shanghai 200092, China)

【Abstract】UML can not carry out a formal check directly to models established. In order to solve the problem, based on the static structures and dynamic behaviors of UML models, this paper proposes two kinds of methods combing Temporal Logic of Actions(TLA) for model formalization. Formal description method for translating the UML model to TLA+ is given, and TLC toolkit is used to verify TLA+ specification. An example of UML model translation and verification is given to show the effectiveness of the method.

【Key words】formal method; formal verification; UML; Temporal Logic of Actions(TLA)

DOI: 10.3969/j.issn.1000-3428.2011.02.025

1 概述

随着计算机软件系统复杂性的提高和规模的扩大，开发难度和成本也相应提高，交付的软件产品出现错误和安全隐患的可能性越来越大，用人工的方法排除非常困难，而一般的测试方法很难发现系统所有的设计错误。针对上述问题，人们提出了形式化方法验证软件在设计、实现以及测试过程中的正确性和可靠性[1]。形式化方法是一种基于离散数学和形式逻辑的方法，借助其设计的模型及模型性质经过严格的数学描述、逻辑证明，较传统方法更完整、周密。但该方法需要有数学知识和技巧并经过形式建模训练的系统设计人员的参与，因此，未得到广泛应用，目前也缺乏成熟的形式化工具支持。另一方面，半形式化方法接近于自然语言，采用图形元素抽象建模，如统一建模语言(UML)，较单纯的形式化方法更易于使用。因此，本文在 UML 模型中引入了形式化方法——行为时序逻辑(Temporal Logic of Actions, TLA)，提出一种方便有效的软件模型形式化描述和验证方法。

本文通过 UML 对系统进行建模，然后对系统的关键部分和方面进行形式化描述。TLA 和与它相关的描述语言 TLA+提供了一种将一个系统描述为一系列动作的表示法。每个动作的执行都会引起全局状态的某种转换。在从 UML 模型转化为 TLA+的过程中，将类图中的对象转化为 TLA+中的不变量，将用例图中的对象转化为 TLA+中的动作。每当 UML 模型转化为 TLA+结束时，使用 TLA 工具模型检测器 TLC[2] 来检验转化后的模型是否存在错误。

2 UML

UML 是一种定义良好、易于表达、功能较强且普遍适用的建模语言，已成为软件建模语言事实上的标准[3]。由于拥有多种类型视图的特点，UML 可从多个侧面为系统提供一种整体性描述。

2.1 类图及对象图

类图是静态结构模型的图形化标示图，它是(静态)声明的模型元素集合。而对象图显示了在一个时间点上系统细节状态的一个快照。类图和对象图中用到的建模元素的标示法主要包括类的标示和类之间关联的标示。类的标示分上中下 3 栏给出类名、属性名与类型、操作名。关联是指类之间的联系，用实线标示。图 1 为一个银行用户的部分类图。

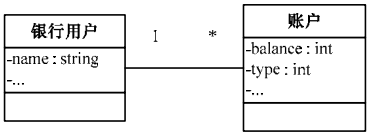

图 1 银行用户类图

2.2 状态图

状态图用来对一个反应式对象的生存周期状态进行建模[3]。一个状态图即为一个反应式对象上的状态机。UML 状态图的主要元素有：(1)状态名；(2)入口动作：在进入该状态的同时执行的动作；(3)出口动作；(4)动作：随着由事件触发的内部转换而执行；(5)状态之间的转换：name(params)[guards]/actions_list，其中，name 为转换的名字，params 为传递的参数，guards 为执行所需满足的条件，actions_list 为伴随此转换执行的动作列表。

基金项目：国家自然科学基金资助项目(60972036)

作者简介：梁盟磊(1985－)，男，硕士研究生，主研方向：软件形式化，软件工程；王小平，教授、博士；薛小平，副教授、博士；李 刚，硕士研究生

收稿日期：2010-04-15　**E-mail：**menglei.leung@gmail.com

3 TLA及TLC

TLA[2,4]用于规范性描述及分析并发系统。其使用一个简单的逻辑公式来明确地描述转换系统及其属性，它是从线性时态逻辑到行为逻辑的扩展。具体包括：

(1)状态

一个状态是指对变量的一次赋值,即一个状态s将值s(x)赋给变量x，形式s[[x]]被用来表示s(x)。

(2)态函数

态函数f是关于变量的表达式，s[[f]]表示通过状态s赋给f的值：s[[f]]==f(∀v:s[[v]]/v)，其中，f(∀v:s[[v]]/v)表示对于任意的变量v，均用s[[v]]代替将原v代入函数f计算得出的值。

(3)状态谓词

状态谓词是态函数的一种，它的值为布尔型。若s[[P]]=true，则状态s满足状态谓词P，否则不满足。

(4)动作

一个动作A是一个具有真假值的表达式，表示从旧状态s到新状态n的关系,用形如v的变量表示旧状态s中的变量，形如v'的变量表示新状态n中的变量。s与n的关系定义为：s[[A]]n==A(∀v:s[[v]]/v, t[[v]]/v')。

TLA中的公式由动作、布尔运算符和时序运算符“[]”(总是)组成。如果对于行为σ中所有状态中的变量u，P(u)都为真，则定义时序公式[]P(u)为真。对于任何态函数f，定义$[A]_f$==A∨(f'=f)，表示如果一个单步满足$[A]_f$当且仅当它满足A或者它保持f不变。一个常见的TLA格式定义为：Φ==$Init_Φ$∧[][A]$_f$，其中，$Init_Φ$表示系统的初始状态。

TLC是一个用来检测TLA+[2,4]规范性描述正确性的TLA工具。在一个规范性描述中检查出错误最有效的方式就是检测此描述是否满足它所应满足的属性[2]。TLC首先检测TLA+规范性描述的句法和语义的正确性，然后计算所有的有效状态，这些有效状态构成一个状态空间，最后在状态空间上检测时序属性的正确性。当TLA+描述不满足某一属性时，TLC将会产生一个错误轨迹，此错误轨迹由一系列完整的状态组成，每个状态都记录了所有声明的变量的值，依据此TLC便能有效地跟踪和定位错误。

4 UML模型到TLA的转化

使用TLA+描述UML模型的流程如图2所示。

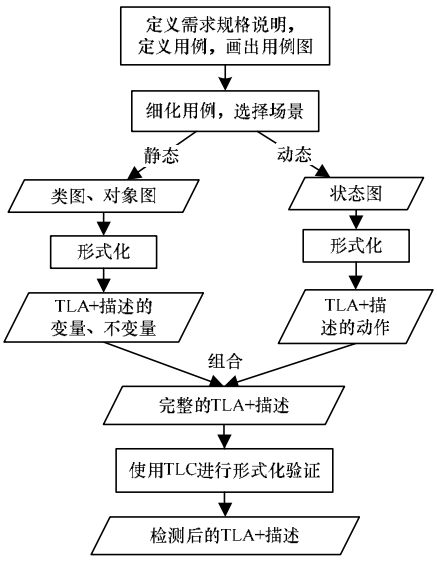

图2 UML模型转化为TLA的流程

此方法从需求分析开始，通过定义用例画出并细化用例图，分静态属性和动态属性分别构造类图、对象图及状态图。将类图、对象图形式化描述为TLA中的变量或者不变量，状态图形式化描述为TLA中的动作。将以上两部分组合成完整的TLA+描述，进一步经TLC的形式化验证后，得到TLA+描述。

4.1 类图及对象图的转化

类图和对象图中主要有3类元素需要进行形式转化。

(1)类和对象

用TLA+语言将类的类型描述成若干条记录的集合，记录的每个元素都表示该类的一个属性。例如，图1中的银行用户有一个名为“name”的属性，而账户有2个属性，分别为“type”和“balance”：

```
CustomerType==[name:STRING]
AccountType==[type:{savings,checking},balance:Int]
```

(2)类之间的关联

将类之间的关联用TLA+描述为变量(二元组)。关联中的约束用不变式描述。例如，假定图1中银行用户和账户之间的关联称为“has_account”：

```
has_account \in Rel(Customer,Account)
```

其中,Rel(Customer,Account)定义了银行用户与账户之间所有可能的关联，定义在模块“Relationships”中：

```
Rel(A,B)==SUBSET(A\X B)
First(R)=={t[1]:t \in R}
Second(R)=={t[2]:t \in R}
OneToOne(R)==/\\A x \in First(R),y1,y2 \in Second(R):<x,y1> \in R/\<x,y2> \in R=>y1=y2        /\\A y \in Second(R),x1,x2 \in First(R):<x1, y> \in R/\<x2,y> \in R=>x1=x2
/*…
```

(3)类型和数据等不变式

TLA中并没有类型的表示方法[2]，所以，模型中每个属性的类型和模型中出现的数据等不变式可以直接在TLA+中使用不变量描述。例如，图1中存在3种不变式：类型不变式，数据不变式以及连接不变式：

```
TypeInvariants==/\\A cstm \in Customer:cstm \in CustomerType/\\A acnt \in Account:acnt \in AccountType
DataInvariants==\A acnt \in Account:acnt.balance>=-credit_limit
AssociationInvariants==/\has_account \in Rel(Customer,Account)/\OneToMany(has_account)
```

4.2 状态图的转化

UML中使用状态图和时序图对模型的动态特征进行描述，在TLA中使用前置条件和后置条件描述UML状态图中的一个状态转换。具体步骤如下：

(1)将每个状态转换定义为一个TLA公式，转换的名字和参数保持不变。

(2)进行上述转换必须满足的约束条件用前置条件描述。

(3)上述转换的动作列表、状态的入口动作、内部转换动作和出口动作均由后置条件来描述。

(4)此次状态转换未被改变的变量使用UNCHANGED进行描述说明，若所有状态转换均被定义为TLA公式，转步骤(5)，否则，转步骤(1)。

(5)将所有被定义为TLA公式的状态转换合并组成整个模型的“Next”TLA公式。

假设图1中的银行模型具有3类操作：存款(Deposit)，取款(Withdraw)和转账(Transfer)，银行模型的状态图如图3所示。

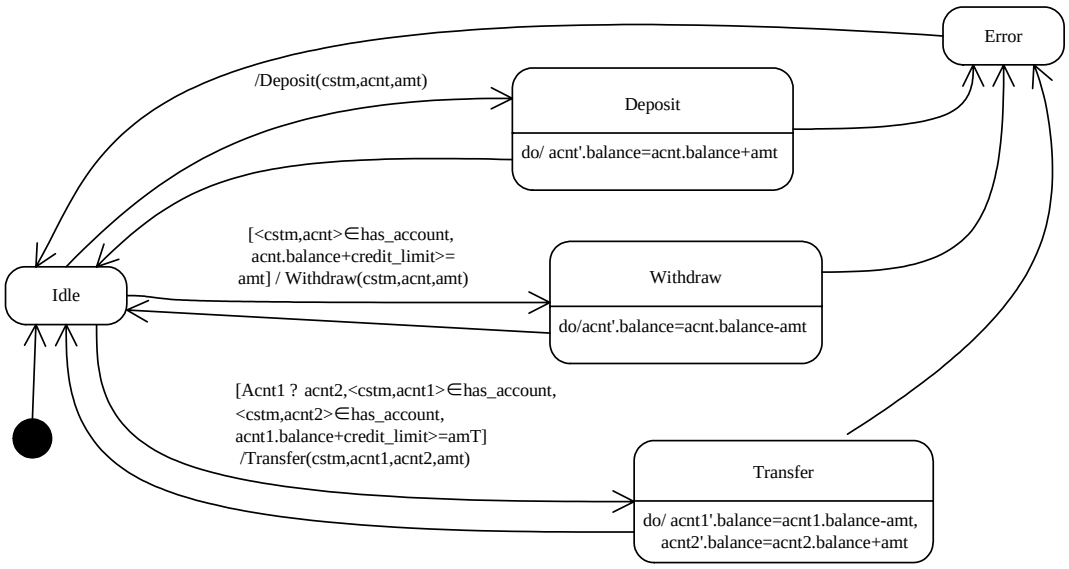

图 3 银行模型状态图

以“转账”操作为例，在 Bank 模块中定义一个称为“Transfer(cstm,acnt1,acnt2,amt)”的公式：

Transfer(cstm,acnt1,acnt2,amt)==/\acnt1#acnt2/\<cstm,acnt1>\in has_account

/\<cstm,acnt2>\in has_account/\acnt1.balance+credit_limit>=amt

/\acnt1’.balance=acnt1.balance–amt/\acnt2’.balance= acnt2.balance+amt/\UNCHANGED Customer

5 基于 TLC 的模型检测

TLC 模型检测器能检验如下形式的规范描述：

Init∧[][Next]_vars∧Temporal

其中，Init 为模型的初始谓词； Next 为执行下一个状态转换的动作；vars 为所有变量组成的多元组；Temporal 为标示存活条件的时序公式。

在上述银行系统中，TLC 检测的规范性描述 BankSpec 和属性 InvProperty 定义如下：

vars==<<Customer,Account,has_account>>

BankSpec==VarInit/\[][Next]_vars

InvProperty==[]Invariants

由于 TLC 是通过穷举状态空间中所有可到达的状态进行检测的，因此在对上述银行系统的检测中应该将 Deposit、Withdraw 及 Transfer 中的 amt \in Nat 改为 amt \in Data，其中，Data=={100,200}。检测中出现的模型错误主要是如上 Transfer 公式中的 acnt1’.balance 及 acnt2’.balance，因为 acnt1 和 acnt2 并非系统中变量，所以使用 acnt1’和 acnt2’表示进入下一个状态后的 acnt1 和 acnt2 是错误的，会导致 TLC 搜索状态的过程死锁。按 TLC 检测结果的提示信息，做如下修改即可改正这一错误。

Transfer(cstm,acnt1,acnt2,amt)==LET newac1==[acnt1 EXCEPT !.balance=@-amt]

newac2==[acnt2 EXCEPT !.balance=@+amt]

IN /\acnt1#acnt2/\<<cstm,acnt1>>\in has_account/\ <<cstm,acnt2>>\in has_account

/\acnt1.balance+credit_limit>=amt/\Account'=(Account\ {acnt1, acnt2})\cup{newac1,newac2}

/\has_account'=(has_account\{<<cstm,acnt1>>,<<cstm,acnt2>>})\cup {<<cstm,newac1>>,<<cstm,newac2>>}/\UNCHANGED Customer

6 结束语

本文提出了一种将 UML 模型转化成 TLA 的方法，并使用 TLA 模型检测工具 TLC 对形式化描述进行检测。虽然对简单模型而言，TLC 检测时间很短，但对复杂的大规模模型而言，由于 TLC 的工作机制是通过穷举搜索状态空间中所有可到达的状态来检测 TLA+描述的正确性，并且状态空间将随状态中变量个数的增长呈指数增长[5]，因此使用 TLC 分析复杂的大规模模型会导致常见的组合爆炸问题。为了有效解决这一问题，状态的个数应当限制在很小的范围内，主要的途径包括：

(1)通过类似于将 NFA 转换为 DFA 并简化转换后的 DFA 的方法来减少 UML 模型中状态图的状态个数。

(2)将对象模型划分成多个耦合度低的组件，对这些组件分别进行形式化描述，检测各个组件形式化描述的正确性。

这些也是今后将重点研究的内容。

参考文献

[1] 陈振庆. 基于 SHOIN(D)的 UML 类图形式化方法[J]. 计算机工程, 2009, 35(19): 43-45.

[2] Merz S. The Specification Language TLA+[M]//Bjorner D, Henson M C. Logics of Specification Languages: Part Ⅱ. Berlin, Germany: Springer, 2008: 401-451.

[3] Booch G, Rumbaugh J, Jacobson I. The Unified Modeling Language User Guide[M]. 2nd ed. Beijing, China: Machine Press, 2006.

[4] Freinkel L. An Approach to Combining UML and TLA+ in Software Speciation[D]. Reno: University of Nevada, 2003.

[5] Shrotri U, Bhaduri P, Venkatesh R. Model Checking Visual Specification of Requirements[C]//Proc. of International Conference on Software Engineering and Formal Methods. Brisbane, Australia: [s. n.], 2003.

编辑 张 帆