基于场景分析的系统形式化模型生成方法

王 曦 徐中伟
(同济大学电子与信息工程学院 上海 201804)

摘 要 采用形式化方法对系统的安全性进行分析与验证,是构造可靠安全软件系统的一个重要途径。当前的形式化安全分析方法,面临着系统的形式化建模难的问题。以铁路车站联锁系统中基本进路建立为例,提出基于场景分析的系统形式化模型生成方法。该方法首先采用OCL前/后置条件分析法对UML时序场景作一致性分析,然后将UML时序图中对象交互的行为序列转换成FSP进程代数模型,进而得到系统的形式化模型。该方法为系统的形式化建模提供了新思路,从安全质量方面改善了安全苛求软件的设计与开发,丰厚了基于模型的软件形式化开发方法。

关键词 安全苛求系统,安全性分析,形式化方法,形式化验证
中图法分类号 TP39 **文献标识码** A

Method for Generating Formal System Model Based on Scenarios Analysis

WANG Xi XU Zhong-wei
(School of Electronics&Information Engineering, Tongji University, Shanghai 201804, China)

Abstract An important operation for constructing reliable and safety software systems is system's safety analysis and verification by formal methods. Formal modeling for system is vital, which can affect the results of safety analysis and verification. We provided a method for generating formal system model based on scenarios analysis. The method adopted UML sequence diagrams to specify system's requirement firstly. In order to get consistent scenario-based requirements, we combined pre-condition and post-condition of object constraint language with domain knowledge to analyze conflicts in UML sequence diagrams. Besides, we proposed a model conversion algorithm for transforming interactions of objects in UML sequence diagram to finite stated process. Finally, we generated a finite states model for system's formal model, which is conformed to system's functional requirements. The correctness and feasibility of the proposed method were confirmed by generating formal model for railway station interlocking system. The new method provides a good way for system's formal modeling, and improves safety quality in demanding for safety software's designing and development.

Keywords Safety-critical system, Safety analysis, Formal methods, Formal verification

在航空航天、交通运输、核电能源和医疗卫生等安全苛求(Safety Critical)计算机系统领域[1],系统的安全性尤为重要,一旦这些系统失效,将造成人员伤亡、生命财产的重大损失或者环境的严重破坏。为了确保此类系统的安全性,已有一些国际标准组织制定了不同领域的安全标准与规范。根据欧洲电气化标准委员会的EN标准EN50128[2],在安全完善度等级[3]为SIL4(Safety Integrity Level 4)的安全系统中,对软件需求规格说明书和软件设计强列推荐使用形式化方法。功能安全国际标准IEC61508[4,5]指出,为了确保安全相关系统的充分安全性,其需求描述和系统设计需要使用形式化方法。形式化方法采用严格的数学符号和数学法则对目标软件系统的结构与行为进行有效的综合、分析和推理,为系统的需求规格描述、开发和验证提供了一个框架。为了进一步保障铁路运营安全,如何将形式化技术较好地应用在铁路安全领域成了学术界和产业界关注的热点话题。

目前,对铁路车站联锁系统的安全性保障主要通过仿真测试[6,7]和模拟验证[8,9]的方法对系统的功能行为进行验证和确认。文献[6]以计算机联锁软件的测试为例,研究了基于测试的安全性评价基准。文献[7]采用Dijkstra分布式终止探测算法对国际标准测试语言TTCN-3进行改进,对铁路控制系统软件进行测试。文献[8]采用Rhapsody工具软件对铁路联锁系统的UML模型进行功能模拟,根据模拟结果分析是否存在不安全的进路来判断系统功能行为的安全性。文献[9]通过基于进程控制的EVALPSN程序模拟器对铁路联锁系统的安全性进行模拟验证。但测试和模拟这两种方法只能证明系统有错,不能证明系统没有错,在验证过程中都需要依赖于测试向量的选取,然而要合理充分地选取测试向量,达到高覆盖率,是一个十分艰巨的课题。

采用形式化方法对系统的安全性进行分析与验证,是构造可靠安全软件系统的一个重要途径。文献[10]采用FTA

到稿日期:2011-09-22 返修日期:2011-11-23 本文受国家科技支撑计划重大项目(2009BAG11B00,2011BAG01B03),国家自然科学基金项目(60674004,61075002)资助。

王 曦(1974-),女,博士生,CCF会员,主要研究方向为模型检测、安全软件的安全性分析与评估,E-mail:wang_xi_happy@163.com;徐中伟(1964-),男,博士,教授,博士生导师,主要研究方向为基于通信的列车控制系统及软件、通信协议的安全性形式验证和测试评估。

结合 LTS 模型检测的方法对安全苛求系统的安全性进行分析与验证，提出了基于模型检测的软件安全性验证方法。文献[11]采用 RAISE 方法验证分布式铁路控制系统的功能实现和安全需求。文献[12]采用符号模型检测技术和 CTL 时序逻辑，利用验证工具 Verus 验证了 Ansaldo 联锁系统的安全逻辑。在以上形式化方法及当前基于模型的安全性分析方法[13-15]中，主要关注的一个方面是系统的形式化建模。建模的准确性将直接影响到安全性分析与验证的结果，因此，通常要求安全分析人员掌握娴熟的形式化技术，但应用该类方法的难度比较大。

鉴于此，本文以铁路车站联锁系统中基本进路的建立为例，提出基于场景分析的系统形式化模型生成方法。该方法采用 UML 时序图描述系统需求场景，通过 OCL(Object Constraint Language)中的前/后置条件对 UML 时序图中的消息进行分析，从而得到一致的需求场景；在识别 UML 时序图中对象交互的行为序列及其类别的基础上，通过系统形式化模型生成算法将 UML 时序图转换成 FSP(Finite State Pro-cess)，并得到系统的有穷状态机模型。该方法为安全苛求领域面临的形式化建模难问题的解决提供了有效途径，也为形式化验证与分析过程中系统的形式化建模提供了新思路，从安全质量方面改善了安全苛求软件的设计与开发，丰厚了基于模型的软件形式化开发方法。

1 背景知识

1.1 OCL 简介

对象约束语言 OCL[16]是一种简单的格式语言，它允许把附加的语义追加到 UML 模型中。OCL 可用来定义系统数据的范围、默认值、前置条件和后置条件，还可用来描述施加于模型元素或模型元素的属性、操作等上的约束条件等。在 OCL 中，约束的固化类型有不变量(invariant)、前置条件(pre-condition)和后置条件(post-condition) 3 种。不变量约束主要用于类，是一个在上下文(被约束的元素或对象)生存期必须始终为真值的；前置条件和后置条件约束主要用于操作，其中前置条件是：在一个操作被调用时必须为真的约束，后置条件就是在操作完成时必须为真的约束。

OCL 定义了很多基本类型，大部分 OCL 表达式都属于这些基本类型。表 1 描述了 OCL 的基本类型及其操作。

表 1 OCL 中常用的基本类型及其操作

类型	值	操作
Boolean	True, false	And, or, nor, implies, if-then-else
Integer	整数值	*, +, −, /, abs(), round(), floor()
Real	实数值	*, +, −, /, floor()
String	字符串	Toupper(), size(), concat()

1.2 FSP 简介

FSP[17]能够有效地描述并发系统和反应式系统的动态行为，是一种描述进程模型的代数标记，它通过层次、并发和广播通信机制来描述和解释系统的复杂行为，刻画了进程在其生命周期内接受激励后的状态变化过程。FSP 由进程操作、组合进程、公共操作和属性 4 部分组成。进程操作包括活动前缀(->)、选择(|)、守卫活动(when)和字母表的扩展(+)。组合进程操作包括进程标识(:)、重复符(forall)、并行组合(||)、优先级高(<<)和优先级低(>>)。公共操作包括重标识(/)、隐藏(\)、条件(if then else)和接口(@)。属性包括安全性和活性。假设 E 是 FSP 的进程表达式，P、Q 是进程标识符，BE 是布尔表达式，FSP 的主要语言定义如下，其中符号"//"后的说明为该行 FSP 语言的相应解释。

(1) P::=Q=E //进程定义：P 为 Q=E；

(2) |a->P //在完成动作行为 a 后，其后行为由 P 来确定；

(3) |when BE a->P //在条件 BE 为真的时候，动作 a 到 P 的迁移才会发生；

(4) |a|b //表示 a 与 b 之间的非确定性选择；

(5) |E[X<-rec(X=E)] //表示递归；

(6) |E+B //表示字母表扩展。

(7) |P/{new/old} //表示重标号，即将 P 中的行为标识 old 用 new 替换；

(8) |P\{actions} //表示隐藏 P 中的行为 actions；

(9) |E@I // 用 I 表示 E 中的接口行为；

(10) |P1 || P2 //表示子进程 P1 与 P2 的并行组合操作；

(11) |STOP //说明进程终止，或进程死锁；

(12) |ERROR //说明进程错误。

FSP 的形式语义通过标号迁移系统 LTS(LabeledTransitionSystems)来定义。LTS 的定义如下：

假设 S 是状态全集，且 $\pi \in S$ 是 S 中包含的一个出错状态；Labels 是标号全集，且 Act=Labels$\cup\{\tau\}$，Act 表示动作，τ 表示不能被外部环境观察到的内部行为。则一个有限 LTS M 的模型表示为 $M=(Q,A,T,Q_0)$，其中 $Q\subseteq S$ 是 M 的有穷状态集合，$Q_0\in Q$ 为初始状态，$A=ap\cup\{\tau\}$，且 $ap\subseteq$ Labels 是 M 中的有限字母表，$T\subseteq Q\times 2^A\times Q$ 为迁移关系。

2 基于场景分析的系统形式化模型生成方法

本节提出基于场景分析的系统形式化模型生成方法。该方法从基于 UML 时序图的需求描述出发，根据 OCL 前/后置条件对 UML 时序图中的消息进行分析。通过结合领域知识，分析需求场景中存在的矛盾，并对多个 UML 时序图中包含的相同或相似行为进行合并，在得到一致的需求场景的基础上，识别 UML 时序图中对象交互的行为序列并将其转换成相应的 FSP 进程代数模型。将其做组合运算后，得到系统的有穷状态机模型 FSM(Finite States Model)，在模型检测器 LTSA 中对 FSM 进行功能模拟，若 FSM 的功能行为满足系统的功能需求，则此 FSM 为系统的形式化模型；否则，结合需求场景，重复上述过程对 FSM 进行修改，直到 FSM 满足系统需求为止。基于场景分析的系统形式化模型生成方法主要由系统需求场景的 OCL 分析子算法和系统形式化模型生成子算法构成，以下将具体描述。

2.1 系统需求场景的 OCL 分析子算法

基于场景的需求描述通常采用 UML 时序图来描述，系统需求场景的 OCL 分析子算法根据 UML 时序图描述的需求场景，以及 OCL 前/后置条件分析和领域知识对 UML 时序图进行矛盾检测及矛盾消解，识别出 UML 时序图中对象包含的顺序行为序列、可选择性行为及并发行为，并对多个 UML 时序图中包含的相同或相似行为进行合并。具体的算法执行过程如下：

1. 根据需求场景，将系统设计过程中需要定义的全局变量组合成一个状态向量，状态向量的表现形式为：$v_i=\langle v_{i,1}, v_{i,2},\cdots,v_{i,n}\rangle$，其中 $v_{i,j}(1\leqslant j\leqslant n)$表示状态变量。

2. 根据需求场景，结合领域知识，写出 UML 时序图中主要消息需要满足的 OCL 前/后置条件。

3. 初始状态向量用 v_0 表示，根据 UML 时序图中消息 m_i 的 OCL 前置条件生成的状态向量值记为 m_i 的前置状态向量值，用 pre_v_i 表示；根据消息 m_i 的 OCL 后置条件生成的状态向量值记为 m_i 的后置状态向量值，用 post_v_i 表示。

4. 按如下规则生成 UML 时序图中所有消息的前/后置状态向量值：

规则 1　若 $v_{0,j}$ 未知，则 $v_{0,j}$ 为 null；

规则 2　若 pre_$v_{i,j}$ 的值为 x，则 pre_$v_{i,j}=x$；

规则 3　若 post_$v_{i,j}$ 的值为 y，则 post_$v_{i,j}=y$；

规则 4　若 pre_$v_{i,j}$ 未定义，则 pre_$v_{i,j}$ = $post\_v_{i-1,j}$；

规则 5　若 post_$v_{i,j}$ 未定义，则 post_$v_{i,j}$ = pre_$v_{i,j}$。

5. 按以下情形检测消息之间的矛盾：

情形 1　对单个 UML 时序图，若 m_{i-1} 与 m_i 为两个顺序发生的消息，且 post_$v_{i-1}\neq$pre_v_i，则 m_{i-1} 与 m_i 之间存在矛盾；

情形 2　对两个 UML 时序图 UML_i 与 UML_j，假设 $UML_{i.m_i}$ 为 UML_i 中的消息，$UML_{j.m_j}$ 为 UML_j 中的消息，且 $UML_{i.m_i}$ 与 $UML_{j.m_j}$ 为同一消息，若 $UML_{i.m_i}$ 与 $UML_{j.m_j}$ 的状态向量值不相等，则 UML_i 与 UML_j 之间存在矛盾。

6. 根据领域知识，对检测出的矛盾进行分析，若需求场景不正确，则修改需求场景；否则，修改存在矛盾的 OCL 前/后置状态向量值。

7. 消除 UML 时序图中的矛盾后，得到一致的 UML 时序图。

8. 对于 UML 时序图 UML_i 与 UML_k，若消息 $m_i\in UML_{i.m_k}\in UML_k$，且 $m_i\neq m_k$，m_i 与 m_k 的前置状态向量值相等，则 m_i 与 m_k 为并发或可选择性行为。将不同 UML 时序图中的相同或相似行为进行合并，生成新的 UML 时序图 UML_new，若合并时遇到并发或可选择性行为，则在 UML_new 中增加相应的交互框。

9. 若 $m_i\in$UML_new，$m_j\in$UML_new，且 m_i 的后置状态向量值与 m_j 前置状态向量值相等，则 m_i 与 m_j 为顺序行为，否则根据领域知识识别 UML_new 中的并发行为与可选择性行为。

2.2　系统形式化模型生成子算法

UML 时序图描述的系统功能行为主要为顺序行为序列、可选择性行为及并发行为。本节根据 UML 时序图中对象交互不同类型的行为序列，提出基于 UML 时序图的系统形式化模型生成子算法，以下简称 UML_FSM 算法。UML_FSM 算法先将 UML 时序图中的各个对象的交互行为转换成 FSP，再将其做组合运算，得到系统的有穷状态机模型 FSM；然后在模型检测器 LTSA 中对 FSM 进行功能模拟，若 FSM 的功能行为满足系统的功能需求，则此 FSM 为系统的形式化模型；否则，结合需求场景，对 FSM 进行修改，直到 FSM 满足系统的功能需求为止。UML_FSM 算法的具体描述如下，UML_FSM 算法中每一行前面的数字表示行号，后面的"//"表示对该程序行的解释与说明。

```
UML_FSM(uml_mode, FRS) //uml_mode 为 UML 时序图，FRS 为系统的功能需求；
1. {Int N=uml_mode.Object.size();// 对 UML 时序图中的对象个数计数为 N；
2.    for i=1 to N do
3.      {if (actk..actl∈Objecti) and(actk..actl in par or alt) //对象 Objecti 的交互行为序列中存在并发或选择行为 actk，actk+1，…，actl，par 表示并发，alt 表示选择；
4.       Then {actk.pre=Tempk；actl.post=Templ；} //在并发或选择行为的开始和结束处增加子进程标识 Tempk 与 Templ；
5.     If(Objecti.actl.Objecti.actN is sequence) // 若 Objecti 的交互行为序列 act1，act2，…，actN 都是顺序行为序列；
6.       Then Objecti=(act1－> act2－>…－> actN－> Objecti).；//写出 Objecti 相应于顺序行为序列的 FSP 进程代数模型；
7.       Else //否则，Objecti 的交互行为序列中包含并发或选择性行为；
8.         {if(actl.pre≠Tempk) //若 Objecti 的第一个交互行为 actl 不是并发或可选择性行为；
9.          { Then Objecti=(actl－> act2－>…－> actj－> Tempk；//以 Tempk 为子进程，写出 Objecti 的 FSP 进程代数模型；
10.          Else Objecti=Tempk；//否则，Objecti 与 Tempk 为同一 FSP 进程代数模型；
11.          If(actk..actl in alt) and(actk.pre==Tempk) //若 Tempk 后的行为 actk，actk+1，…，actl 为可选择性行为
12.          Then Tempk=( actk－> Templ //写出 Tempk 的 FSP 进程代数模型；
                        | actk+1－> Templ // Templ 为 Tempk 后的子进程；
                        | actk+2－> Templ
                          ……
                        | actl－> Templ),
13.        Else Tempk=( actk－> actk+1－>…－> Templ //否则，actk，actk+1，…，actl 为并发行为，写出各个并发行为在一条路径上交替出现的行为迹；
                        | actk+1－> actk －>…－> Templ
                          ……
                        | actm－> actn－>…－> Templ).
14.      Write FSP model for Templ according to 5－13}；//根据程序行 5－13，写出 Templ 的 FSP 进程代数模型；
15.    } //end else in 7；
16.  } //end for in 2；
17. ‖Sysmodel=( Object1 ‖ Object2 ‖ … ‖ ObjectN).//得到 UML 时序图中所有对象的 FSP 进程代数模型后，对其作组合运算，生成系统的进程代数模型 Sysmodel；
18. Run Sysmodel in LTSA and get FSM;//在模型检测器 LTSA 中运行 Sysmodel，得到系统的有穷状态机模型 FSM；
19. Animation for FSM;//在 LTSA 中执行 Animation 操作，对 FSM 进行功能模拟；
20. If ( FSM.function -| FRS) //若 FSM 的功能行为满足 FRS；
21.   Then Exit()；// 则退出程序；
22.    Else modify FSM until FSM.function -| FRS;} //否则，结合 FRS，对 FSM 进行修改，直到 FSM 的功能行为满足 FRS 为止；
```

3 实例研究

本文以铁路车站联锁系统中的进路建立[18]为例，验证基于场景分析的系统形式化模型生成方法的实际可行性。在铁路车站联锁系统中，从车站值班人员开始办理进路到防护该进路的信号机开放，可分为 6 个阶段：

(1)操作阶段：值班人员按规定操作办理进路，以确定进路的范围、方向、性质(列车)以及特征(基本进路等)。

(2)选路阶段：根据已确定的进路范围，选出一条相应的进路并确定与进路有关的道岔及其符合进路开通的位置。

(3)道岔转换阶段：检查已选出的道岔的实际位置是否符合进路要求，若不相符，应将其转换到实际的位置。

(4)一致性检查阶段：检查进路中的各个道岔位置是否符合进路要求，为锁闭道岔作准备。

(5)进路锁闭阶段：在道岔位置正确、进路空闲、未建立敌对进路的条件下，将道岔和敌对进路锁闭，为信号开放作准备。

(6)开放信号阶段：开放信号，指示列车或车列驶入进路，在信号开放期间，还需不断地检查进路空闲和道岔状态，一旦出现危及行车安全的因素或列车驶入进路，应立即关闭信号。在列车出清了进路中所有道岔区段后，各个道岔同时解锁。

采用 UML 时序图描述上述基本进路建立的场景，如图 1 和图 2 所示，其中各消息的具体含义见表 2。

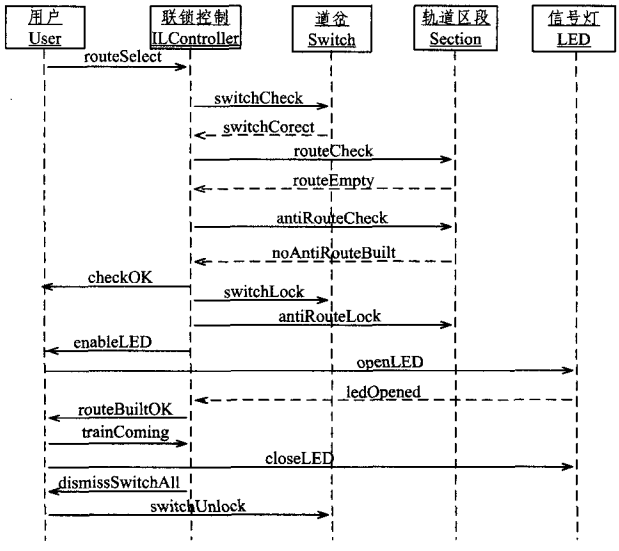

图 1 UML 时序图场景 1

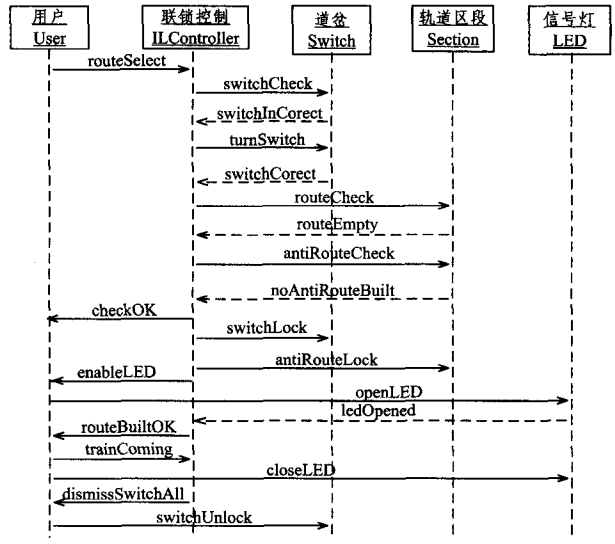

图 2 UML 时序图场景 2

表 2 UML 时序图中各对象之间行为交互的具体含义

消息名称	简称	含义	消息名称	简称	含义
routeSelect	rS	进路预选	openLED	oLED	开放信号灯
switchCheck	sCh	检查道岔的实际位置	ledOpened	ledO	信号灯已开放
switchInCorect	sIC	道岔位置不正确	routeBuiltOK	rBO	进路建立成功
turnSwitch	tS	转换道岔	trainComing	tC	列车驶入
switchCorect	sC	道岔位置正确	closeLED	cLED	关闭信号灯
routeCheck	rC	进路检查	dismissSwitch-All	dSA	所有道岔区段出清
routeEmpty	rE	进路空闲	switchUnlock	sUL	道岔解锁
antiRoute-Check	aRC	敌对进路检查	RouteOcupy	RO	进路占用
noAntiRoute-Built	nARB	未建立敌对进路	antiRouteBuilt	aRB	敌对进路建立
checkOK	cO	一致性检查正常	lEDOpen	lEDO	信号灯开放
switchLock	sL	锁闭道岔	RouteBuilt	RB	进路建立
antiRouteLock	aRL	敌对进路锁闭	SwitchOcupy	SO	道岔占用
enableLED	eLED	信号灯开放就绪			

表 3 UML 时序图场景 1 中消息的前/后置状态向量值

message	Pre-state vector value	Post- state vector value	message	Pre-state vector value	Post- state vector value
routeSelect	<null,null,null,null,f,f,f,f,f>	<null,null,null,null,f,f,f,f,f>	antiRouteLock	<t,f,f,f,t,f,f,f,f>	<t,f,f,t,t,f,f,f,f>
switchCheck	<null,null,null,null,f,f,f,f,f>	<null,null,null,null,f,f,f,f,f>	enableLED	<t,f,f,t,t,f,f,f,f>	<t,f,f,t,t,f,f,f,f>
switchCorect	<null,null,null,null,f,f,f,f,f>	<t,null,null,null,f,f,f,f,f>	openLED	<t,f,f,t,t,f,f,f,f>	<t,f,f,t,t,t,f,f,f>
routeCheck	<null,null,null,null,f,f,f,f,f>	<null,null,null,null,f,f,f,f,f>	ledOpened	<t,f,f,t,t,t,f,f,f>	<t,f,f,t,t,t,f,f,f>
routeEmpty	<null,null,null,null,f,f,f,f,f>	<null,f,null,null,f,f,f,f,f>	routeBuiltOK	<t,f,f,t,t,t,f,f,f>	<t,f,f,t,t,t,t,f,f>
antiRouteCheck	<null,null,null,null,f,f,f,f,f>	<null,null,null,null,f,f,f,f,f>	trainComing	<t,f,f,t,t,t,t,f,f>	<t,f,f,t,t,t,t,t,t>
noAntiRouteBuilt	<null,null,null,null,f,f,f,f,f>	<null,null,f,f,f,f,f,f,f>	closeLED	<t,f,f,t,t,t,t,t,t>	<t,f,f,t,t,f,t,t,t>
checkOK	<null,null,f,f,f,f,f,f,f>	<t,f,f,f,f,f,f,f,f>	dismissSwitchAll	<t,f,f,t,t,f,t,t,t>	<t,f,f,t,t,f,t,t,f>
switchLock	<t,f,f,f,f,f,f,f,f>	<t,f,f,f,t,f,f,f,f>	switchUnlock	<t,f,f,t,t,f,t,t,f>	<null,null,null,null,f,f,f,f,f>

矛盾 1　矛盾 2　矛盾 3

根据需求场景，选取 boolean 型的铁路车站联锁系统的状态向量：SwitchCorrect，RouteOcupy，antiRouteBuilt，antiRouteLock，SwitchLock，lEDOpen，RouteBuilt，TrainComing，SwitchOcupy。关于状态向量中各变量的具体含义参见表 2。根据系统需求场景的OCL分析子算法，生成 UML 时序图中消息的前/后置状态向量值，见表 3 和表 4，表 4 中的“……”表示以下消息的前/后置状态向量值同表 3。根据消息之间的矛盾检测情形，易知表 3 和表 4 中都存在“矛盾 1”和“矛盾 2”，表 3 和表 4 之间存在“矛盾 3”。

表 4　UML 时序图场景 2 中消息的前/后置状态向量值

message	Pre-state vector value	Post- state vector value	message	Pre-state vector value	Post- state vector value
routeSelect	<null,null,null,null,f,f,f,f,f>	<null,null,null,null,f,f,f,f,f>	switchCorect	< f,null,null,null,f,f,f,f,f >	< f,null,null,null,f,f,f,f,f>
switchCheck	<null,null,null,null,f,f,f,f,f>	<null,null,null,null,f,f,f,f,f>	routeCheck	< null,null,null,null,f,f,f,f,f>	<null,null,null,null,f,f,f,f,f>
switchInCorect	<null,null,null,null,f,f,f,f,f>	< f,null,null,null,f,f,f,f,f>	routeEmpty	< null,null,null,null,f,f,f,f,f>	< null,f,null,null,f,f,f,f,f>
turnSwitch	<f,null,null,null,f,f,f,f,f>	< f,null,null,null,f,f,f,f,f>	antiRouteCheck	< null,null,null,null,f,f,f,f,f>	<null,null,null,null,f,f,f,f,f>
……	……	……	……	……	……

矛盾 3　矛盾 1　矛盾 2

结合领域知识，分析“矛盾 1”、“矛盾 2”、“矛盾 3”得知，消息 switchCheck、routeCheck 与 antiRouteCheck 为并发行为，它们可在一条执行路径上交替出现，switchCorect 与 switchInCorect 为 switchCheck 发出后可出现的两种选择性行为，其余的消息为顺序行为。

根据 UML_FSM 算法，得到铁路车站联锁系统进路，建立各部件对象的 FSP 进程代数模型，描述如下：

User=(routeSelect->checkOK->enableLED->openLED->routeBuiltOK->trainComing->closeLED->dismissSwitchAll->switchUnlock->User).

ILController=(routeSelect->Temp),

Temp=(switchCheck->Temp11 |routeCheck->routeEmpty->Temp21 |antiRouteCheck->noAntiRouteBuilt->Temp31),

Temp11=(switchCorect->Temp12 |switchInCorect->Temp13),

Temp12=(routeCheck->routeEmpty->antiRouteCheck->noAntiRouteBuilt->Temp2 |antiRouteCheck->noAntiRouteBuilt->routeCheck->routeEmpty->Temp2),

Temp13=(turnSwitch->switchCorect->Temp12),

Temp2=(checkOK->Temp3),

Temp3=(switchLock ->antiRouteLock->Temp4),

Temp4=(enableLED->ledOpened->routeBuiltOK->trainComing->dismissSwitchAll->ILController),

Temp21=(switchCheck->Temp211 |antiRouteCheck->noAntiRouteBuilt->switchCheck->Temp212),

Temp211=(switchCorect->Temp2112 |switchInCorect->turnSwitch->switchCorect->Temp2112),

Temp2112=(antiRouteCheck->noAntiRouteBuilt->Temp2),

Temp212=(switchCorect->Temp2 |switchInCorect->turnSwitch->switchCorect->Temp2),

Temp31=(switchCheck ->Temp311 |routeCheck->routeEmpty->switchCheck->Temp212),

Temp311=(switchCorect->Temp3112 |switchInCorect->turnSwitch->switchCorect->Temp3112),

Temp3112=(routeCheck->routeEmpty->Temp2).

Switch=(switchCheck->SwitchTemp1),

SwitchTemp1=(switchCorect ->SwitchTemp2 |switchInCorect ->turnSwitch->switchCorect->SwitchTemp2),

SwitchTemp2=(switchLock->switchUnlock ->Switch).

Section=(routeCheck->routeEmpty->antiRouteCheck->noAntiRouteBuilt->SectionTemp |antiRouteCheck->noAntiRouteBuilt->routeCheck->routeEmpty->SectionTemp),

SectionTemp=(antiRouteLock->Section).

LED=(openLED->ledOpened->closeLED->LED).

在 LTSA 中执行组合运算后，得到系统的有穷状态机模型，如图 3 所示。图 3 中状态之间的迁移标识采用简称，其具体含义见表 2。通过功能模拟，系统的有穷状态机模型 SysModel 满足系统的功能需求，因此得到系统的形式化模型为 SysModel。

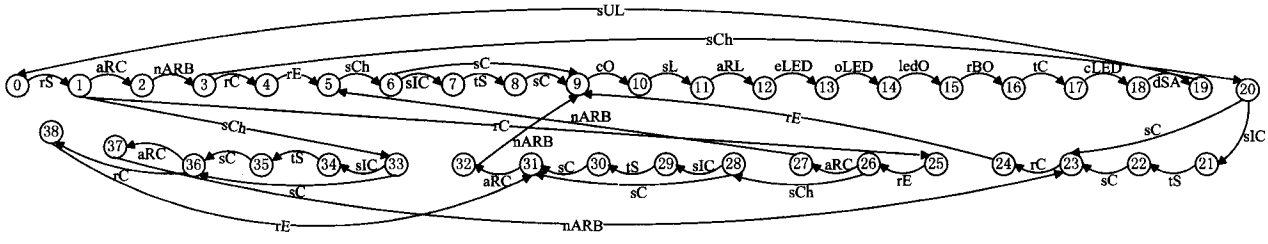

图 3　系统的有穷状态机模型

结束语　本文结合系统需求场景的 OCL 分析与模型转换算法，提出了基于场景分析的系统形式化模型生成方法。该方法在通过 UML 时序图描述系统需求场景的基础上，采用 OCL 前/后置条件分析法对 UML 时序图中的消息进行分析，检测出需求场景中存在的矛盾；然后结合领域知识，进行矛盾消解，并合并多个 UML 时序图中的相同或相似行为，得到一致的需求场景；在识别 UML 时序图中，在对象交互的行为序列及其类别的基础上，通过系统形式化模型生成算法将 UML 时序图中对象的交互行为转换成 FSP，最后得到满足系统功能需求的形式化模型。通过铁路车站联锁系统的形式化模型生成实例，验证了所提方法的正确性和实际可行性。基于场景分析的系统形式化模型生成方法为安全苛求领域面临的形式化建模难问题的解决提供了有效途径，也为形式化验

（下转第 163 页）

这是因为进行检测时，在无实例映射的情况下，并不知道要查询的事件在外存中是否存在，这时需要打开可能存储此事件的外存文件，在文件中查找此事件，若查找不到，就浪费了大量查询时间。有实例映射表时，在实例映射表对应位就可以直接查到在外存文件中是否有此事件的存在，避免了无意义的查找。所以 Instance_Map 可以提高检测的效率，节省检测的时间。

结束语 面向长过程的复杂事件检测是目前研究比较少的一个领域，当前该领域的技术还处于空白，主要是由于长过程复杂事件的事件海量性。本文针对面向长过程的复杂事件检测，提出了一系列的存储和置换策略，内存对象树结构能够有效地对存储于内存中的事件进行压缩，使得有限的内存空间存储更多的事件实例。置换策略将检测发生不频繁的事件调出到外存，极大地减少了 I/O 的开销，提高了检测的效率。事件实例映射表便利了对存储于外存的事件的检测，大大提高了检测的效率。本文基于真实数据集验证了这一系列存储和置换策略能够使长过程复杂事件的检测具有完整性，并具有较高的效率。

参 考 文 献

[1] Vu C, Beyah R, Li Y S. Composite Event Detection in Wireless Sensor Networks[C]// Proc. of IPCCC. 2007: 264-271

[2] Al M, Simson G, Beth R. RFID: applications, security, and privacy [M]. Addison-Wesley, 2006: 33-47

[3] Angell I, Kietmann J. RFID and the end of cash [C]// Communications of the ACM(CACM). December 2006: 91-96

[4] Wu E, Diao Y, Rizvi S. High-performance complex event processing over streams [C]// ACM SIGMOD Conference on Management of Data. 2006: 407-418

[5] Demers A, Gehrke J, Panda B, et al. Cayuga: a general purpose event monitoring system [C]// 3rd Biennial Conference on Innovative Data Systems Research(CIDR'07). 2007: 412-422

[6] Mei Y, Madden S. ZStream: a cost-based query processor for adaptively detecting composite events [C]// ACM SIGMOD Conference on Management of Data. 2009: 1-14

[7] Gruber R E, Krishnamurthy B B, Panagos E. The architecture of the READY event notification service [C]// Proc. of the 19th IEEE International Conference on Distributed Computing Systems Middleware Workshop. 1999: 108-113

[8] Chakravarthy S, Mishra D. Snoop: an expressive event specification language for active databases [J]. Data Knowledge Engineering, 1994, 14(1): 1-26

[9] Geppert A, Tombros D. Event-based distributed workflow execution with EVE [R]. ifi-96. 05. University Zurich, Computer Science Department, 1996: 1-16

[10] Cooper O, Edakkunni A, Franklin M J, et al. HiFi: a unified architecture for high fan-in systems [C]// International Conference on Very Large Data Bases(VLDB'04). 2004: 1357-1360

[11] Gatziu S S, Dittrich K R. SAMOS: an active object-oriented database system [J]. IEEE Bulletin on Data Engineering, 1992, 15 (1-4): 23-26

[12] Intel Berkeley Research lab. Intel Lab DataSite [DB/OL]. http://db.csail.mit.edu/labdata/labdata.html, 2004-6-2/2011-3-1

(上接第 140 页)

证与分析过程中系统的形式化建模提供了新思路，并从安全质量方面改善了安全苛求软件的设计与开发，丰厚了基于模型的软件形式化开发方法。

参 考 文 献

[1] Neil S. Safety Critical Computer Systems[M]. Addison Wesley, 1996, 8

[2] CENELEC. EN 50128, Railway Applications: Communications, signalling and processing systems— Software for railway control and protection systems[S]. CENELEC. 2001

[3] CENELEC. EN 50126, Railway Applications: The specification and demonstration of Reliability, Availability, Maintainability and Safety(RAMS)[S]. CENELEC. 1999

[4] IEC. IEC61508, Functional Safety of electrical/electronic/programmable electronic safety-related systems-part3: software requirements [S]. IEC. 2000

[5] IEC. IEC61508, Functional Safety of electrical/electronic/programmable electronic safety-related systems-part7: Overview of techniques and measures[S]. IEC. 2006

[6] 吴芳美. 计算机联锁软件基于测试的安全性评价基准研究[J]. 铁道学报, 2005, 27(3): 97-101

[7] Blom S, Ioustinova N, Pol J, et al. Simulated Time for Testing Railway Interlockings with TTCN-3[C]// Proceedings of the 5th International Workshop on Formal Approaches to. Testing of Software. LNCS 3997, 2006: 1-15

[8] Hon Y M, Kollmann M. Simulation and Verification of UML-based Railway Interlocking Designs[C]// AvoCS. 2006: 168-172

[9] Nakarnatsu K, Kiuchi Y, Chen W Y, et al. Intelligent Railway Interlocking Safety on Annotated Logic Program and Verification Based its Simulator[C]// Proceedings of the 2004 IEEE International Conference on Networkinp, Sensing & Control. Taipei, Taiwan, 2004

[10] 王曦, 徐中伟, 梅萌. 基于模型检测的软件安全性验证方法[J]. 武汉大学学报, 2010, 56(2): 156-160

[11] Haxthausen A E, Peleska J. Formal Development and Verification of a Distributed Railway Control System[J]. IEEE Transactions on Software Engineering, 2000, 26(8): 1546-1563

[12] Garmhausen V H, Campos S, Cimatti A. Verification of a safety-critical railway interlocking system with real-time constraints [J]. Elsevier Science of Computer Programming, 2000, (36): 53-64

[13] Yang Shuang-hua, Yang Li-li. Automatic safety analysis of control systems[J]. Journal of Loss Prevention in the Process Industries, 2005, 18(3): 178-185

[14] Bozzano M, Villa_orita A. The FSAP/NuSMV-SA Safety Analysis Platform [J]. Software Tools for Technology Transfer, 2007, 9(1): 5-24

[15] Koh K Y, Seong P H. SMV model-based safety analysis of software requirements[J]. Reliability Engineering & System Safety, 2009, 94(2): 320-331

[16] 郭宁. UML 及建模[M]. 北京: 清华大学出版社, 2007

[17] Magee J, Kramer J. Associated Concurrency: State Models and Java Programs[M]. Wiley, 1999

[18] 赵志熙. 计算机联锁系统技术[M]. 北京: 中国铁道出版社, 1999