软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(8):2851−2874 [doi: 10.13328/j.cnki.jos.006602]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于 SysML 的机载软件分层精化建模与验证方法*

肖思慧[1,2], 刘 琦[1,2], 黄滟鸿[1,2], 史建琦[1,2], 郭 欣[1,2]

[1](华东师范大学 软件工程学院, 上海 200062)

[2](国家可信嵌入式软件工程技术研究中心(华东师范大学), 上海 200062)

通信作者: 黄滟鸿, E-mail: yhhuang@sei.ecnu.edu.cn; 史建琦, E-mail: jqshi@sei.ecnu.edu.cn

摘 要: 机载软件被广泛应用于航空航天领域, 大幅提升了机载设备的性能. 随着机载软件规模逐渐增大、功能逐渐增多, 给软件的开发带来了难度. 如何保障机载软件的正确性和安全性, 也成为一个难题. 基于模型的开发可以有效提升开发效率, 而形式化方法能够有效保障软件的正确性. 为了降低开发难度, 同时保障机载软件的正确性、安全性, 提出一种基于SysML状态机图子集的机载软件分层精化建模与验证方法. 首先, 使用SysML状态机图对机载软件的动态行为进行建模, 根据提出的精化规则对初始模型进行手动逐层精化, 得到精化设计模型; 然后, 针对软件模型动态变化的特性, 将SysML状态机模型自动转换为时间自动机网络, 并从软件需求中手动提取形式化 TCTL 性质进行模型检验. 其次, 为了实现编码自动化, 将 SysML 模型自动转换至 Simulink, 利用 Simulink Coder 生成源代码. 最后, 以一个自动飞行控制软件为例进行了开发和验证, 实验结果表明了该方法的有效性.

关键词: 机载软件; 模型精化; 模型转换; 模型检验

中图法分类号: TP311

中文引用格式: 肖思慧, 刘琦, 黄滟鸿, 史建琦, 郭欣. 基于 SysML 的机载软件分层精化建模与验证方法. 软件学报, 2022, 33(8): 2851–2874. http://www.jos.org.cn/1000-9825/6602.htm

英文引用格式: Xiao SH, Liu Q, Huang YH, Shi JQ, Guo X. Hierarchical Refined Modeling and Verification Method of Airborne Software using SysML. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 2851−2874 (in Chinese). http://www.jos.org.cn/1000-9825/6602.htm

Hierarchical Refined Modeling and Verification Method of Airborne Software Using SysML

XIAO Si-Hui[1,2], LIU Qi[1,2], HUANG Yan-Hong[1,2], SHI Jian-Qi[1,2], GUO Xin[1,2]

[1](Software Engineering Institute, East China Normal University, Shanghai 200062, China)

[2](National Trusted Embedded Software Engineering Technology Research Center (East China Normal University), Shanghai 200062, China)

Abstract: Airborne software is widely used in aerospace, which dramatically improves the performance of airborne equipment. Nevertheless, with airborne software's increasing scale and function, it is challenging to develop airborne software. How to ensure the correctness and safety of airborne software has become a difficult problem to be solved. Model-based development can effectively improve development efficiency, and formal methods can effectively guarantee the correctness of software. To reduce the difficulty of development and ensure airborne software's correctness and safety, this study proposes a hierarchical refinement modeling and verification method of airborne software using the SysML state machine diagram subset. Firstly, the SysML state machine diagram is used to model the dynamic behavior of airborne software. According to the proposed refinement rules, the initial model is refined to obtain the refined design model step by step manually. Then, according to the dynamic characteristics of the software model, the SysML state machine model is automatically converted to a network of timed automata, and the formal TCTL properties are manually extracted from

* 基金项目: 国家重点研发计划(2019YFB2102602)

本文由“形式化方法与应用”专题特约编辑陈立前副教授、孙猛教授推荐.

收稿时间: 2021-09-05; 修改时间: 2021-10-14, 2022-02-07, 2022-03-01; 采用时间: 2022-03-19

the software requirements for model checking. Secondly, to realize coding automation, the SysML model is automatically converted to Simulink, and Simulink Coder generates the source code. Finally, an automatic flight control software is developed and verified based on the proposed method, and the experimental results show the effectiveness of the method.

Key words: airborne software; model refinement; model transformation; model checking

随着计算机系统在航空航天领域的广泛应用, 机载软件已经成为飞行控制、通信导航、火力控制以及维修保障的核心[1]. 但机载软件的安全性得不到足够的保障, 一旦软件失效, 很可能造成巨大的财产损失和人员伤亡. 为确保机载软件的正确性、安全性, 机载软件必须经过安全认证、符合适航标准才能投入使用. 目前, 航空领域广泛采用的是航空无线电技术委员会(radio technical commission for aeronautics, RTCA)提出的适航认证标准体系[1]. 适航标准是以目标为导向的, 规定了机载软件开发以及验证过程中所必须满足的一些目标, 但并未提供详细的开发和验证方法说明[1].本文基于 DO-331[2]和 DO-333[3]标准对机载软件进行开发和验证, 可为面向适航标准的相关工作提供参考.

在民用航空领域, 研究人员常基于模型进行机载软件的开发和验证. 模型的使用, 便于对系统工程进行管理, 还支持模型仿真、模型验证、自动化代码生成等手段, 可以大幅提高开发效率. 软件开发人员通常采用半形式化的建模语言对机载软件进行建模, 如统一建模语言(unified modeling language, UML)、系统建模语言(system modeling language, SysML)等. 其中, SysML[4]是 UML[5]的完善和扩展, 支持对软件的需求、动态行为以及静态结构进行建模. 美国宇航局的喷气推进实验室曾基于 SysML 模型来设计研发飞行系统[6]. 文献[7]对民用飞机的高升力系统进行基于模型的设计, 利用模型的关联性和追溯关系来保证设计过程的完整性.

模型检验是一种常用的形式化方法, 可以自动地对系统的功能属性定量地可重复性验证[1]. 目前, 基于模型的形式化验证方法已经存在一系列工具, 包括 NuSMV[8], Spinp[9], UPPAAL[10]等. 利用模型转换将 SysML 转换至形式化模型, 可以重用这些工具已有的验证和分析能力. 其中, UPPAAL 可以将实时系统建模为时间自动机(timed automata, TA), 可以对目标系统的动态行为进行仿真, 验证模型是否满足时间计算树逻辑(timed computation tree logic, TCTL)[11]语言描述的形式化性质.

代码自动生成可以提高软件开发的效率, 减轻软件开发的工作量. 为实现软件自动编码, 可以利用现有的支持模型生成代码的工具, 如 Mathworks 公司开发的可视化仿真工具 Simulink、Esterel 公司开发的 SCADE 等. 如文献[12]将 UML 与 Simulink 结合起来, 自动生成飞控软件的产品级代码.

为确保机载软件安全性和正确性, 一方面需要对软件运行模式进行分析, 获取软件的功能性需求, 同时对软件的故障模式进行分析, 从中获取软件的安全性需求, 以支持软件开发和验证; 另一方面, 从软件开发和验证过程出发, 根据适航标准相应的目标, 判断软件开发和验证过程中是否符合适航审定目标. 基于上述背景, 本文提出了一种基于 SysML 状态机图子集的机载软件分层精化建模与验证方法(如图 1 所示).

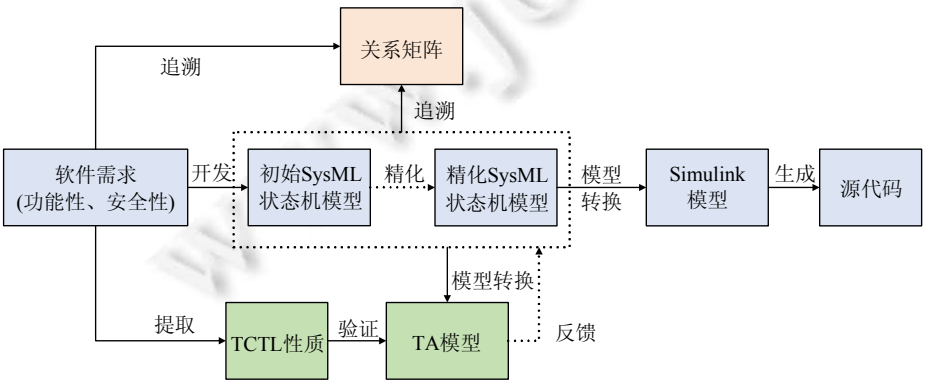


图 1 基于 SysML 的分层精化建模和验证方法框架

首先, 利用 SysML 需求图来描述和记录机载软件需求, 便于后续管理. 其次, 基于 SysML 状态机图(state

machine diagram, SMD)子集建立初始设计模型(高级需求模型), 然后手动应用精化规则得到详细精化模型(低级需求模型). 同时, 建立关系矩阵记录模型和需求间的追溯关系. 检测出内在缺陷的时间越晚, 修复该缺陷的代价也越高[13]. 因此在每层模型建立后, 通过自动模型转换得到形式化的时间自动机模型, 从需求中手动提取 TCTL 形式化性质进行模型检验, 对没有通过验证的模型, 修改其源 SysML 模型并再次进行模型检验, 直到所有需求约束通过检验. 最后, 将精化模型进行自动模型转换得到 Simulink 模型, 利用 Simulink Coder 自动生成源代码.

总结该方法主要优点有: (1) 可以对复杂机载软件进行分析和开发, 保障其安全性和正确性; (2) 提出了 SysML 状态机模型的精化规则, 从而进行分层建模; (3) 模型转换过程和编码过程高度自动化.

本文第 1 节主要介绍相关基础知识. 第 2 节介绍基于 SysML 状态机图子集的精化规则和精化建模实例. 第 3 节详细介绍模型转换的算法、映射规则和实例, 并分析模型转换的正确性. 第 4 节给出模型转换的工具实现. 第 5 节通过实例验证文中提出方法的有效性. 第 6 节分析国内外相关工作并与本文方法进行比较. 第 7 节总结本文工作并提出未来的研究方向.

1 基础知识介绍

1.1 适航标准DO-331和DO-333

机载软件作为安全关键软件, 必须通过适航审定才能投入使用[1]. RTCA 发布了适航标准 DO-331 和 DO-333, 为机载软件的安全性保障提供了审查目标. 其中, DO-331 提供了一些行业实践中的模型使用样例. 本文参考 DO-331 中模型使用样例 4, 将机载软件的生命周期划分为需求、设计模型和源代码这 3 个阶段. 其中, 初始设计模型描述较高级别需求, 精化设计模型描述较低级别需求, 二者之间没有明确界限. DO-333 介绍了 3 类形式化方法: 演绎法、模型检验、抽象解释. 模型检验与演绎法相比, 自动化程度更高且不需要验证人员掌握深厚的逻辑知识; 与抽象解释相比, 可应用于需求和设计过程的软件制品[14]. 因此, 本方法中使用模型检验, 对机载软件制品进行形式化验证, 验证其是否满足需求约束. 同时, 审查软件制品是否满足 DO-331 和 DO-333 规定的目标.

1.2 SysML基础知识与子集选取

系统建模语言 SysML 是对象管理组织(object management group, OMG)提出的标准建模语言. 在实际应用中, 机载软件建模不会涉及 SysML 的全部元素. 基于机载软件动态行为建模研究需要, 本工作依据最小化原则以及可描述性原则, 选取 SysML 状态机图子集进行机载软件建模.

(1) 最小化原则

最小化原则指选取的 SysML 子集是满足机载软件建模的最小子集. SysML 状态机图定义了 3 种状态内部行为 *entry*, *do*, *exit*, 它们分别可以用进入状态、状态向自身状态以及迁出状态的迁移”影响”代替. 因此, 基于最小化原则, 本工作选取的状态机子集不包括 *entry*, *do*, *exit*.

(2) 可描述性原则

可描述性指选取的SysML子集足以用于机载软件建模. 本工作主要考虑对机载软件动态行为的安全性和正确性进行分析, 其中: 安全性可以通过状态机能否正常运行, 以及故障条件判断、故障状态来体现; 正确性是从状态机对输入的处理是否符合需求来体现, 主要用到的是状态、迁移以及输入的数据变量.

依据上述子集选取原则, 最终我们选取了状态、迁移、变量、复合状态进行建模, 并给出 SysML 状态机子集的形式化定义.

定义 1. 状态机可以定义为一个六元组 $STM:=\langle S,s_0,s_f,V,T,CS\rangle$.

- S 为状态的有限集合, 状态主要分为初始状态(initial state)、简单状态(simple state)、最终状态(final state)以及复合状态(composite state);
- $s_0\in S$, 表示初始状态;

- $s_f \in S$, 表示最终状态;
- V 是变量的有限集合, 其数据类型有整型、浮点型、布尔型和实数型等;
- $T \subseteq S \times \{Tri, G, Eff\} \times S$, 表示状态迁移有限集合, 其中, Tri 表示事件触发器, G 表示守卫, Eff 表示影响. 源状态向目标状态的迁移可写作 $t(s_s, s_t)$, $t=(s_s, tri, g, eff, s_t)$;
- $CS \subseteq S$, $CS=S \times T \times S$, 表示复合状态, 内含嵌套子状态以及子状态之间的迁移. 复合状态内部结构与状态机总体类似, 复合状态活动时, 首先进入子初始状态; 复合状态活动时, 有且仅有一个子状态活动.

SysML 状态机初始处于初始状态 s_0, 后根据迁移上的 tri 和 g 进行状态迁移, 同时执行 eff, 进入最终状态 s_f 时结束运行. 迁移踪迹可以写作 $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \ldots \xrightarrow{t_n} s_f$.

1.3 UPPAAL基础知识

UPPAAL 是一个用于实时系统建模、仿真和验证的集成环境. UPPAAL 编辑器将系统行为描述为时间自动机网络(network of timed automata, NTA), 模拟器可以对系统行为进行动态执行, 验证器探索系统状态空间检查系统是否满足形式化 TCTL 公式.

定义 2. 时间自动机可以被定义为一个六元组 $TA:=\langle L, l_0, V, Act, I, E\rangle$, 其中,

- L 是时间自动机中位置的有限集合;
- $l_0 \in L$ 是时间自动机的初始位置;
- V 是变量的有限集合, 包括时钟变量和数据变量;
- Act 是动作的有限集合, 包括输入(记为?)和输出(记为!)的同步信号;
- $I: L \to B(V)$是位置集 L 到变量集 V 的映射, 称为位置不变式, 用于约束是否要离开当前位置. $B(V)$是由形如 $x \sim expr$ 的公式组成的集合, 其中, $x \in V$, $\sim \in \{<, \leqslant, =, \geqslant, >\}$, $expr$ 表示任意实数;
- $E \subseteq L \times G \times Act \times U \times L$ 表示边的有限集合, 其中, $G \subseteq B(V)$表示迁移约束条件集合, U 是对时钟变量或数据变量的更新操作. 边可以写作 $e(l_s, l_t)$或 $l_s \xrightarrow{(g,\alpha,r)} l_t$, 其中, $g \in G$, $\alpha \in Act$, $r \in U$.

时间自动机的初始状态位于初始位置 l_0, 当位置不变式 $I(l_0)$和边上的约束 g 时, 可以迁移至下一个位置并执行边上的动作α. 时间自动机迁移踪迹可以写作 $l_0 \xrightarrow{e_0} l_1 \xrightarrow{e_1} l_2 \ldots \xrightarrow{e_{n-1}} l_n \ldots$

定义 3. 时间自动机网络可以被定义为一个二元组 $NTA:=\langle \overline{TA}, Decl\rangle$, 其中,

- $\overline{TA} = \langle TA_1, TA_2, \ldots, TA_n\rangle$ 是时间自动机的有限集合;
- $Decl$ 是全局声明有限集合, 包括数据变量、时钟变量和同步信道的全局声明.

时间自动机网络是由一组相互平行且相互关联的时间自动机组成, 时间自动机之间通过全局数据变量、全局时钟变量和同步信道进行交互.

定义 4. TCTL 的巴科斯范式(Backus-Naur from, BNF)格式如下:

$$\varphi ::= p \mid \text{not } \varphi \mid \varphi_1 \text{ and } \varphi_2 \mid \varphi_1 \text{ or } \varphi_2 \mid \varphi_1 \text{ imply } \varphi_2 \mid A[]\varphi \mid A<>\varphi \mid E[]\varphi \mid E<>\varphi.$$

UPPAAL 所使用的 TCTL 语言包含两部分: 一部分是状态公式, 另一部分是路径公式. 状态公式用于评估单个状态, 同时, 路径公式用来量化模型的踪迹. 状态公式中, p 表示原子命题, φ是由 p 组成的表达式. 路径公式中, A(always)表示所有路径, E(eventually)表示存在某个路径, []表示该路径上的所有状态, <>表示该路径上的某个状态.

UPPAAL 使用的 TCTL 性质可分为可达性、活性和安全性, 具体含义如下.

- 可达性: 可达性是最简单的性质, 在检查模型基本功能行为时, 可以使用可达性. 如使用公式 $E<>$light 来检查灯模型是否可以“亮”(light);
- 活性: 活性意为“最终会发生一些事情”, 用于检查模型较高级的功能. 如通信协议模型中的消息最终会被接收(receive message), 可使用 $A<>$receive message 表示. 另一种常用的形式是φ_1 imply φ_2, 用于表达φ_1 满足时, φ_2 也会被满足;
- 安全性: 安全性通常有两种表现形式, $A[]\varphi$和 $E[]\varphi$, 用于表示“坏的事情永远不会发生”. 例如, $A[]$ not

deadlock 表示所有路径可达的状态无死锁, *E*[] not *deadlock* 则表示存在某路径所有状态无死锁.

TCTL 可达性和活性公式可检验模型功能行为以确保模型正确性, TCTL 安全性公式可验证模型安全性.

1.4 Simulink/Stateflow基础知识

Simulink 是一款功能强大的仿真工具, 支持对动态系统进行仿真和分析. 更重要的是, Simulink Coder 组件支持基于模型的自动代码生成. Stateflow 是 Simulink 的一个组件, 它由图块、状态、迁移等元素组成.

定义 5. Stateflow 可以被定义为一个五元组 $SF:=\langle S,Var,T,t_d,Ch\rangle$.

- S 为状态的有限集合;
- Var 是变量的有限集合, 其数据类型有 int16, int32, bool 等;
- $T\subseteq S\times\{Ev,Con,Act\}\times S$, 表示状态迁移有限集合. 其中, Ev 表示事件, Con 表示约束条件, Act 表示动作. 迁移可写作 $t=(s_s,e,c,\alpha,s_t)$, 其中, s_s 表示源状态, s_t 表示目标状态, $e\in Ev$, $c\in Con$, $\alpha\in Act$;
- $t_d\in T$, 是 Stateflow 中的默认迁移, 指向初始状态;
- $Ch=S\times T\times S$ 表示图块. 图块和状态机类似, 包含迁移、默认迁移和状态, 图块最多只能有一个状态活动.

2 基于 SysML 状态机图子集的分层精化建模

2.1 SysML状态机精化规则理论

本文提出了基于 SysML 状态机的精化开发方法进行机载软件建模, 以此来分解复杂软件. 本文提出的 SysML 状态机精化规则分为 3 类: 简单状态的简单精化、简单状态的分解精化以及迁移精化. 为了方便描述, 使用符号$\rightsquigarrow$表示精化关系, $name(s)$表示状态 s 的名称, 符号 S'表示精化后的状态集合, T'表示精化后的迁移集合, t'表示精化后的迁移, tri', g', eff'分别表示精化后的触发器、守卫和影响, CS'表示精化后的复合状态.

(1) 简单状态的简单精化

规则 1. $$ref(state\_add)=\frac{S'=S\cup\{s_0,s_1,...,s_k\},\ \forall n\neq m,\ s_n,s_m\in S',\ name(s_n)\neq name(s_m)}{S\rightsquigarrow S',\ s\rightsquigarrow\{s_0,s_1,...,s_k\}}.$$

规则 1 表示在原状态集合基础上新增多个状态 $s_0,s_1,\ldots,s_k$, 原状态集合 S 精化为 S'. 为了避免命名冲突, 需确保精化后的状态集合中状态命名不会产生重复.

(2) 简单状态的分解精化

规则 2. $$ref(state\_composite)=\frac{S'=S/\{s_i\}\cup CS',\ \forall s_i,s_j\in S',\ name(s_i)\neq name(s_j)}{S\rightsquigarrow S',\ s_i\rightsquigarrow CS'}.$$

规则 2 表示一个简单状态 s_i 精化为一个复合状态 CS', $CS'=\{S'\times T'\times S'\}$. 通过精化和分割简单状态为多个嵌套子状态, 并根据需求使用迁移的精化规则来添加状态之间的迁移. 精化后的复合状态任意两个状态的名称均不相同. 精化后迁移集合 $T'=\{t|t=(s_s,tri,g,eff,s_t),s_s\in S',s_t\in S'\}$.

(3) 迁移的精化

对迁移的精化规则可分为新增迁移和精化迁移信息两种.

规则 3. $$ref(transition\_add)=\frac{T'=T\cup\{t'\},\ t'=(s_s,tri',g',eff',s_t),\ s_s,s_t\in S}{T\rightsquigarrow T'}.$$

规则 3 为新增迁移, 表示在原有迁移集合 T 上新增迁移 $t'=(s_s,tri',g',eff',s_t)$, 其中, tri', g', eff'可以为空, 即:

$$t'=(s_s,\varnothing,\varnothing,\varnothing,s_t).$$

精化迁移可选信息如规则 4~规则 6 所示, 在状态机中原有迁移的基础上, 分别对触发器、守卫或影响进行更为详细的描述. 如迁移精化规则 4 是对迁移上触发器的精化.

规则 **4**.

$$ref(transition\_trigger)=\frac{T'=T/\{t\}\cup\{t'\},\ Tri'=Tri/\{tri\}\cup\{tri'\},\ t=(s_s,tri,g,eff,s_t),\ t'=(s_s,tri',g,eff,s_t)}{T\rightsquigarrow T',\ t\rightsquigarrow t'}.$$

规则 **5**. $ref(transition\_guard)=\dfrac{T'=T/\{t\}\cup\{t'\},\ G'=G/\{g\}\cup\{g'\},\ t=(s_s,tri,g,eff,s_t),\ t'=(s_s,tri,g',eff,s_t)}{T\rightsquigarrow T',\ t\rightsquigarrow t'}$.

规则 **6**.

$$ref(transition\_effect)=\frac{T'=T/\{t\}\cup\{t'\},\ Eff'=Eff/\{eff\}\cup\{eff'\},\ t=(s_s,tri,g,eff,s_t),\ t'=(s_s,tri,g,eff',s_t)}{T\rightsquigarrow T',\ t\rightsquigarrow t'}.$$

2.2 SysML状态机精化实例

本节以一个简单的音乐播放器模型为例, 阐述精化规则的应用.

- 首先建立如图 2(1)所示的初始 SysML 模型, 包含关闭(OFF)和开启(ON)状态;
- 在模型(1)的基础上应用精化规则 1, 添加新的状态异常(FAULT), 用于描述播放器发生故障的状态. 然后应用精化规则 3, 添加了迁移 ON 到 FAULT 和 FAULT 到 OFF, 精化后的模型如图 2(2)所示;
- 在模型(2)的基础上应用精化规则 5, 在 OFF 到 ON 的迁移上添加了新的守卫并应用精化规则 4, 在 ON 到 FAULT 的迁移上添加触发器 *ERROR*, 得到如图 2(3)所示模型;
- 在模型(3)的基础上应用精化规则 2, 将简单状态 ON 精化为一个复合状态, 并应用精化规则 1 得到 ON 嵌套状态集 *S*={*IDLE*,*PLAYING*,*PAUSED*}, 表示就绪、播放和暂停. 在复合状态 ON 中使用精化规则 3, 添加嵌套状态之间的迁移. 最后得到如图 2(4)所示的最终模型.

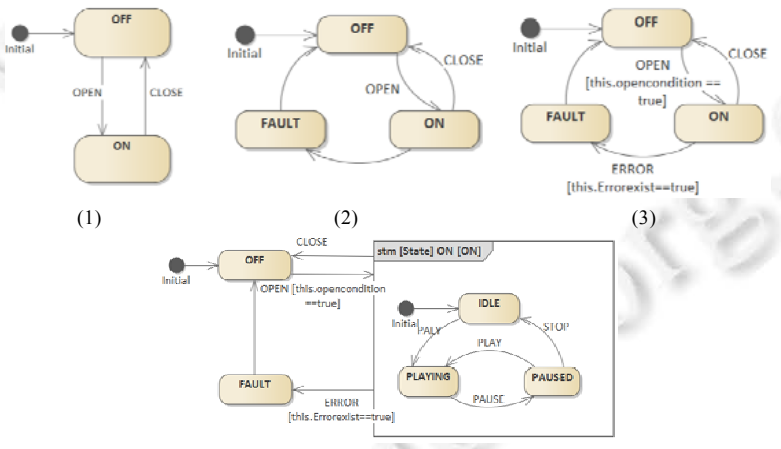


图 2 音乐播放器模型精化示例

3 模型转换: 从 SysML 到 TA 和 Simulink/Stateflow

3.1 SysML2TA模型转换

为了对非形式化的 SysML 模型进行形式化验证, 本文采用了 SysML 模型到 TA 模型的自动转换方法. 由于 SysML 状态机和时间自动机图结构非常相似, 因此 SysML2TA 模型转换较为直接, 二者的基本模型元素可以进行一对一映射.

模型转换涉及语义和语法转换两个层面. SysML2TA 模型转换框架如图 3 所示, SysML 模型和 TA 模型的语义和抽象语法符合 MOF (meta object facility)[15]对各自元模型的定义, 本文通过建立元模型的映射规则进行模型基本元素的映射实现模型转换. 而具体语法转换依赖于 XML (extensible markup language)文本模型进行,

根据 SysML 文本语法规则可解析 SysML.xml, 同时, 根据 TA 的文本语法规则生成转换后的 TA.xml (SysML 和 TA 的具体文本语法可在模型导出的 XML 文本查看, 限于篇幅, 本文不介绍具体语法). 最后使用 UPPAAL 打开 TA.xml 文本, 即可得到图形化 TA 模型.

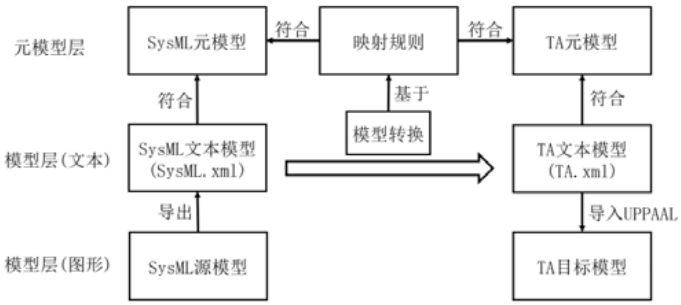


图 3 SysML2TA 模型转换框架

算法 1 是 SysML2TA 自动模型转换的伪代码, 以 SysML 模型为输入, 输出一个时间自动机网络. 算法第 3 行 *parse* (*STM*)表示解析 SysML 状态机模型, 从而得到源模型状态、迁移和变量等元素, 存储在模型元素数组 *elementArr* 中. 第 4 行–第 7 行将 SysML 模型中的每个元素 *E* 转换为相应的时间自动机元素, 并添加到 TA 模型中. 具体包括: 第 4 行表示遍历数组 *elementArr* 中每一个源模型元素 *E*; 在第 5 行, 根据 SysML2TA 映射规则 *Trans*1, 结合 TA 文本语法得到转换目标模型元素的语法模板 *template*; 第 6 行表示将转换得到的模板添加至目标 TA 模型中; 第 7 行表示所有源模型元素都已转换完成. 最终, 第 8 行给出转换得的时间自动机网络.

算法 1. SysML2TA 模型转换.

1. Input: SysML State Machine (STM);
2. Output: Network of Timed Automata (NTA).
3. *elementArr*←*parse*(*STM*)
4. **for each** *element E* in *elementArr* **do**
5. *template*←*Trans*1(*E*)
6. *NTA*.*add*(*template*)
7. **end for**
8. **return** NTA

算法 1 中的 SysML2TA 映射规则 *Trans*1 具体为 $Trans1(STM)=\langle \overline{TA}, Decl\rangle$, 即:

$$\langle S,s_0,s_f,V,T,CS\rangle \rightarrow \langle\langle L,l_0,V,Act,I,E\rangle,Decl\rangle.$$

为方便区分, 添加前缀“*sys_*”用于表示 SysML 的元素, 用前缀“*ta_*”表示时间自动机(TA)的元素.

(1) 状态的映射

- *Trans*1(*sys_S*)=*ta_L*, 状态 *sys_S* 映射为 *ta_L*, 总结得表 1 中的规则 1;
- *Trans*1(sys_s_0)=ta_l_0, 初始状态 sys_s_0 映射为 TA 中的初始位置 ta_l_0, 即表 1 的规则 2;
- *Trans*1(sys_s_f)=*ta_l*, 最终状态表示状态机行为的完成, 时间自动机中没有最终位置元素, 无迁出边的位置可表示时间自动机的结束. 因此, SysML 最终状态映射为 TA 的位置, 得到表 1 的规则 3.

(2) 变量的映射

- *Trans*1(*sys_V*)=*ta_Decl*, 状态机中的变量映射为时间自动机的全局变量声明. 其中, 数据类型 Integer, Double 和 Boolean 分别映射为 Integer, Double 和 Bool. 自定义类型映射为结构体类型 struct, 得到表 1 中的转换规则 4–转换规则 7.

(3) 迁移的映射

- *Trans*1(*sys_T*)=*ta_E*. SysML 的迁移映射为 TA 中的边, 得到表 1 的转换规则 8, 同时需保持转换前后

迁移的源节点和目标节点一致, 即 ta_l_s=*Trans*1(*sys_s*$_s$), ta_l_f=*Trans*1(*sys_s*$_f$);

- *Trans*1(*sys_Tri*)=*ta_A*. *sys_Tri* 判断是否接收到触发器信号, *ta_A* 中输入同步(?)判断是否接收到同步信号. 因此, *sys_Tri* 映射为 *ta_A* 的输入同步, 即转换规则 9;
- *Trans*1(*sys_G*)=*ta_G*. SysML 模型迁移当守卫 *sys_G* 映射为边约束 *ta_G*, 得到转换规则 10;
- *Trans*1(*sys_Eff*)=(*ta_A*,*ta_U*). *sys_Eff* 为迁移过程中执行的行为. 而在 TA 中, 边活动时会执行边上的时钟、变量更新 *ta_A* 或 *ta_U* 中的输出同步(!), 因此, *sys_Eff* 映射为 *ta_A* 和 *ta_U* 的输出同步, 即表 1 的转换规则 11、转换规则 12.

(4) 复合状态的映射

- *Trans*1(*sys_CS*)=*ta_TA*. 复合状态可映射为单个时间自动机, 即转换规则 13. 复合状态内部的状态和迁移等元素可依据情形(1)–情形(3)的规则, 即表 1 中的规则 1–规则 12 进行映射.

关于时间自动机中的位置不变式 *I*, 源 SysML 模型中没有状态不变式或类似的元素, 因此没有元素被转换为 *I*. 根据上述模型转换策略, 总结 SysML 到 TA 的具体映射规则见表 1.

表 1 SysML 到 TA 映射规则

编号	转换规则	解释
1	state2location	状态和位置的含义基本相同, 直接映射
2	initial state2initial location	初始状态映射为初始位置
3	final state2location	最终状态映射为无迁出边的位置
4	integer2integer	整型直接映射
5	double2double	浮点型直接映射
6	boolean2bool	布尔型直接映射
7	user-defined2struct	自定义类型映射为结构体
8	transition2edge	迁移映射为边
9	trigger2synchronisation	触发器约束映射为输入同步
10	guard2guard	守卫映射为约束
11	effect2synchronisation	影响映射为输出同步
12	effect2assignment	影响映射为时钟或变量更新
13	composite state2timed automata	复合状态映射为时间自动机

3.2 SysML2Simulink模型转换

为进行模型自动代码生成, 本文采用了 SysML 模型到 Simulink/Stateflow 模型的自动转换方法. Stateflow 模型本质是一种有限状态机, 其图结构与 SysML 状态机图相似, 都由状态、迁移等元素组成, 二者的模型元素基本可以直接映射. 在语义转换方面, 本文通过建立 SysML 和 Simulink 元模型的映射实现语义转换. 在语法方面, 本工作中利用 MATLAB 脚本生成 Simulink/Stateflow 模型, 即模型转换的目标文件是 Stateflow API 函数组成的 MATLAB 脚本. 具体根据 SysML 文本语法规则和 Stateflow API 规则进行转换(限于篇幅, 本文不介绍 Stateflow API 具体语法).

算法 2 是 SysML2Simulink 自动模型转换的伪代码, 以 SysML 模型为输入, 输出一个 Stateflow 模型. 算法第 3 行 *parse*(*STM*)表示解析 SysML 状态机模型, 获得模型中状态、迁移等元素及其信息, 并存储在元素数组 *elementArr* 中. 第 4 行–第 7 行是模型转换的核心, 将 SysML 模型中的每个元素 *E* 转换为相应的 Stateflow 模型元素, 包括: 第 4 行遍历源模型元素数组 *elementArr* 中的每一个源模型元素 *E*; 第 5 行根据 SysML2Stateflow 转换规则 *Trans*2, 得到目标 Stateflow 模型元素对应的 Stateflow API; 第 6 行将转换得到的 Stateflow API 添加到目标 Stateflow 模型中; 第 7 行所有源模型元素转换完毕. 第 8 行返回转换得到的 Stateflow 模型, 转换结束.

算法 2. SysML2Stateflow 模型转换.

1. Input: SysML State Machine (STM);
2. Output: Stateflow Model (SF).
3. *elementArr*←*parse*(*STM*)

4. **for each** *element E* in *elementArr* **do**
5. *Stateflow API*←*Trans*2(*E*)
6. *SF*.*add*(*Stateflow API*)
7. **end for**
8. **return** Stateflow Model

模型转换算法 2 中的 SysML2Simulink 转换规则 $Trans2(STM)=\langle S,Var,T,td,Ch\rangle$, 也就是:

$$\langle S,s_0,s_f,V,T,CS\rangle\rightarrow\langle S,Var,T,td,Ch\rangle.$$

为了方便区分, 添加前缀“*sys_*”用于表示 SysML 的元素, 用前缀“*sf_*”表示 Stateflow 的元素.

(1) 状态的映射

- $Trans2(sys\_S)=sf\_S$. SysML 和 Stateflow 状态含义一致, 可以直接映射, 得表 2 的转换规则 1;
- $Trans2(sys\_s_0)=sf\_s$. Stateflow 中没有初始状态元素, 但默认迁移指向的状态是 Stateflow 模型初始活动的状态, 因此, 初始状态 sys_s_0 映射为连接默认迁移的 *sf_s*, 得表 2 的转换规则 2;
- $Trans2(sys\_s_f)=sf\_s$. Stateflow 中也没有最终状态元素, 但是没有输出迁移的状态可以表示 Stateflow 的结束, 因此, sys_s_f 映射为无输出迁移的 *sf_s*, 即转换规则 3.

(2) 变量的映射

- $Trans2(sys\_V)=sf\_Var$. 状态机的数据类型 Integer, Double 和 Boolean 分别映射为 Int32, Double 和 Bool. 自定义类型映射为 Simulink 自定义结构体, 从而得到表 2 的规则 4–规则 7.

(3) 迁移的映射

- $Trans2(sys\_T)=sf\_T$. SysML 和 Stateflow 中的迁移都表示状态的转变, 可直接映射. 同时需保持相应的源状态和目标状态, 即 $sf_s_s=Trans2(sys_s_s)$和 $sf\_s_t=Trans2(sys\_s_t)$, 得转换规则 8;
- $Trans2(sys\_Tri)=sf\_Ev$. 触发器 *sys_Tri* 激活时迁移才能执行, 事件 *sf_Ev* 发生才能执行迁移. 因此, 迁移上的 *sys_Tri* 映射为 *sf_Ev*, 即表 2 的转换规则 9;
- $Trans2(sys\_G)=sf\_Con$. 守卫 *sys_G* 和条件 *sf_Con* 都是布尔表达式, 决策迁移是否能执行. 因此, *sys_G* 映射为 *sf_Con*, 得转换规则 10;
- $Trans2(sys\_Eff)=sf\_Act$. *sys_Eff* 是 SysML 迁移时执行的动作, 可映射为 Stateflow 状态迁移时执行的动作 *sf_Act*, 即规则 11.

(4) 复合状态的映射

- $Trans2(sys\_CS)=sf\_Ch$. 复合状态 *sys_CS* 可映射为同样包含嵌套子状态和迁移的图块 *sf_Ch*, 得表 2 的转换规则 12. 复合状态内部的状态、迁移等元素按照情形(1)~情形(3)的规则, 也就是表 2 中的规则 1–规则 11 进行映射.

根据上述模型转换策略, 总结得 SysML 到 Simulink/Stateflow 的转换规则, 见表 2.

表 2 SysML 到 Stateflow 映射规则

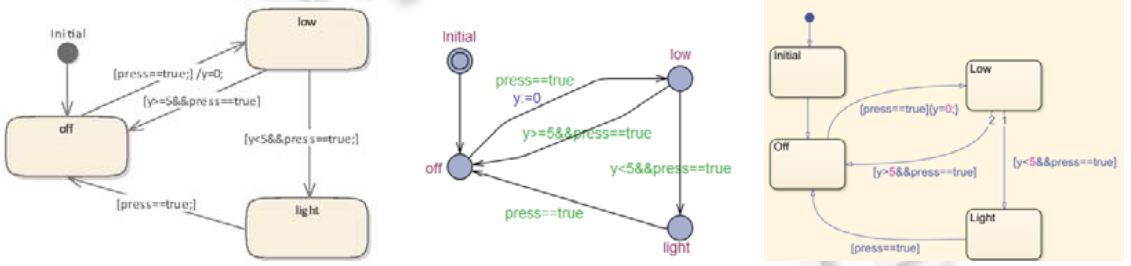
编号	转换规则	解释
1	state2state	状态直接映射
2	initial state2state	初始状态映射为默认迁移指向的状态
3	final state2state	最终状态映射为无输出迁移的状态
4	integer2int32	整型映射为 int32
5	double2double	浮点型直接映射
6	boolean2bool	布尔型直接映射
7	user-defined2strut	自定义类型直接映射
8	transition2transition	迁移直接映射
9	trigger2event	触发器映射为事件
10	guard2condition	守卫映射为约束条件
11	effect2action	影响映射为动作
12	composite state2chart	复合状态映射为图块

3.3 模型转换实例

图 4 为一个可调节亮度的灯 SysML 模型转 TA 模型的转换实例, 图 4(1)是源 SysML 模型, 包含 3 个状态 off, low 和 light, 分别表示灭、暗和亮. 变量 y 为 int 型数据变量表示时间, *press* 是布尔型数据变量表示开关. 该台灯模型意为: 台灯初始为熄灭(off)状态, 打开开关(*press*)后进入“暗”状态, 并在迁移中将“时钟 y 置 0”. 若短时间内(y<5)再次点击开关则进入“亮”状态. 处于“亮”状态时或在进入“暗”状态较长时间后(y≥5)再次点击开关, 台灯会熄灭(off).

对源模型进行 SysML2TA 模型转换, 根据算法 1 第 3 行 *parse*(*STM*)解析 SysML 模型得到其所有元素如初始状态 Initial、状态 off, low, light、变量 *press*, y 等存放至元素数组 *elementArr* 中. 根据算法 1 中第 4 行–第 7 行遍历数组中的元素 *E*, 根据表 1 总结的映射规则 *Trans*1 进行转换, 如根据表 1 的规则 2 将初始状态“Initial”转换为初始位置“Initial”, 根据规则 1 将状态“off”转换为位置“off”, 根据规则 10 和规则 11 将源迁移“off”到“low”上的守卫“*press*==true”和影响“y=0”转换为目标“off”到“low”边上的约束“*press*==true”和更新“y:=0”. 根据模型转换算法 1 第 5 行, 将转换得到的 TA 模型元素的文本语法模板 *template* 添加到 TA 模型中, 最终得到图 4(2)所示的 TA 模型.

同样的, 根据算法 2 和表 2 的映射规则进行 SysML2Simulink 模型转换, 如初始状态“Initial”根据表 2 规则 2 转换为默认迁移指向的状态“Initial”, 根据规则 1 将状态“off”转换为状态“off”, 根据表 2 中规则 10、规则 11 将源“off”到“low”迁移上的守卫“*press*==true”和影响“y=0”转换为目标模型中“off”到“low”迁移上的条件“*press*==true”和动作“y:=0”. 最后得到如图 4(3)所示的 Stateflow 模型.



(1) SysML 模型 (2) TA 模型 (3) Stateflow 模型

图 4 模型转换样例

3.4 模型转换正确性分析

基于模型转换的开发和验证中, 保证模型转换的正确性非常重要. 目前, 针对模型转换正确性的研究基本分为语法和语义两个层面[16]. 一般情况下, 当模型转换符合语法和语义正确性时, 可以证明模型转换是正确的. 对模型转换的正确性进行分析如下.

(1) 语法正确性. 语法正确性是模型转换的基本要求, 即转换得到的目标模型符合目标语法规则. 在 SysML2TA 模型转换中, 依据 TA 文本语法模板生成了文本模型 TA.xml, 且 TA.xml 能被 UPPAAL 得到 TA 图形化模型打开而不会报错, 说明转换后的模型符合 TA 语法规则. 在 SysML2Simulink 转换中, 根据 Stateflow API 语法规则生成了 MATLAB 脚本, 且脚本能运行生成 Simulink 模型, 说明 SysML2Simulink 的输出模型符合语法正确性.

(2) 语法完备性. 完备性要求转换规则对源模型元素完全覆盖. 在第 2.1 节中, 根据最小化原则和可描述性原则选取了 SysML 子集元素用于建模; 第 3.2.1 节总结的模型转换规则只对选取的 SysML 子集完全覆盖.

(3) 语义可终止性. 可终止性指源模型在有限步转换后必定得到一个确定的目标模型. 在第 3.1 节和第 3.2 节中, 在算法 1 和算法 2 中, 源 SysML 模型的解析和目标模型的生成都是在有限步骤内完成, 且由表 1 和表 2 可知, 源模型和目标模型的元素 1:1 映射, 因此转换得到的目标模型是确定且唯一的.

(4) 语义一致性. 语义一致性是模型转换最重要的标准, 要求转换前后的模型得到相同输入时具有相同

的行为. 互模拟等价[17]表示两个迁移系统之间的等价关系, 当两个系统互模拟等价时, 它们可以相互模拟彼此的动作.

对于模型转换前后的 SysML 状态机 STM 和时间自动机 TA, 根据第 3.1 节中状态的映射规则, SysML 中的每个状态 *sys_s* 和该状态转换得到的 TA 位置 *ta_l* 满足互模拟关系, 即$(sys_s,ta_l)\in R$. 根据变量和迁移的映射规则, 保证了模型转换前后迁移的约束和行为方式相同. 因此在相同输入情况下, 经过相同的动作α, *ta_l* 的迁移目标状态 *ta_l′*是由 *sys_s* 的迁移目标状态 *sys_s′*映射得到. 因此, *sys_s′*和 *ta_l′*也满足互模拟关系. 即对于任意状态对$(sys_s,ta_l)\in R$, 公式 a)成立; 根据映射规则, SysML 状态机和 TA 的每个模型元素一一映射, 同理, 公式 b)也成立.

a) 当 $sys\_s \xrightarrow{\alpha} sys\_s'$, 存在 $ta\_l \xrightarrow{\alpha} ta\_l'$, 使得$(sys_s',ta_l')\in R$;

b) 当 $ta\_l \xrightarrow{\alpha} ta\_l'$, 存在 $sys\_s \xrightarrow{\alpha} sys\_s'$, 使得 $(ta\_l',sys\_s')\in R$.

因此, 根据互模拟等价的定义, 源 SysML 状态机和目标时间自动机满足互模拟关系, 保持行为等价, 模型转换前后保持语义一致性.

SysML2Simulink 和 SysML2TA 模型转换算法和映射规则相似, SysML 和 Stateflow 模型元素一一映射, 图结构基本一致; 同理, 源 SysML 状态机和目标 Stateflow 模型满足互模拟关系, 保持语义一致性.

4 模型转换工具实现

本方法提出的模型转换工具在Eclipse开发环境中基于Java语言编程实现, 模型转换工具的主要步骤及其代码数量统计见表 3. 模型转换工具主要分为 3 大模块: SysML 模型解析模块、TA 模型生成模块以及 Simulink/Stateflow 生成模块.

表 3 模型转换工具实现主要步骤

工具界面	模型转换工具步骤		执行功能	代码行数
	–		显示工具实现的可视界面	150+
模型解析模块	1. 获取每个状态机		从源 SysML 模型文本文件中获取每个状态机	100+
	2. 解析状态机	解析状态	解析源模型文件, 获取状态节点信息, 并构建状态数组	100+
		解析迁移	解析源模型文件, 获取迁移信息, 并构建迁移数组	200+
		解析变量	解析源模型文件, 获取数据变量、结构体等信息, 并构建变量数组	100+
TA 模型生成模块	3. 构建 template 部分	生成 TA 模型的 declaration 部分	基于表 1 变量的转换规则, 生成符合 UPPAAL 中 TA 规范的 declaration	100+
		生成 location	根据 state 转换规则, 生成符合 UPPAAL 中 TA 规范的 location	50+
		生成 transition	根据 transition, guard, trigger 和 event 转换规则, 生成符合 UPPAAL 中 TA 规范的 transition	250+
Stateflow 模型生成模块	4. 构建 Stateflow	生成新建 Stateflow 模型声明	根据 SysML 模型名等信息, 生成创建 Stateflow 模型的 Stateflow API 脚本	50+
		生成添加 Stateflow 模型具体元素声明	根据源模型元素数组和表 2 转换规则, 生成添加目标 Stateflow 模型内元素的 Stateflow API 函数	500+

其中, 模型解析模块基于 DOM (document object model)接口实现转换算法中对源模型文本文件的解析. 根据 SysML 状态机各个元模型的特性, 我们定义了多个类, 如定义了迁移类用于存储迁移的源节点、目标节点、触发器、守卫等信息. 解析源模型得到的信息保存在元素数组中. 在模型生成模块, 工具会遍历元素数组,根据模型转换规则结合 TA 模型和 Stateflow 模型的语法, 使用 Java 文件输出流生成相应的模型文本文件(TA.xml 和 Stateflow.m), 最后使用 UPPAAL 和 MATLAB 工具分别打开相应的模型文件, 可得到目标模型.

模型转换工具代码结构及界面如图 5 所示, 图左侧第 1 个包为自定义类, 用于存储源模型各类元素信息. 文件 TransformEntry 为工具入口类, 文件 Parser 实现模型解析模块, 文件 StateFlowTransformlmpl 实现 Stateflow 模型生成模块, TimeAutomataTransformImpl 则实现了 TA 模型生成模块.

图 5 中弹窗为模型转换工具的可视化界面, “Model File”栏用于输入源 SysML 模型文件, “Save Path”用于选择目标模型文件的保存路径, 点击按键“UPPAAL”或“Simulink”后, 工具会进行相应的模型转换.

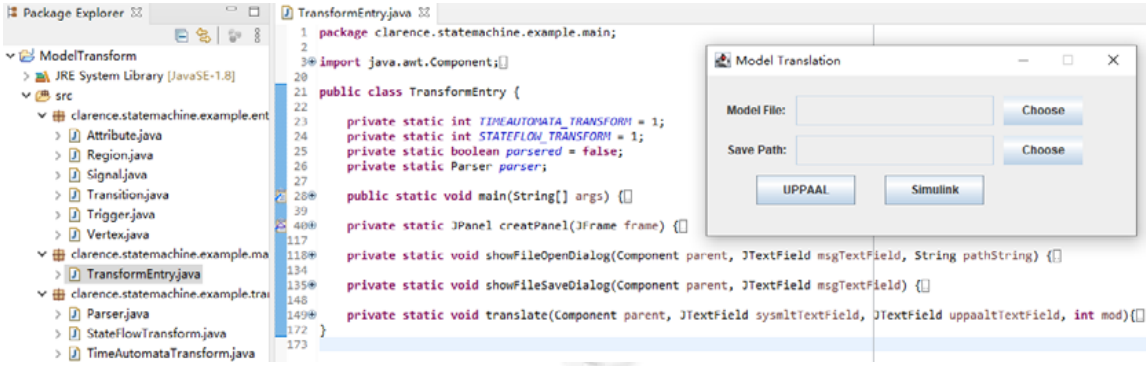


图 5 模型转换工具代码结构及界面

5 实例研究

在本文提出的方法中: 为了降低开发难度, 提出了基于 SysML 的分层精化建模技术; 为了保障模型安全性和正确性, 提出了基于 UPPAAL 的模型转换和验证方法; 为了提高编码效率, 提出了基于 Simulink 的模型自动转换和自动编码. 本节以自动飞行控制软件(automatic flight control software, AFCS)为例, 验证本文提出的方法的可行性.

5.1 自动飞行控制软件需求确认

AFCS 主要负责控制飞机的轨迹、姿态及速度, 根据飞机进行纵向或横向运动可分为纵向运行控制模式和横向运动控制模式, 且同一时刻只会进入一个运动控制模式[18].

本案例主要对自动飞行控制软件的纵向运动进行分析建模. 如图 6 所示: 在纵向运动控制模式中, 驾驶员可以自主开启/关闭 AFCS, 可以预设飞行高度, 当预设高度超过目前飞行高度时, 飞机爬升; 当预设高度低于目前飞行高度时, 飞机下降. 为了保障软件故障时不影响飞行, 软件运行时会自检, 在故障时告警并停止工作, 驾驶控制权限完全交由驾驶员.

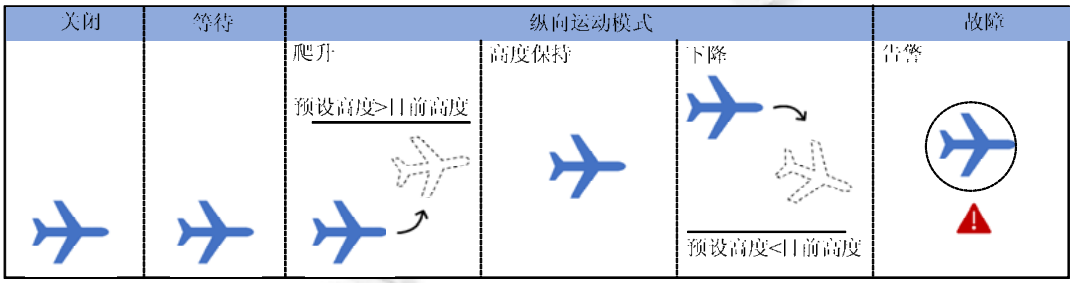


图 6 自动飞行控制软件工作图

通过对软件分析, 得到自动飞行控制软件的功能性需求如下.

- REQ-1: 自动飞行控制软件的开启和关闭由驾驶员控制;
- REQ-2: 软件运行前会进行自检;
- REQ-3: 自动飞行控制软件可以控制飞机纵向运动;
- REQ-4: 软件能够在发生故障时告警.

得到自动飞行控制软件的安全性需求如下.

- REQ-5: 自动飞行控制软件的功能正常运行.
- REQ-6: 软件运行中不会发生死锁;
- REQ-7: 检测到故障时, 自动飞行控制软件不再操控飞机飞行;
- REQ-8: 为了保障安全, 自动控制软件在飞行高度在 7000 米~12000 米之间执行飞行任务.

为了便于进行工程管理, 如图 7 所示, 使用 SysML 需求图描述自动飞行控制软件的需求.

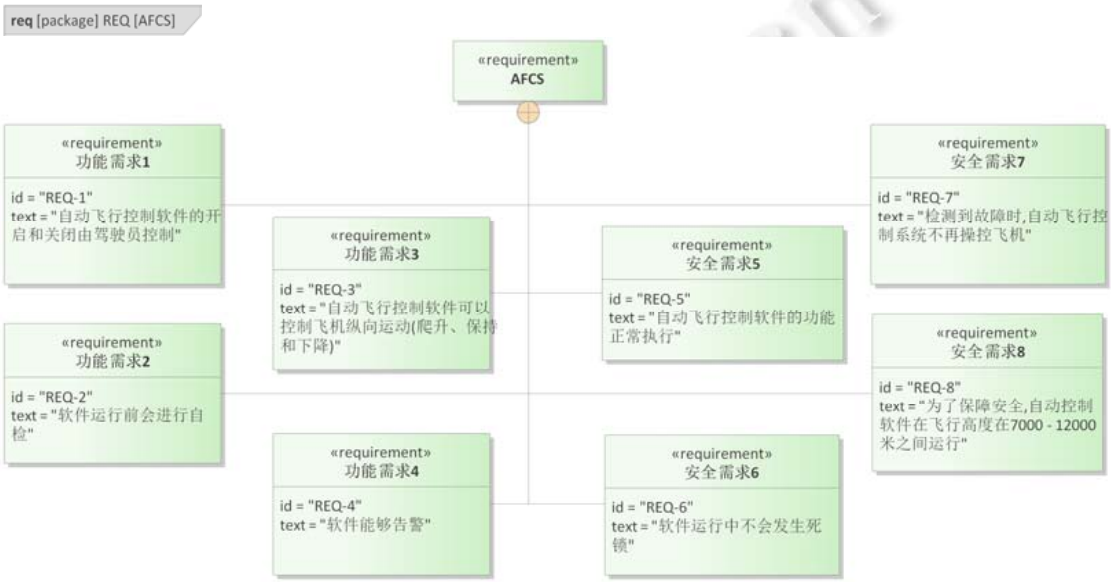


图 7 自动飞行控制软件需求图

5.2 自动飞机控制软件分层建模与验证

在本小节中, 将基于 SysML 对 AFCS 的纵向运动模式进行分层精化建模. 通过分析 AFCS 软件, 见表 4, 将模型划分为 3 层. 对每层 SysML 模型验证不同的需求性质, 直到所有需求都通过模型检验.

表 4 SysML 设计模型分层说明

层级	模型名	说明
1	AFCS 模型 1	将自动飞行控制软件抽象建模出大致的工作状态, 划分出软件关闭、就绪、运行和故障状态
2	AFCS 模型 2	在 AFCS 模型 1 的基础上对状态之间的迁移进行精化. 软件运行过程中会不断检测是否有故障, 因此还在运行状态内添加故障检测状态
3	AFCS 模型 3	在 AFCS 模型 2 的基础上, 进一步对软件运行时故障的检测情况进行了精化

下面介绍 AFCS 模型 1–模型 3 的分层精化建模和验证过程.

5.2.1 AFCS 模型 1 建模与验证

(1) AFCS 模型 1 建模

根据第 5.1 节中的需求分析, 建立初始 SysML 模型 1 如图 8 所示. 初始自动飞行控制软件模型的状态划分为关闭、就绪、运行和故障, 即 $sys\_S(AFCS1)=\{OFF,IDLE,PERFORM,FAILURE\}$. 运行状态(PERFORM)内部嵌套了保持、爬升和下降状态, 即 $sys\_S(PERFORM)=\{CURISE,CLIMB,DESCENT\}$. 状态之间的迁移条件比较简单, 如 $sys_t(OFF,IDLE)=(TurnOn,\varnothing,\varnothing)$表示在接收到开启信号后, 软件进入就绪等待状态.

(2) AFCS 模型 1 验证

对 SysML 模型进行转换, 具体操作为导出图 8 所示初始 SysML 模型的文本模型(SysML.xml), 使用第 4

节介绍的模型转换工具进行转换, 生成目标 TA 模型文件(TA.xml), 在 UPPAAL 中打开得到图 9 所示目标时间自动机模型. 其中, SysML 的状态 initial, OFF, FAILURE, IDLE, PERFORM, CURISE, CLIMB, DESCENT, Final 对应 TA 模型中的位置(命名相同), 状态间的迁移对应为位置之间的边, 迁移上的可选信息如触发器 TurnOn, TurnOff 等对应输入同步信号 TURNON_SIGNAL 和 TURNOFF_SIGNAL 等.

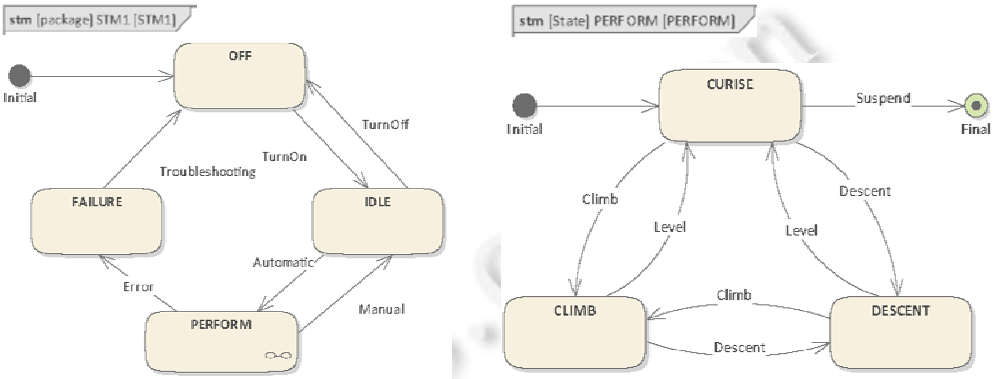


图 8 自动飞行控制软件 SysML 模型 1

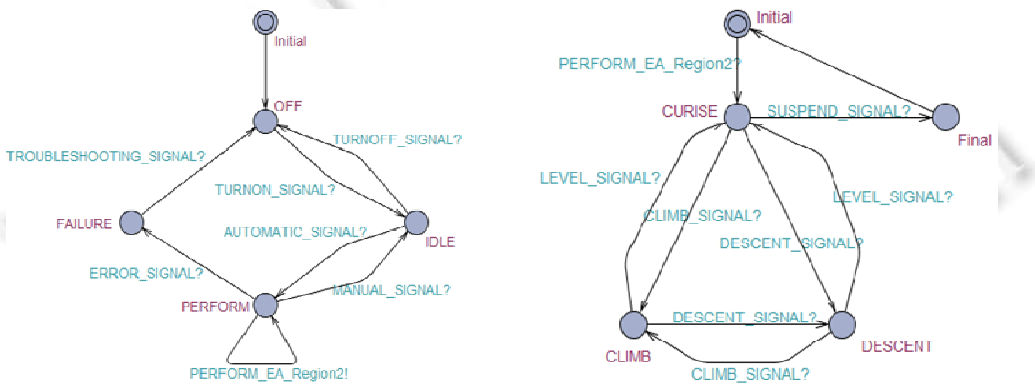


图 9 自动飞行控制软件 TA 模型 1

TA 模型 1 映射了所有源 SysML 模型 1 中的元素, 说明了模型转换的语法完备性. TA 模型可被 UPPAAL 接收(若语法有错误, TA 模型不能正常显示), 说明了转换语法正确性. 源 SysML 经过有限步转换得到了确定的 TA 模型 1, 说明了模型转换的语义可中止性. 源 SysML 模型的状态、变量、迁移一一映射为 TA 模型的位置、变量、边, 保障了两个模型初始状态一致, 对事件的处理一致, 在相同输入的情况下保持等价的行为特征,理论上可保障模型转换前后语义一致性.

图 8 所示源 SysML 模型初始处于 OFF 状态, 接收到打开信号(TurnOn)时, 进入 IDLE 状态等待自动控制命令(automatic), 若接收到自动控制命令, 则进入运行状态, 即 $OFF \xrightarrow{TurnOn} IDLE \xrightarrow{Automatic} PERFORM$.

目标 TA 模型初始处于 OFF 状态, 接收到打开信号(TURNON_SIGNAL), 会迁移至 IDLE 位置, 并在接收到自动控制命令(AUTOMATIC_SIGNAL)的情况下迁移至 PERFORM 状态, 即:

$$OFF \xrightarrow{TURNON\_SIGNAL} IDLE \xrightarrow{AUTOMATIC\_SIGNAL} PERFORM.$$

SysML 模型 1 和 TA 模型 1 在相同输入的情况下保持相同的行为特征, 证明了模型转换的语义一致性.

对部分待验证的需求进行形式化提取, 得到如下 TCTL 公式.

- REQ-3: 自动飞行控制软件可以控制飞机纵向运动. 用可达性验证 AFCS 是否可以控制飞行纵向运动模式: 爬升(CLIMB)、下降(DESCENT)以及巡航(CURISE).

- *E*<>*PERFROM_SUB.CLIMB*;
- *E*<>*PERFORM_SUB.DESCENT*;
- *E*<>*PERFORM_SUB.CURISE*.

- REQ-4: 软件能够在发生故障时告警. 由于模型还不够细化, 因此在模型 1 对软件能否告警(FAILURE)进行验证, 即验证是否能到异常状态.
  - *E*<>*AFCS.FAILURE*.
- REQ-5: 自动飞行控制软件的功能正常运行. 使用可达性验证 AFCS 能执行所有功能: 关闭(OFF)、就绪(IDLE)、执行(PERFORM)等.
  - *E*<>*AFCS.OFF*;
  - *E*<>*AFCS.IDLE*;
  - *E*<>*AFCS.PERFROM*;

  以及 REQ-3 和 REQ-4 的部分.
- REQ-6: 软件运行中不会发生死锁.
  - *A*[] not *deadlock*.

在 UPPAAL 中进行模型检验的结果如图 10 所示, 所有待验证性质通过验证. 例如性质 *E*<>*AFCS.IDLE*, 初始时模型位于 Initial 位置, 由于到 OFF 的迁移上没有约束, 模型迁移至 OFF 位置, 当接收到 TurnOn 信号时, 可以跳转至 IDLE 位置, 说明 IDLE 是可达的, 性质成立.

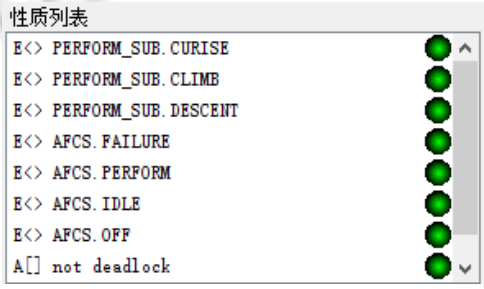


图 10 TA 模型 1 验证结果

5.2.2 AFCS 模型 2 建模与验证

(1) AFCS 模型 2 建模

AFCS 模型 2 如图 11 所示, *sys_S*(*AFCS*2)={*OFF*,*FAILURE*,*IDLE*,*PERFORM*}, *sys_S*(*PERFORM*)={*CURISE*, *CLIMB*,*ERRORCHECK*,*DESCENT*}. 模型 2 主要是在模型 1 的基础上对状态间的迁移进行了精化, 并在复合状态“PERFORM”中添加了新状态“ERRORCHECK”.

对部分精化规则的使用进行说明如下.

- 应用精化规则 1,在模型 1 的基础上增加了新的状态 *s*, *name*(*s*)=*ERRORCHECK*,用于在运行过程中检查软件故障;
- 应用精化规则 3, 在模型 1 的“CLIMB”和“DESCENT”间添加新的迁移 *t*′=(*CLIMB*,∅,∅,*eff*′,*CLIMB*), 其中, *eff*′:=*this.SystemInfo.CurrentHeight*=*this.SystemInfo.CurrentHeight*±50, 用于模拟 AFCS 根据给出命令进行高度调整的操作;
- 应用精化规则 5, 在模型 1 的基础上, 对模型中迁移 *t*(*CURISE*,*CLIMB*)=(*CURISE*,*climb*,∅,∅,*CLIMB*)进行精化, 添加守卫 *g*′:=*this.Climb.Height*>*this.SystemInfo.CurrentHeight*. 结合规则 1, 添加了节点“Choice”, 得到迁移:

$$t_1'(CLIMB,CHOICE)=(CLIMB,climb,\varnothing,\varnothing,CHOICE)$$
$$t_2'(CHOICE,CLIMB)=(CHOICE,\varnothing,g',\varnothing,CLIMB)$$

- 应用精化规则 6, 在模型 1 的基础上, 对模型 1 中“IDLE”到“PERFORM”的迁移 $t(IDLE,PERFORM)=(IDLE,automatic,\varnothing,\varnothing,PERFORM)$, 添加新的影响 $eff:=this.SystemInfo.ManualOperated$=false, 得到了新的迁移 $t'(IDLE,PERFORM)=(IDLE,automatic,\varnothing,eff,PERFORM)$, 用于更新系统消息中驾驶员是否手动控制.

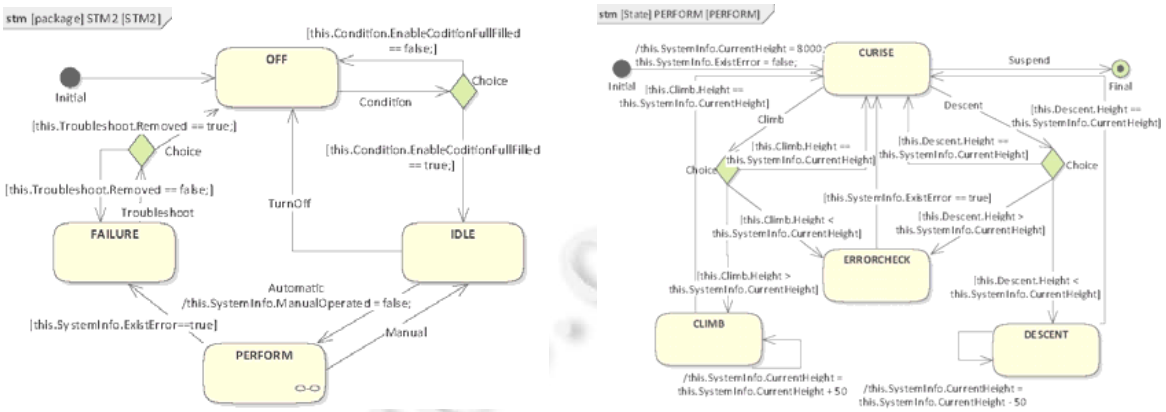


图 11 自动飞行控制软件 SysML 模型 2

(2) AFCS 模型 2 验证

利用模型转换工具, 将图 11 所示 SysML 模型 2 转换成 TA 模型 2, 转换结果如图 12 所示. 模型转换正确性理论分析见第 3.4 节, 结合具体实例的模型转换正确性分析见第 5.2.1 节, 这里不再赘述.

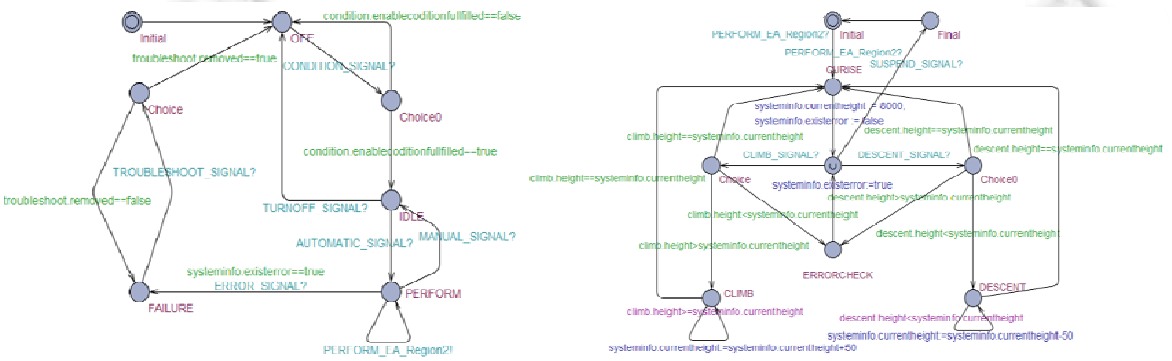


图 12 自动飞行控制软件 TA 模型 2

接下来, 从需求中提出形式化性质进行模型检验. 在第 5.2.1 节中检验的形式化性质的基础上, 补充需求 2、需求 3、需求 7 的形式化性质如下.

- REQ-2: 软件运行前会进行自检.

AFCS 在迁移至就绪状态(IDLE)前, 需要先进行运行条件的检测, 当运行条件均满足($condition.enableconditionfullfilled$==true)时, 才能进入就绪状态:

$A[]$ ($AFCS.OFF$ and $condition.enablecoditionfullfilled$==true) imply ($AFCS.OFF$ or $AFCS.IDLE$).

当 AFCS 处于就绪状态时, 运行所需条件必定被满足:

$A[]$ $AFCS.IDLE$ imply $condition.enablecoditionfullfilled$==true.

- REQ-3: 自动飞行控制软件可以控制飞机纵向运动.

AFCS 处于巡航控制(CURISE)时, 接收到爬升信号(climb)且目标高度高于当前巡航高度, AFCS 控制飞机爬升(CLIMB):

(*PERFORMSUB.CURISE* and *climb.height*>*systeminfo.currentheight*)→
(*PERFORMSUB.CURISE* or *PERFORMSUB.CLIMB*).

AFCS 巡航控制(CURISE)时, 接收到下降信号(descent)且目标高度低于当前巡航高度, 会控制飞机下降(DESCENT):

(*PERFORMSUB.CURISE* and *descent.height*!=0 and *descent.height*<*systeminfo.currentheight*)→
(*PERFORMSUB.CURISE* or *PERFORMSUB.DESCENT*).

- REQ-7: 检测到故障时, 自动飞行控制软件不再操控飞机飞行.

AFCS 正常运行(PERFORM)时, 一旦检测到故障(*systeminfo.existerror*==true), 则跳转至故障状态(FAILURE), 不再执行飞行任务:

A[] (*AFCS.PERFORM* and *systeminfo.existerror*==true) imply (*AFCS.PERFORM* or *AFCS.FAILURE*).

若飞机处于故障状态, 说明检测到了故障:

A[] *AFCS.FAILURE* imply *systeminfo.existerror*==true.

对模型进行验证, 当 AFCS 处于 CURISE 状态(复合状态 PERFORM 的子状态)时, 接收到爬升或者下降指令, 会进行预设高度的检测, 若超出性能范围, 则表明系统运行异常(*systeminfo.existerror*==true), 则结束 AFCS 的正常运行, 跳转至异常状态(FAILURE), 即模型满足 *A*[] (AFCS.PERFORM and *systeminfo.existerror*==true) imply (*AFCS.PERFORM* or *AFCS.FAILURE*). 其他性质验证结果如图 13 所示, 待验证性质都被满足.

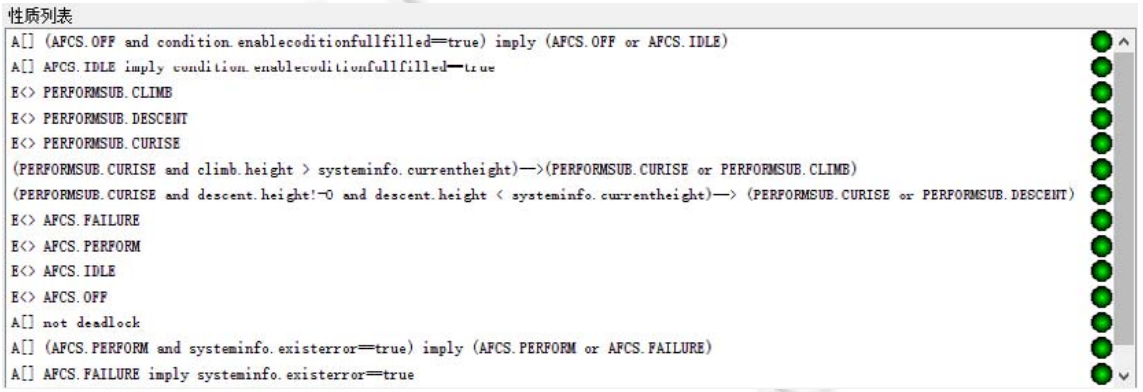


图 13 TA 模型 2 验证结果

5.2.3 AFCS 模型 3 建模与验证

(1) AFCS 模型 3 建模

模型 3 进一步对软件运行时故障的检测进行了处理, 模型 2 中的简单状态“ERRORCHECK”精化成为了如图 14 所示的复合状态(限于篇幅, 本文不再列出所有的第 3 层 SysML 模型和 TA 模型, 完整第 3 层模型文件见模型网盘 https://pan.baidu.com/s/1Jm2dMRJiDLegi6rwL5C5yA?pwd=q7m9).

对部分精化规则的应用进行说明如下.

- 应用精化规则 2, 在模型 2 的基础上, 将“ERRORCHECK”精化为复合状态. 对 AFCS 的系统信息、预设高度以及手动航行等进行验证检测;
- 应用精化规则 4, 对“NORMAL”到“MANUALCHECK”的迁移添加了触发器 *tri*':=*Manual*, 用于对“人工操控”发生时进行处理.

此外, 建立矩阵来展示需求和设计模型之间的可追溯关系. 如图 15 所示, 通过追溯矩阵可以看出需求和设计模型之间的追溯关系, 如功能需求 3 “REQ-3: 自动飞行控制软件可以控制飞机纵向运动(爬升、下降和巡航)”在 SysML 模型中以“CLIMB”, “DESCENT”以及“CURISE”状态进行描述. 通过查看需求追溯矩阵, 可知需求完整地描述为状态机模型.

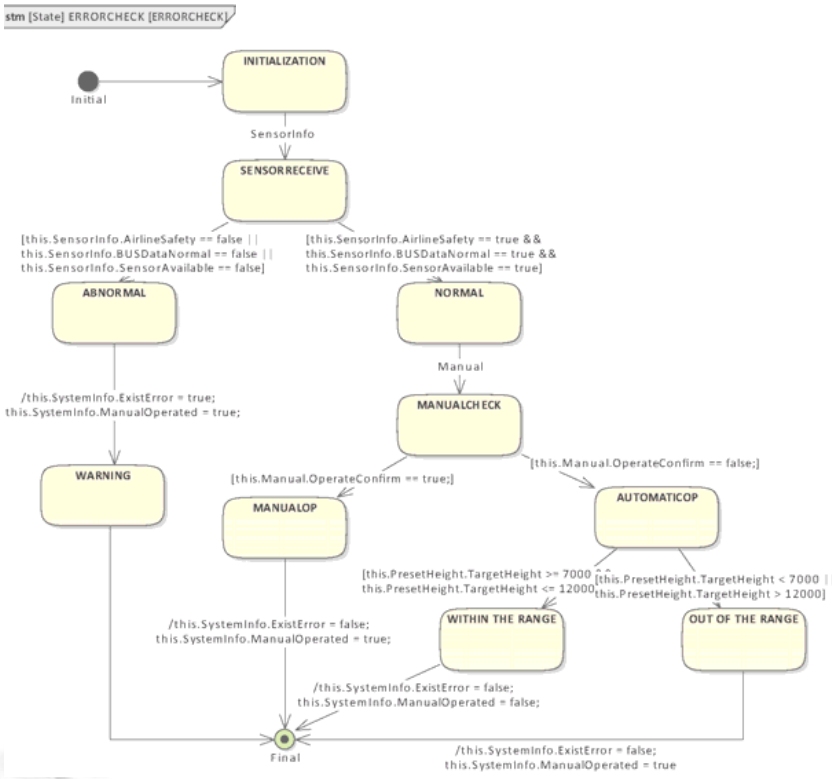


图 14　自动飞行控制软件 SysML 模型 3 中的“ERRORCHECK”

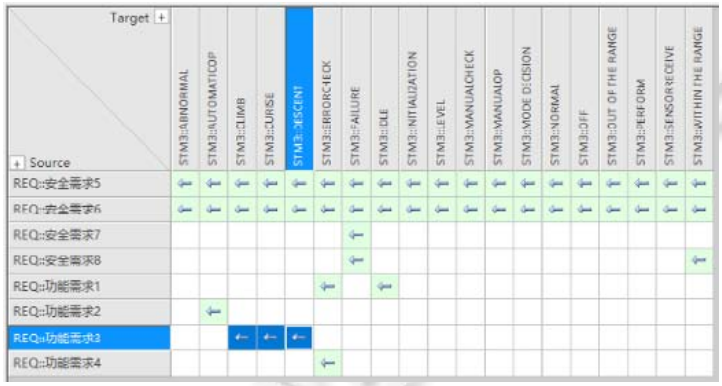


图 15　追溯关系矩阵

(2) AFCS 模型 3 验证

利用模型转换工具, SysML 模型 3 复合状态“ERRORCHECK”转换的 TA 模型如图 16 所示, 完整 TA 模型 3 见模型网盘.

得到了完整描述需求的 TA 模型 3 后, 将所有的需求进行形式化约束提取并进行模型检验. 在第 5.2.1 节和第 5.2.2 节中 SysML 模型 1 和 SysML 模型 2 形式化需求的基础上, 补充需求 REQ-1, REQ-4, REQ-8 的形式化性质如下.

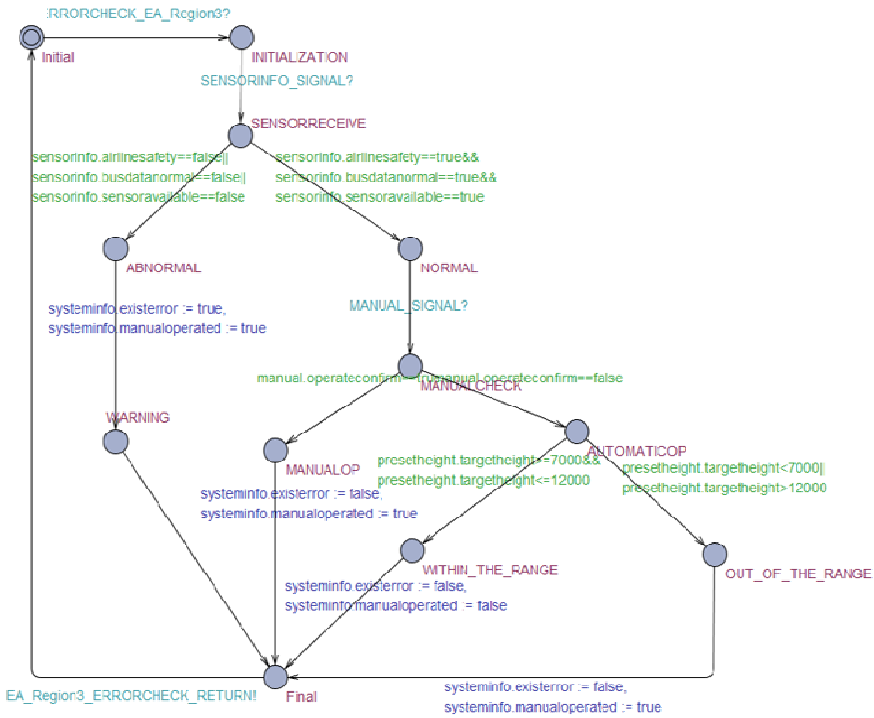


图 16 自动飞行控制软件 TA 模型 3 中的"ERRORCHECK"

- REQ-1: 自动飞行控制软件的开启和关闭由驾驶员控制.

当 AFCS 就绪(IDLE)且飞机在安全高度(7000<*height*<12000)时, 接收到驾驶员打开 AFCS 的信号(automatic), AFCS 会开始运行(PERFORM):

$$A[]\ (AFCS.IDLE \text{ and } automatic.height>7000 \text{ and } automatic.height<12000)$$
$$\text{imply } (AFCS.IDLE \text{ or } AFCS.PERFORM).$$

当 AFCS 处于运行状态(PERFORM)且没有故障(existerror)时, 收到驾驶员手动操控信号(manual operated)时, AFCS 会进入空闲状态(IDLE):

$$A[]\ (AFCS.PERFORM \text{ and } systeminfo.existerror==\text{false} \text{ and } systeminfo.manualoperated==\text{true})$$
$$\text{imply } (AFCS.IDLE \text{ or } AFCS.PERFORM).$$

- REQ-4: 软件能够在发生故障时告警.

AFCS 在巡航状态(CURISE)自检时发现异常(systeminfo.existerror==true)会发出警报(WARNING), 并停止执行飞行任务进入故障状态(FAILURE):

$$(PERFORMSUB.CURISE \text{ and } systeminfo.existerror==\text{true})\rightarrow ERRORSUB.WARNING$$
$$(PERFORMSUB.CURISE \text{ and } systeminfo.existerror==\text{true})\rightarrow AFCS.FAILURE$$

- REQ-8: 为了保障安全, 自动控制软件在飞行高度在 7000~12000 米之间执行飞行任务.

当飞行执行飞行任务(PERFORM)时, 即操控飞机爬升(CLIMB)、降落(DESCENT)以及巡航(CURISE)时, 飞机所处的高度必定在安全高度(7000~12000).

$A[]\ PERFORMSUB.CURISE$ imply ($systeminfo.currentheight \geqslant 7000$ and $systeminfo.Currentheight \leqslant 12000$)

$A[]\ PERFORMSUB.CLIMB$ imply ($systeminfo.currentheight \geqslant 7000$ and $systeminfo.currentheight \leqslant 12000$)

$A[]\ PERFORMSUB.DESCENT$ imply ($systeminfo.currentheight \geqslant 7000$ and $systeminfo.currentheight \leqslant 12000$)

对 TA 模型 3 进行性质验证, 如验证 REQ-8 提取 TCTL 性质, 若当前飞行高度超出安全高度范围

(*currentheight*<7000||*currentheight*>12000)会迁移至 OUT_OF_THE_RANGE 位置, 再执行到位置 Final 的迁移, 将 existerror 更新为 true, 意味着模型检测到故障, 不能再执行飞行任务(CURISE, CLIMB 或 DESCENT). 因此, 自动控制软件在执行飞行任务时必定处于安全高度,REQ-8 成立.

全部待验证 TCTL 公式以及其验证结果如图 17 所示, TA 模型 3 满足了所有从需求中提取的性质. 意味着 SysML 模型 3 满足 AFCS 所有需求, 可以进行下一步工作, 将 SysML 模型 3 转换至 Simulink 进行代码生成.

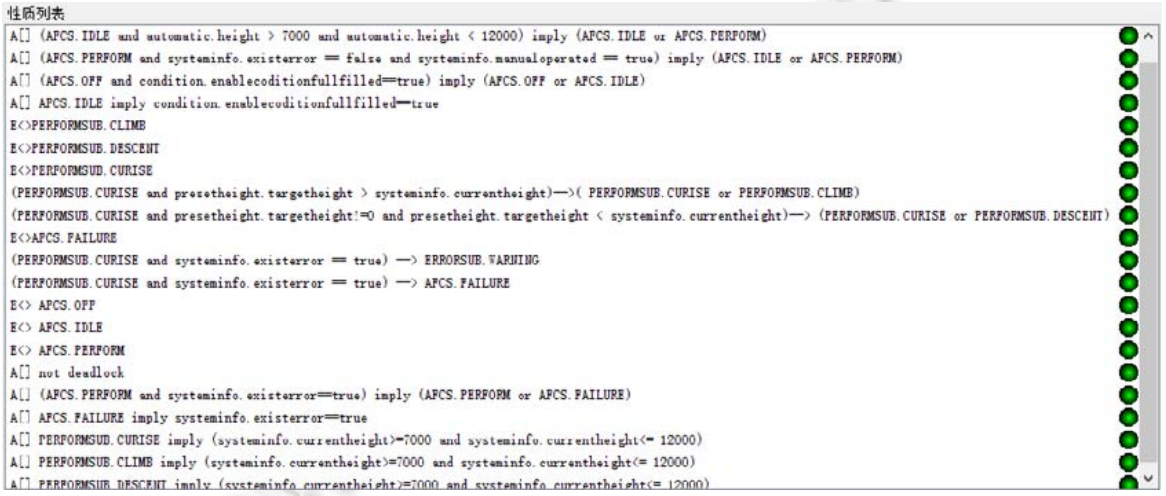


图 17 TA 模型 3 验证结果

5.3 自动飞机控制软件代码生成

利用模型转换工具对源 SysML 模型进行解析和模型转换, 得到相应的 MATLAB 脚本. 然后, 使用 MATLAB 引擎编译运行脚本生成 Simulink 模型, 模型顶层如图 18 所示. 模型转换在有限步内完成, 且得到的 Simulink 模型编译通过, 目标 Simulink 模型包含了源 SysML 模型的所有信息, 可知模型转换语法正确性、完备性和语义可终止性. 源 SysML 模型和目标 Simulink 模型初始状态、变量、迁移、状态一一映射, 在相同输入的情况下行为特征一致, 保持了语义一致性. 最后, 使用 Simulink Coder 生成的部分源代码如图 19 所示.

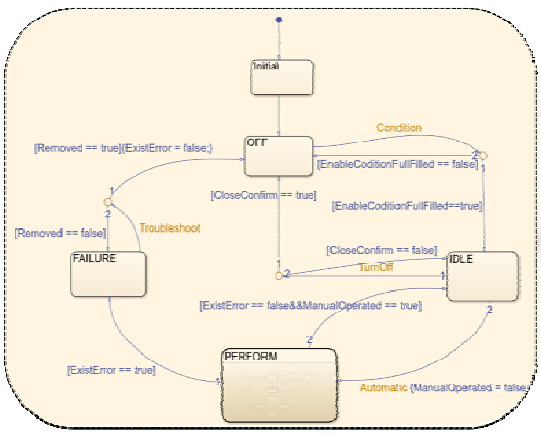


图 18 自动飞行控制软件 Simulink 模型

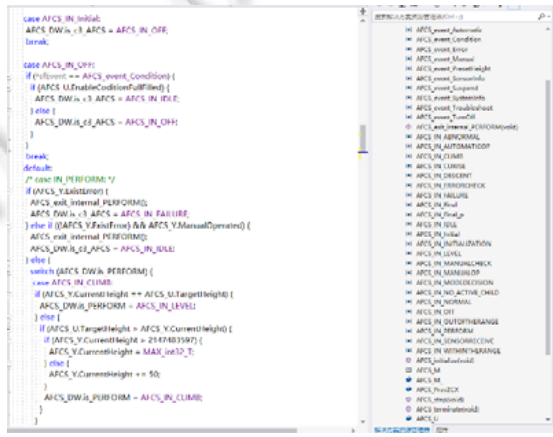


图 19 Simulink 模型生成的部分源代码

5.4 开发和验证过程适航标准审定

为保障软件开发和验证过程的安全性, 根据 DO-331 和 DO-333 标准中相应的目标对案例研究的开发和验

证过程进行审查. 表 5 和表 6 分别记录了软件开发过程和形式化验证过程中所满足的适航审定目标. 通过适航标准审查, 可证明本工作提出的机载软件开发和验证方法符合部分 DO-331 和 DO-333 标准.

案例研究通过自动飞行控制软件实践了本文提出的开发与验证方法. 通过模型转换和提取形式化需求来验证模型是否满足功能需求和安全需求, 从而保障了机载软件模型的正确性和安全性. 审查案例中的开发和验证过程发现可满足 DO-331 标准和 DO-333 标准部分目标, 保障了过程的正确性、安全性. 因此, 案例研究证明了本方法可以有效保障机载软件的正确性和安全性.

表 5 开发过程满足的 DO-331 目标

目标编号	目标描述	解释说明	本文参考章节
A-1.1	定义了软件生命周期过程的活动	定义了软件生命周期中需求、设计模型和编码过程的活动	1.2
A-1.3	选择并定义了软件生命周期环境	对每个软件生命周期活动环境进行了介绍说明	1.2
A-1.6	软件计划符合 DO-331 文件要求	在 DO-331 审查目标的指导下建立本工作软件开发计划	1.2
A-2.1	开发了高级需求	初始 SysML 模型描述了高层需求	5.2
A-2.4	开发了低级需求	精化 SysML 模型描述了低层需求	5.2
A-2.6	开发了源代码	使用 Stateflow 模型生成源代码	5.3
A-3.1	高级需求符合系统需求	高级需求模型依据系统需求建立	5.2
A-3.2	高级需求是准确且一致的	SysML 状态机图描述的高级需求是准确的, 模型转换后验证通过说明了与需求一致性	5.2
A-3.4	高级需求是可验证的	高级需求模型转换为 TA 模型进行形式化验证	5.2
A-3.5	高级需求符合标准	高级需求模型符合 SysML 语法标准	5.2
A-3.6	高级需求可追溯至系统需求	高级需求模型与系统需求之间建立了追溯矩阵	5.2
A-4.1	低级需求满足高级需求	低级需求模型由高级需求模型精化得到	5.2
A-4.2	低级需求准确且一致	使用 SysML 状态机图来描述低级需求, 且模型转换后能通过验证	5.2
A-4.4	低级需求可验证	低级需求模型转换为形式化模型并验证	5.2
A-4.5	低级需求符合标准	使用系统建模语言 SysML 来描述需求	5.2
A-4.6	低级需求可追溯至高级需求	低层模型由高层模型精化得到	5.2
A-5.1	源代码符合低级需求	源代码由低级需求模型转换至 Stateflow 模型后生成	5.3
A-5.4	源代码符合标准	源代码由 Simulink 生成且符合计算机语言标准	5.3

表 6 验证过程满足的 DO-333 目标

目标编号	目标描述	解释说明	本文参考章节
FM.A-4.14	形式化分析案例和过程是正确的	案例研究中对转换的模型进行形式化验证, 结果正确	5.2
FM.A-4.15	形式化分析结果是正确的且解释差异	模型验证结果正确	5.2
FM.A-4.16	需求的形式化是正确的	需求的形式化符合 TCTL 语法	5.2
FM.A-4.17	形式化方法是正确定义且合理恰当的	使用 UPPAAL 工具来进行形式化验证, 模型由设计模型转换, TCTL 约束从需求中提取	1.2, 5.2

6 相关工作与比较

(1) 模型驱动开发相关工作

在软件开发过程, 业内通常采用模型驱动进行开发. 如文献[19]使用基于模型的系统工程(model-based systems engineering, MBSE)开发方法和 SysML 建模语言对卫星通信系统架构进行了设计和分析. 文献[20]提出了一种精简且高度自动化的基于模型的软件开发流程, 利用 Simulink 工具进行模型开发、测试以及代码生成, 从而减少手动开发工作, 同时保持较高的软件质量. 文献[21]使用时间自动机对多处理器实时系统进行建模, 便于后续在 UPPAAL 中验证系统可调度性, 实现对系统性能的分析. 文献[22]基于 Event-B 对航天器内存管理系统进行了建模和形式化验证. 在机载软件的开发方面, 文献[19,20]使用半形式化语言建模, 可以降低建模难度但不利于进行形式化验证; 文献[21,22]使用形式化语言建模便于进行验证且无二义性, 但是对开发人员来说理解形式化语言并不容易. 本文提出的方法基于半形式化的 SysML 状态机子集进行建模, 使开发工作较为简便. 同时, 为了方便进行自动代码生成和形式化模型检验, 我们还将半形式化的 SysML 模型转换为了 Simulink 模型和形式化 TA 模型.

在模型精化方面, 通常使用基于 Event-B[22,23]的方法. 如文献[22]基于 Rodin 工具, 利用 Event-B 对航天器

内存管理系统进行分层模型开发, 通过逐步精化得到设计模型, 从而保障上下层模型的一致性. 但是 Event-B 对开发人员来说较难理解, 且模型元素有限不便进行系统工程管理. 本文参考了文献[22,23]分层精化建模的思想, 提出了基于 SysML 状态机图的分层精化建模方法, 并总结了相应的精化规则. 但由于目前业内没有相应的 SysML 自动精化工具, 基于 SysML 的精化建模需要手动完成.

(2) 形式化验证相关工作

在安全关键领域, 常使用形式化方法对软件进行分析和验证. 文献[24]中, 美国喷气推进实验室使用 SPIN 工具检验火星探测器的控制系统, 发现了一些发射前可以改正的错误. 文献[25]将 SysML 模型集成到 ABD (abstract block diagram)公理集中来验证自上而下设计的系统结构的一致性. 文献[26]使用 AADL 对航空系统进行建模, 并将模型转换至确定性随机 Petri 网进行失效概率分析和安全性评估, 从而验证系统是否满足安全性需求. 本文同样对软件的安全性进行了验证, 不同的是, 本文分析并验证机载软件是否符合安全性需求. 文献[27]使用 PRISM 模型检验器(probabilistic symbolic model checker)验证其设计的模型是否满足性能需求. 与文献[24–27]的验证方法类似, 本文采用将非形式化的设计模型转换为形式化模型的方式进行模型检验.

用于模型检验的约束条件大多提取自需求. 如文献[28]将部分襟缝翼控制单元软件 C 源码手动转换为形式化 PROMELA 语言, 并使用 LTL 公式在模型检验器 SPIN 中进行验证, 从而实现自动错误定位. 文献[29]使用形式化 Fiacre 语言建模飞机起落架系统, 验证了模型在正常模式和故障模式下对应的需求, 并详细介绍了从需求中提取 LTL 公式. 本文采用与文献[28,29]类似的方法, 从需求中手动提取 TCTL 形式化约束进行模型检验. 不同之处在于, 本文侧重于检验模型的功能性需求和安全性需求, 从而保障机载软件模型的正确性和安全性. 文献[30]中, 空客公司进行了大量机载系统模型检验的实验, 证明了模型检验可以有效揭露细微的缺陷. 但由于模型检验是一个迭代的过程, 需要根据反例不断进行修改. 目前的验证工具不足以支持迭代过程, 因此空客公司考虑一种自动化方法管理不同时期的验证版本. 本文采用了分层精化的方法进行建模, 相应的模型检验也是分层进行的, 若模型中存在错误则会进行修改, 一定程度上实现了手动的验证版本管理.

(3) 机载软件适航审定相关工作

文献[20]针对适航标准 DO-178C 和 DO-331 提出了一种精简且高度自动化的基于模型的软件开发流程, 但是没有为验证目标提供具体依据. 与文献[20]相似, 本文面向适航标准提出了机载软件开发和验证方法. 文献[31]对自主无人机进行建模和验证, 提供了模型检验输出、高真实物理环境模型检验和基于仿真的模型比较和改进这 3 方面的适航审查依据. 文献[32]对飞控系统中的襟缝翼控制单元高、低级需求进行规约和模型检验,为 DO-333 标准中关于高级和低级需求的验证目标提供证据. 与文献[31,32]类似, 本文的开发和验证过程的产物为标准中的验证目标提供审定依据. 不同之处在于, 本文的开发和验证过程较为完整, 能够为软件计划过程、高低级需求开发和验证过程、源代码开发过程提供审查依据.

7 总结与展望

本文针对机载软件, 提出了基于 SysML 状态机图子集的分层精化建模和验证方法. 整个方法较为完整地覆盖了软件生命周期中需求分析、设计和编码这 3 个阶段. 首先使用 SysML 状态机图进行分层精化建模, 进行从简要到详细的开发, 可以帮助对复杂系统进行分解, 同时便于在早期进行模型检验. 使用了 SysML 需求追溯矩阵, 可以直观地看出需求和设计模型之间的关系, 便于进行系统管理. 设计并实现了 SysML 向 TA, Stateflow 的模型转换方法, 以便进行模型检验和自动代码生成. 最后, 通过飞行控制系统验证本方法的有效性, 并根据适航标准 DO-331 和 DO-333 中的审定目标对开发和验证过程进行了审查, 从而确保机载软件的安全性、正确性.

目前, UPPAAL 中验证的形式化属性是从需求中手动形式化提取的, 可能会造成需求和 TCTL 性质存在差别. 在未来的工作中, 考虑使用自然语言处理方法来实现需求的自动分析和形式化提取. 另外, 在目前的工作中, 我们对机载软件的动态行为进行了建模和验证, 未来考虑加入软件静态结构的建模和可靠性验证.

References:

[1] Huang ZQ, Xu BF, Kan SL, *et al*. Survey on embedded software safety analysis standards, methods and tools for airborne system. Ruan Jian Xue Bao/Journal of Software, 2014, 25(2): 200−218 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4530.htm [doi: 10.13328/j.cnki.jos.004530]

[2] RTCA.DO-331: Model-based Development and Verification Supplement to DO-178C and DO-278A. 2011.

[3] RTCA.DO-333: Formal Methods Supplement to DO-178C and DO-278A. 2011.

[4] Lenny D. SysML Distilled: A Brief Guide to the Systems Modeling Language. 2013.

[5] Object Management Group. Unified modeling language: Superstructure version 2.0. OMG Document, formal/05-07-04, 2005. http://www.omg.org/spec/UML/2.0/Superstructure/PDF/

[6] McKelvin Jr ML, Jimenez A. Specification and design of electrical flight system architectures with SysML. In: Proc. of the AIAA Infotech at Aerospace Conf. and Exhibit. 2012. [doi: 10.2514/6.2012-2534]

[7] Yang H, Zhan C, Wu H, *et al*. Research on modeling of aircraft-level high-lift system architecture based on SysML. Journal of Physics. Conf. Series, 2021, 1827: 12096. [doi: 10.1088/1742-6596/1827/1/012096]

[8] Kim Y, Gomez M, Goppert J, *et al*. Model checking of a training system using NuSMV for humanoid robot soccer. Cham: Springer Int'l Publishing, 2015. 531−540. [doi: 10.1007/978-3-319-16841-8_48]

[9] Ratiu D, Ulrich A. An integrated environment for Spin-based C code checking: Towards bringing model-driven code checking closer to practitioners. Int'l Journal on Software Tools for Technology Transfer, 2019, 21: 267−286. [doi: 10.1007/s10009-019-00510-w]

[10] Ahn SJ, Hwang DY, Kang M, *et al*. Hierarchical system schedulability analysis framework using UPPAAL. IEICE Trans. on Information and Systems, 2016. 2172−2176.

[11] Miotto P, Breger L, Sargent R. Simulation and flight software development using model-based design with MATLAB and UML tools. In: Proc. of the AIAA Modeling and Simulation Technologies Conf. 2012. 2012.

[12] David A, Larsen KG, Legay A, Mikucionis M, Poulsen DB.Uppaal SMC tutorial. Int'l Journal on Software Tools for Technology Transfer, 2015, 17: 397−415.

[13] Chen X, Gu Q, Liu WS, *et al*. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016, 27(1): 1−25 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[14] Hao JF, Ye H, Ren XR. Research on formal methods of DO-333 supplement. Aeronautical Computing Technique, 2020, 50(1): 124−129 (in Chinese with English abstract).

[15] Meta object facility core specification v2.0. 2006. http://www.omg.org/cgi-bin/apps/doc?formal/06-01-01.pdf

[16] Liu ZZ. The study of consistency issues in model transformations based on graph theory [MS. Thesis]. Hefei: University of Science and Technology of China, 2010 (in Chinese with English astract). [doi: 10.7666/d.y1705349]

[17] van Benthem, Johan FAK, ter Meulen A. Handbook of Logic and Language. Oxford: Elsevier, 2011.

[18] Xu J. Aircraft Automatic Flight Control System. Beijing: Beijing Institute of Technology Press, 2020. 5−30 (in Chinese).

[19] Gao S, Cao W, Fan L, *et al*. MBSE for satellite communication system architecting. IEEE Access, 2019, 7: 164051−164067.

[20] Dmitriev K, Zafar SA, Schmiechen K, *et al*. A lean and highly-automated model-based software development process based on DO-178C/DO-331. In: Proc. of the 39th AIAA/IEEE Digital Avionics Systems Conf. (DASC). IEEE, 2020. 1−10.

[21] Dai SX, Hong M, Guo B, *et al*. Schedulability analysis model for multiprocessor real-time systems using UPPAAL. Ruan Jian Xue Bao/Journal of Software, 2015, 26(2): 279−296 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4781.htm [doi: 10.13328/j.cnki.jos.004781]

[22] Qiao L, Yang MF, Tan YL, *et al*. Formal verification of memory management system in spacecraft using Event-B. Ruan Jian Xue Bao/Journal of Software, 2017, 28(5): 1204−1220 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5218.htm [doi: 10.13328 /j.cnki.jos.005218]

[23] Han D, Yang Q, Xing J, *et al*. EasyModel: A refinement-based modeling and verification approach for self-adaptive software. Journal of Computer Science and Technology, 2020, 35: 1016−1046. [doi: 10.1007/s11390-020-0499-x]

[24] Holzmann GJ. The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, 2003.

[25] Graves H, Bijan Y. Using formal methods with SysML in aerospace design and engineering. Annals of Mathematics and Artificialintelligence, 2011, 63: 53−102. [doi: 10.1007/s10472-011-9267-5]

[26] Wei XM, Dong ZQ, Xiao MR, *et al*. Failure probabilities allocation and safety assessment approaches based on AADL. Ruan Jian Xue Bao/Journal of Software, 2020, 31(6): 1654−1671 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5999.htm [doi: 10.13328/ j.cnki.jos.005999]

[27] Baouya A, Bennouar D, Mohamed OA, *et al*. A probabilistic and timed verification approach of SysML state machine diagram. In: Proc. of the 12th Int'l Symp. on Programming and Systems (ISPS). IEEE, 2015. 1−9. [doi: 10.1109/ISPS.2015. 7245001]

[28] Chen Z, Gu Y, Huang Z, *et al*. Model checking aircraft controller software: A case study. Software, Practice & Experience, 2013, 45: 989−1017. [doi: 10.1002/spe.2242]

[29] Berthomieu B, Dal Zilio S, Fronc Ł. Model-Checking real-time properties of an aircraft landing gear system using fiacre. Cham: Springer Int'l Publishing, 2014. 110−125. [doi: 10.1007/978-3-319-07512-9_8]

[30] Bochot T, Virelizier P, Waeselynck H, *et al*. Model checking flight control systems: The Airbus experience. In: Proc. of the 2009 31st Int'l Conf. on Software Engineering—Companion Volume. IEEE, 2009. 18−27. [doi: 10.1109/ICSE-COMPANION.2009. 5070960]

[31] Webster M, Cameron N, Fisher M, *et al*. Generating certification evidence for autonomous unmanned aircraft using model checking and simulation. Journal of Aerospace Information Systems, 2014, 11: 258−278. [doi: 10.2514/1.I010096]

[32] Chen GY, Huang ZQ, Chen Z, *et al*. Safety analysis of slat and flap control unit for DO-333. Computer Science, 2016, 43(5): 150−156, 161 (in Chinese with English abstract). [doi: 10.11896/j.issn.1002-137X.2016.5.028]

附中文参考文献:

[1] 黄志球, 徐丙凤, 阚双龙, 等. 嵌入式机载软件安全性分析标准、方法及工具研究综述. 软件学报, 2014, 25(2): 200−218. http://www.jos.org.cn/1000-9825/4530.htm [doi: 10.13328/j.cnki.jos.004530]

[13] 陈翔, 顾庆, 刘望舒, 等. 静态软件缺陷预测方法研究. 软件学报, 2016, 27(1): 1−25. http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[14] 郝继锋, 叶宏, 任晓瑞. DO-333 标准形式化方法研究. 航空计算技术, 2020, 50(1): 124−129.

[16] 刘峥峥. 使用图转换理论的模型转换一致性研究 [硕士学位论文]. 合肥: 中国科学技术大学, 2010. [doi: 10.7666/d.y1705349]

[18] 徐军. 飞机自动飞行控制系统. 北京: 北京理工大学出版社, 2020. 5−30.

[21] 代声馨, 洪玫, 郭兵, 等. 多处理器实时系统可调度性分析的 UPPAAL 模型. 软件学报, 2015, 26(2): 279−296. http://www.jos.org.cn/1000-9825/4781.htm [doi: 10.13328/j.cnki.jos.004781]

[22] 乔磊, 杨孟飞, 谭彦亮, 等. 基于 Event-B 的航天器内存管理系统形式化验证. 软件学报, 2017, 28(5): 1204−1220. http://www.jos.org.cn/1000-9825/5218.htm [doi: 10.13328/j.cnki.jos.005218]

[26] 魏晓敏, 董泽乾, 肖明睿, 等. 基于 AADL 的失效概率分配及安全性评估方法. 软件学报, 2020, 31(6): 1654−1671. http://www.jos.org.cn/1000-9825/5999.htm [doi: 10.13328/j.cnki.jos.005999]

[32] 陈光颖, 黄志球, 陈哲, 等. 面向 DO-333 的襟缝翼控制单元安全性分析. 计算机科学, 2016, 43(5): 150−156, 161. [doi: 10.11896/j.issn.1002-137X.2016.5.028]

肖思慧(1997－), 女, 硕士, CCF 学生会员, 主要研究领域为形式化建模与验证.

史建琦(1984－), 男, 博士, 副研究员, 博士生导师, 主要研究领域为工业软件, 可信人工智能, 嵌入式控制系统.

刘琦(1996－), 男, 硕士, CCF 学生会员, 主要研究领域为形式化建模与验证.

郭欣(1992－), 男, 硕士, 主要研究领域为可信软件.

黄滟鸿(1986－), 女, 博士, 副研究员, 主要研究领域为可信计算, 形式化建模与验证, 高可信嵌入式控制软件.