基于 SPIN 的 HMSC 模型自动检验方法

李立亚[1]，孙雨荷[2]，马汉杰[2+]，丁佐华[2]，黄鸿云[3]

（1. 无锡科技职业学院 人工智能学院，江苏 无锡 214000；2. 浙江理工大学 计算机科学与技术学院，浙江 杭州 310018；3. 浙江理工大学 图书馆多媒体大数据中心，浙江 杭州 310018）

摘 要：自动检测与验证 HMSC（high-level message sequence chart）模型的正确性对保证文本需求被正确建模具有十分重要的意义，为此提出一种为 HMSC 模型进行自动检验的方法，并将其实现。利用转换规则为 HMSC 模型生成 Promela 检测语言，借助 SPIN 工具对需求进行验证。该方法不仅支持模型检测，同时通过对系统行为的动态模拟可以实现需求的合理性分析。从 Promela 实现到 SPIN 验证整个过程实现自动化操作。在该方法的基础上实现一个文本需求自动建模及检测分析的工具，通过一个实例展示其自动建模检测分析的效果，表明了其有效性和实用性。

关键词：模型检测；HMSC 模型；SPIN 工具；正确性验证；模型转换；Promela 语言；形式化方法

中图法分类号：TP311 **文献标识号**：A **文章编号**：1000-7024（2023）10-3047-09

doi：10.16208/j.issn1000-7024.2023.10.022

Automatic verification method of HMSC model based on SPIN

LI Li-ya[1], SUN Yu-he[2], MA Han-jie[2+], DING Zuo-hua[2], HUANG Hong-yun[3]

(1. School of Artificial Intelligence, Wuxi Vocational College of Science and Technology, Wuxi 214000, China;
2. School of Computer Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China;
3. Library Multimedia Big Data Center, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: Automatic detection and verification of the correctness of the HMSC (high-level message sequence chart) model is of great significance to ensure that the text requirements are modeled correctly. To this end, a method for automatic verification of HMSC model was proposed and realized. The Promela detection language for the HMSC model was generated using the conversion rules, and the requirements were verified with the help of the SPIN tool. This method not only supports model checking, but realizes rationality analysis of requirements through dynamic simulation of system behavior. The entire process from Promela implementation to SPIN verification is automated. Based on this method, a tool for automatic modeling, detection and analysis of text requirements was implemented, and an example was used to demonstrate the effects of automatic modeling, detection and analysis, which shows the effectiveness and practicability of this method.

Key words: model checking; high-level message sequence chart model; SPIN tool; correctness verification; model transformation; Promela language; formal methods

0 引 言

需求分析是软件开发过程中的重要一环，需求通常以自然语言进行描述，但是由于缺乏形式化的定义，其正确性很难验证。基于文本需求 use case 建立 HMSC（high-level message sequence chart）模型，能够帮助用户跟踪应用程序的控制流，将软件系统的通信行为以非常直观方式进行展示。而对建立的 HMSC 模型进行正确性检测与验证是保证文本需求被正确建模必不可少的工作。在现有的研究中，有许多模型检验的方法，然而它们主要存在的问题是在分析中需要过多人为参与，不能实现自动化。在本文中，我们提出了一种自动检测 HMSC 模型的方法。首先建

收稿日期：2022-12-19；**修订日期**：2023-09-18

基金项目：国家自然科学基金项目（62132014）

作者简介：李立亚（1977－），男，河北石家庄人，硕士，副教授，CCF 高级会员，研究方向为软件工程和软件测试；孙雨荷（1998－），女，安徽蚌埠人，硕士研究生，研究方向为软件工程和软件测试；＋通讯作者：马汉杰（1964－），男，江苏南通人，博士，教授，研究方向为视频图像传输与处理、机器视觉、数据挖掘；丁佐华（1964－），男，江苏南通人，博士，教授，研究方向为需求建模及分析、软件测试、软件可靠性计算；黄鸿云（1977－），女，江苏如皋人，硕士，讲师，研究方向为软件工程。E-mail：hanjie.ma@gmail.com

立了 HMSC 模型到 Promela 检测语言的转换规则，并提出了自动转换算法，从而实现了为 HMSC 模型生成 Promela 代码。然后，利用模型检测工具 SPIN 对 Promela 描述的模型进行正确性验证，同时能够检验 HMSC 模型是否满足文本需求设计。此外，本文实现了一个对文本需求进行自动建模以及验证分析的工具，并通过一个实例（EIRENE）展示了该工具在自动建模及检测方面的效果。总的来说，我们的方法具有以下 3 点优势：①SPIN 工具结合 Promela 模型语言能够完成模型检验，并通过对系统行为的动态模拟可以实现需求的合理性分析；②整个模型验证过程实现了自动化；③我们的建模及检测分析工具整合了 SPIN 的功能，可以验证任意 LTL 公式。

1 相关工作

1.1 HMSC 模型

从需求文本中提取信息并转化为形式化的模型，有利于对需求的验证与管理[1]。针对需求文本 use case 建模的方法和工具有很多。例如，将文本 use case 转换成 UML 中的模型[2]。除了 UML 外，MSC（message sequence chart）也可为需求 use case 建立行为模型[3]。

MSC 是一种受欢迎的软件系统行为图形描述语言[4,5]。单个 MSC 用于描述系统的局部行为，其中，实例（Instance）表示软件系统的各组件、用户或外部环境，以垂直箭头表示。实例间的交互称为消息，用水平箭头从消息的发送者指向接收者。事件（Event）是指消息箭头开始的点（发送事件）或结束的点（接收事件）。

HMSC 将多个简单的 MSC 进行组合关联，以描述系统的整体行为[5,6]。HMSC 利用 choice、sequential 和 loop 操作将多个 MSC 关联到一起，choice 代表选择操作，sequential 表示顺序操作，loop 表示循环操作。在图 1 中，我们给出了一个 HMSC 示例，其中单个 MSC 用长方形方框表示，而 HMSC 通过控制流操作将它们组合在一起。

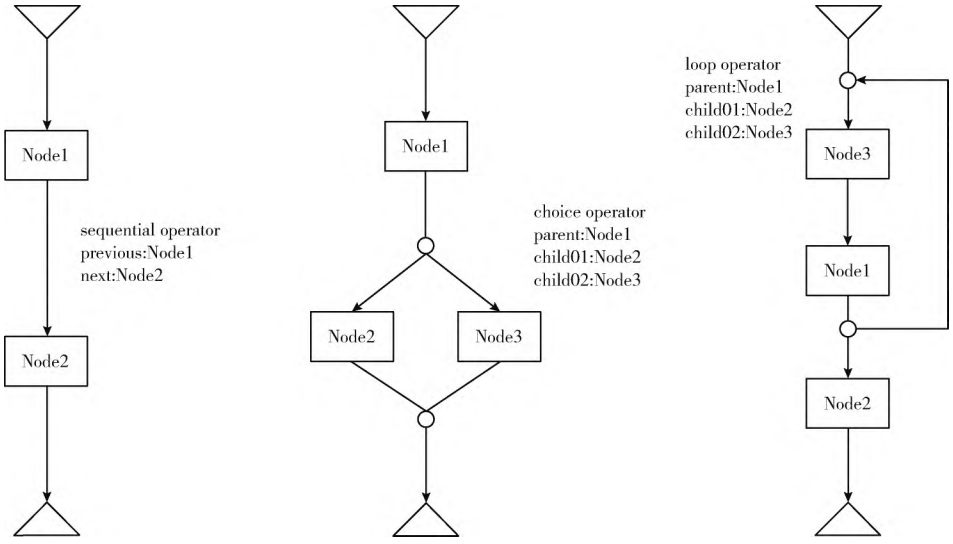

图 1 HMSC 中的 sequential、choice 以及 loop 操作符

1.2 模型正确性检测

模型检测是软件质量保障的一重要环节，从 20 世纪 80 年代开始了模型检测的相关研究[7]。Clarke 等[8]最早提出用 CTL 逻辑来描述并发系统性质。基本思想是将软件系统的某些性质用时序逻辑来描述，并在描述系统的有限状态集合中检查这些性质是否被满足。其中，计算树逻辑（CTL）、有线性时序逻辑（LTL）以及命题 μ—演算等都可以作为模型检测时使用的时序逻辑。同时，也诞生了 SMV[9]、SPIN[10]、CWB[11] 和 UPPAAL[12] 等众多模型检测工具。

在这些工具中，基于 SPIN 的验证在学术界和工业界被广泛使用[13]。由于 SPIN 良好的算法设计和较强的检测能力，本文也使用其作为检测工具。基于 SPIN 进行的模型检测研究也非常多，Ji 等[14] 将 UML 类图、状态图和协作图转换成 verification model，并为其生成 Promela 代码，进而通过 SPIN 完成模型检测；Krook 等[15]通过 SPIN 对汽车自动驾驶系统的控制算法进行验证；Bai 等[16]将 SPIN 用于验证智能合约模板的正确性和必要属性。

2 基于 SPIN 的 HMSC 模型检测方法

2.1 SPIN 检测过程

SPIN 工具通过 never claims、断言（assertion）以及特殊的标记 3 种方式来实现对软件系统的正确性检测，其中正确性包括了不发生死锁情况和不存在活性问题，断言永远为真以及满足自定义的 LTL 公式。

Promela 是一种用来验证并行系统逻辑正确性的进程模

型语言[17]。一个用 Promela 描述的软件行为模型，SPIN 可以通过对其进行随机或者迭代模拟执行来验证模型的正确性，或生成 C 程序对系统的状态空间进行穷举检查。Promela 程序由 processes、message channels 和 variables 组成[18]。其中，processes 定义了系统的行为，而 channels 和 variables 制定了 processes 运行的环境。

本文按照典型的 SPIN 工作模式来完成检测过程。首先，使用 Promela 语言为并发系统构建高层次的行为模型，在确认没有语法错误之后通过 SPIN 的交互模拟功能验证系统的设计行为是否符合预期。然后，SPIN 为其生成一个自定义的 on-the-fly 验证程序并进行编译，进而验证模型中各种正确性断言是否成立（例如 LTL 公式）并给予用户反馈。

2.2 HMSC 模型的 Promela 实现

由于 HMSC 是由多个简单 MSC 组合关联而成的，因此需要为基本 MSC 以及 HMSC 间的选择和循环结构实现 Promela 表示。也就是说，在 HMSC 的 Promela 代码实现过程中，需要解决两个问题：一是对于基本 MSC（不包含 HMSC 中操作符的 MSC，也称 Basic MSC）中描述的有限执行序列如何用 Promela 表示；二是对于 HMSC 中的选择和循环结构如何用 Promela 表示。此外，还需要为 MSC 中的 Instance 选择通信通道以及设定容量。

2.2.1 Basic MSC 的 Promela 实现

为了完成 MSC 的 Promela 代码实现，首先在系统的启动阶段，为每一个 MSC 中的 instance 建立一个 process，并且在 init 代码块中启动所有的 processes。对于不止一个 process 的情况，需要使用 atomic 代码块以保证所有 processes 启动的原子性，例如，设某 MSC 有两个 instances P1 和 P2，在 init 代码块中，需要这样写：atomic{run P1(); run P2();}；在 MSC 和 Promela 中，消息是有类型的，使用 mtype 表示消息的类型。mtype = {a,b} 表示两个一字节整数常量 a 和 b，且 a=1，b=2，即创建了两种消息类型 a 和 b。

在为 MSC 建立基于 Promela 的行为模型时，选择容量为 1 的 channel 作为 processes 中间通信的通道，并且对于 MSC 中的每一个消息箭头，都有一个通信通道与之对应。channel 中的消息类型必须与 MSC 中 event 处理的消息类型相同，在 Promela 中，使用“chan ch = [1] of {byte}”定义一个名称为 ch 容量为 1 接收消息类型为 byte 的消息通道，对应于 MSC 中的消息通道，byte 需要根据实际情况替换成对应通道的消息类型。总结来说，将 basic MSC 转化为 Promela 步骤如下：

步骤 1 必要的数据定义，包括全局 channel 定义等；

步骤 2 通过 proctype 关键字为每一个 instance 独立定义 process，在本文介绍的模型中所有 process 没有需要接受的参数。

步骤 3 在 init 代码块中启动每一个 process。

2.2.2 选择和循环结构的 Promela 实现

Basic MSC 的 Promela 实现后，需要将 HMSC 中的选择或者循环结构用 Promela 代码描述。图 2 是一个包含选择结构的 HMSC。通过前面的方法，我们将 MSC 中的每一个 instance 都转换为了 process。在图 2 中，当 P1 发送完消息 req 之后，P2 可能回复 success 消息也可能回复 failure 消息。

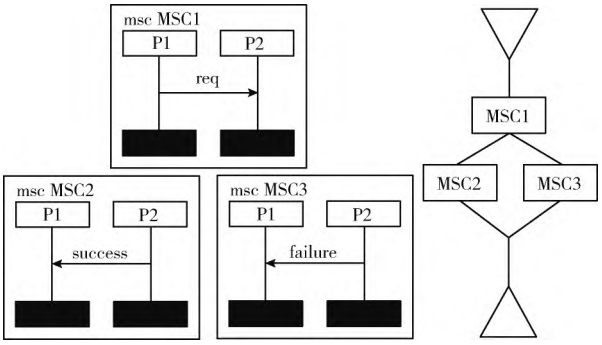

图 2 包含选择结构的 HMSC

这里有 4 个基本的执行指令，分别是：P2 发送 success、P2 发送 failure、P1 接收 success 和 P1 接收 failure。要确保 P2 发送 success 之后，P1 必须要接收该 success 消息。同样，P2 发送 failure 之后，P1 必须要接收该 failure 消息，这里的一个操作涉及到了两个进程。也就是说，用 Promela 实现 HMSC 中的选择或者循环结构时，P1 和 P2 之间需要进行同步。但是 Promela 语言中并没有直接提供同步组件，我们需要自己独立实现。

在图 3 所示的 Promela 中，使用了一个额外的消息通道 sync。在进程 P2 中，选择操作符的实现在 if 代码块中，该 if 代码块中，有两个可执行序列（两个以::开头的代码段）。在第一个执行序列中，向 sync 通道发送 yes 消息，同时执行 ch2！c2，向 ch2 中发送 c2，然后通过 goto 跳转至 SKIP 点；第二个执行序列中，向 sync 通道发送 no 消息，同时向 ch2 发送 c2。根据 Promela 中 if 的语义，将会随机选择一个执行序列执行，这就在 P2 中实现了选择操作，该执行序列称为 signal 代码段。同样的，在 P1 中，我们也使用了一个 if 代码块，其中有两个执行序列。在 Promela 中，如果 if 中所有执行序列的守卫都不满足条件的话，process 将会阻塞，直至一个守卫满足条件为止，在该代码块中，通过判断 sync 中有无消息完成了 P1 和 P2 的同步操作（如果 P1 没有发送消息，P2 将会一直阻塞），通过消息的类型是 yes 还是 no 完成选择操作，该执行序列称为 wait 代码段。对于图 3 代码，如果用 SPIN 进行模拟执行，会发现其输出结果只可能是如下两种：①ch2！ch2；ch2？ch2；②ch3！ch3；ch3？ch3。即要么是 P2 发送 success(ch2！c2) 和 P1 接收

success(ch2?2)，要么是 P2 发送 failure(ch3!c3) 和 P1 接收 failure(ch3?c3)。至此，HMSC 中的选择结构的 Promela 实现已经完成。由于选择操作符和循环操作符的定义非常相似，仅仅是最后包含的顺序操作符跳转的位置不一样（即根据 goto 属性的值判断是选择操作符还是循环操作符），所以在转换为 Promela 描述的时候，循环结构与选择结构也是非常相似的，只需要改变 Promela 中 goto 指令后面的位置即可。

```
mtype={c1,c2,c3,yes,no}
chan ch1 = [1] of {byte};
chan ch2 = [1] of {byte};
chan ch3 = [1] of {byte};
chan sync = [1] of {Byte}
proctype P2(){
   ch1?[c1]->ch1?c1;
   if
      ::atomic{
         sync!yes;
         ch2!c2;
         printf(" ch2!c2\n " );
         goto SKIP;
      }
      ::atomic{
         sync!no;
         ch3!c3;
         printf(" ch3!c3\n " );
      }
   fi
   SKIP:
}
```

```
proctype P1(){
   ch1!c1;
   if
   ::{
      sync?[yes]->{
         atomic{ch2?c2;
         printf(" ch2?c2\n " );}
         goto SKIP;
      }
   }
   ::{
      sync?[no]->{
         atomic{ch3?c3;
         printf(" ch3?c3\n " );}
         goto SKIP;
      }
   }
   fi
   SKIP:
}
init{atomic{ run P1();  run
P2(); }}
```

图 3 选择结构的 HMSC 的 Promela 实现

2.2.3 HMSC 模型到 Promela 代码的自动转换

HMSC 模型到 Promela 代码的转换方法的 UML 类图如图 4 所示。其中，有两个非常重要的类，一是 PromelaGenerator，该类主要提供了对外的使用接口，通过构造函数传入 HMSC 模型，通过 generate() 方法生产 Promela 代码；另一个是 PromelaModel 类，相当于 Promela 语言的模型类，它以组合的方式包含了 Promela 中的大部分特性。例如，Promela 中的变量定义被抽象成了 Variable 类，消息通道的定义被抽象成了 Channel 类等。这些代表语言语法的类通过组合关系被添加进了 PromelaModel 中，我们只需要对 PromelModel 进行操作，就可以自动获得映射好的 Promela 代码。以下是这些实现类中的重点方法。

（1）Process 类

Prcocess 类有 3 个属性，分别是表示该 Process 名称的 name，所有选择或者循环操作符跳转点的集合 gotoPositions 以及该 Process 所对应的 Promela 代码缓存区 sb。

另外，该类有两个关联的类 IfWait 和 IfSignal，这两个类是上文中介绍的在 Promela 中自己实现的同步组件，用以实现选择和循环结构。IfWait 类用于生成 wait 代码段，而 IfSignal 用于生成 signal 代码段。

在 Process 类中还提供了一系列公有方法，如 addSendMessage() 用于向当前的 process 中添加消息发送语句，而消息通道和消息的类型则通过方法参数传递进来；addGotoPoint() 用于添加跳转点；addIfSignal 和 addIfWait 分别用于添加 signal 代码段和 wait 代码段。

（2）Init、Define、Channel、Variable、LTL 和 MType 类

Init 类表示 Promela 中的 init 代码段，用于启动创建的 process，其中方法 addProcess() 可以将一个 Process 添加待启动的列表中；Define 表示 Promela 中的 define 语句；Channel 表示 Promela 中的消息通道定义语句；Variable 表示 Promela 中的变量定义语句；LTL 表示 Promela 中的 LTL 公式，addLTL() 将一个 LTL 公式添加到代码中；MType 类表示 Promela 中的 mtype 语句。

（3）PromelaModel 类

Promela 类组合了 Init、Define、Channel、Variable、LTL、MType 和 Process 类，通过组合它们的功能而构成了整个 Promela 模型。getPromela() 方法可以得到对应的 Promela 代码。

（4）HMSCWrapper 类

HMSCWrapper 是一个 HMSC 的包装类，它主要是为 HMSC 提供一些便捷的操作接口。例如 isJump() 方法判断当前节点是否是一个跳转节点；isLoop() 方法判断当前节点是否是循环结构的起始节点；getNextNode() 方法返回当前节点的下一个节点（顺序操作符的 next 属性指向的节点）；getBranchNode() 方法返回当前节点的分支节点（如选择操作符或循环操作符的 child02 属性）。

（5）HMSCNode 类

HMSCNode 是 HMSC 模型中 Node 类的辅助类，主要用于 Node 的一些判断操作。例如 isHMSC() 方法判断该 Node 是否是 HMSC；getHMSC() 方法返回该 Node 对应的 HMSC。

（6）PromelaGenerator 类

该类主要对外提供操作接口，通过 generate() 方法可以获得生成的 HMSC 的 Promela 实现。

3 工具实现

本节将介绍根据前文介绍的方法，实现的一个文本需求 use case 自动建模及检测分析工具。

3.1 框架介绍

该工具的主界面如图 5 所示，整个界面分为 3 个部分，左上为 HMSC 可视化图形接口，右侧栏为 LTL 公式输入以及 SPIN 分析结果输出接口，而下方为各种操作的接口。

在下方的操作接口中，一共有 4 组，每一组都有一个标签，它们分别是 NLP、Usecase、HMSC 和 ModelChecking，下面分别简要介绍其功能。NLP 标签中的功能实现了自然语句句法解析树的管理，其中预置了支持的语法，也可以自定义进行添加。Usecase 标签中的功能主要实现了文本需求 use case 的自动建模，它以文本需求 use case 为输

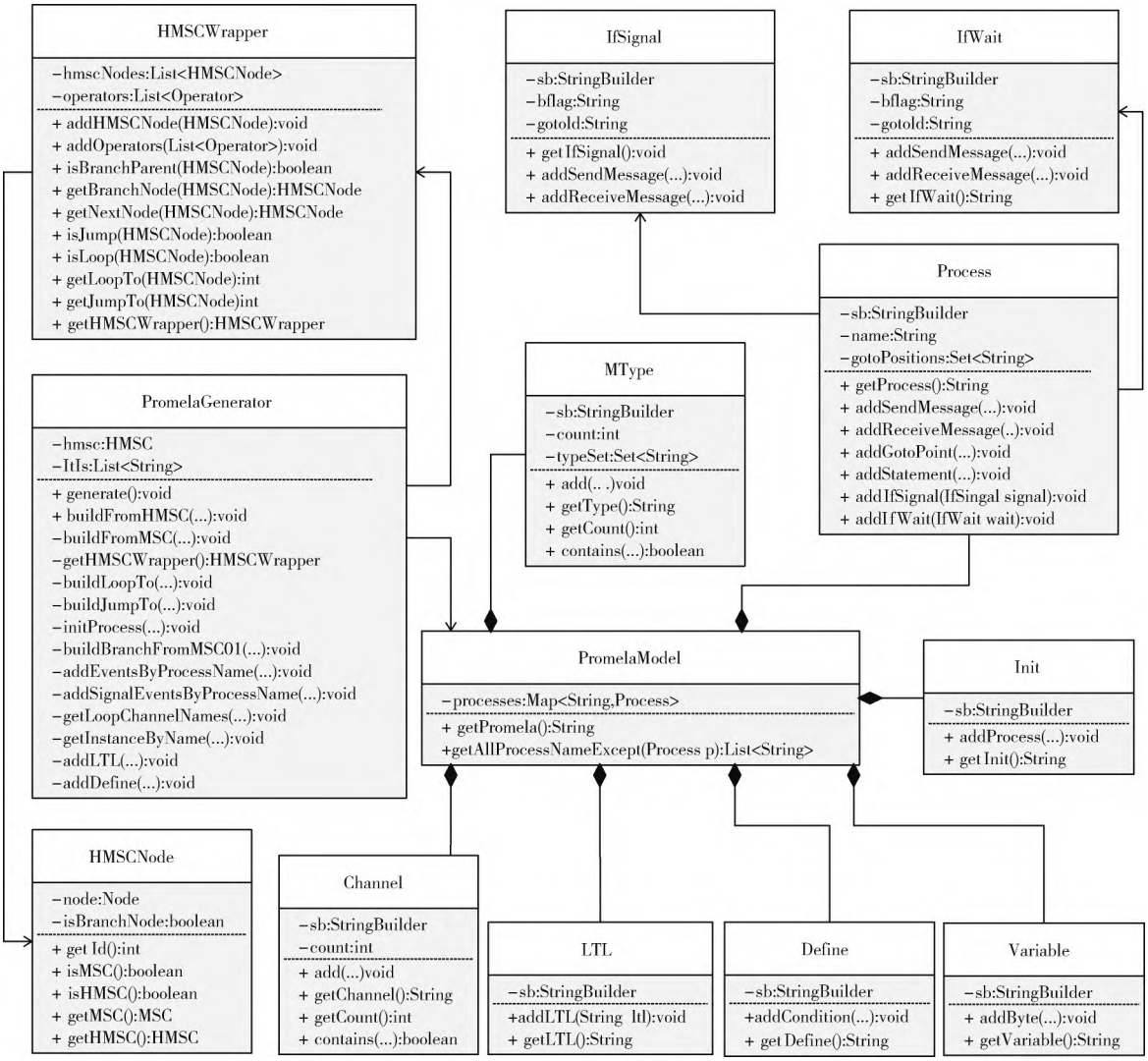

图 4　HMSC 到 Promela 转换方法的 UML 类图

图 5　工具主界面

入，借助自然语言处理工具和预先存入的语法解析树，自动生成 use case 静态模型。HMSC 标签中的功能实现了从 use case 静态模型到 HMSC 动态模型的自动转换，HMSC 除了图形化显示还可以预览其 XMI 文本。Model Checking 标签的功能实现了 HMSC 模型的自动验证，它可以为 HMSC 模型自动生成 Promela 代码，同时整合了 SPIN，完成模型检测功能，在右上的文本框里，还可以输入 LTL 公式进行验证。下面重点介绍 Model Checking 部分的功能。

3.2　模型检测

在 ModelChecking 标签中，有两个功能按钮，分别是 Promela 和 Spin。Promela 用于自动将 HMSC 模型转换为 Promela 模型，Spin 按钮集成了模型检测工具 SPIN 用于模拟执行 Promela，并输出结果，如图 6 所示。而位于主界面右上的 LTL 窗口中，LTL Formula 用于输入合法的 LTL 公式，Check 用于检测该 Promela 模型是否满足该公式，其验证结果将在 Output 文本显示框中显示，如图 7 所示。

3.3　应用场景

利用这个工具可以对多线程软件进行有效的检测。比如检测一个网上购物商城系统，该系统由许多业务子系统

图 6 Promela 自动生成窗口

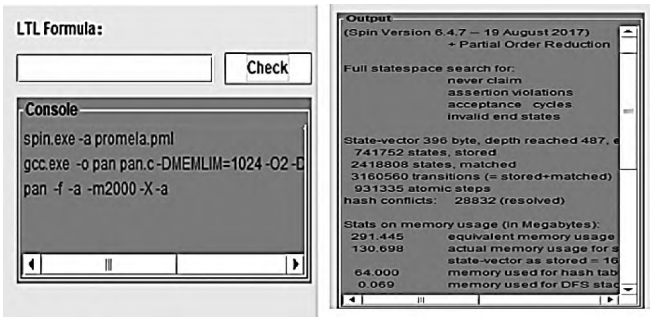

图 7 输入及模型检测结果输出窗口

组合而成，各业务子系统之间通过通用接口进行交互，完成一次特定的功能。此时，对于用户的一次购物操作，其涉及的子系统包括商品主数据系统、商品分类系统、购物车系统、用户登录系统、订单中间件和支付系统。我们可以将以上过程涉及到的任意 use case 输入到工具中，为其自动建立 HMSC 模型，并进行验证。如果模型检测结果输出窗口中出现错误信息提示，则根据具体信息进行判断，从而对文本需求 use case 进行修改，以在软件开发的需求工程阶段找到并解决错误。此外，该工具也可以检测协议、验证算法的正确性和线性时态逻辑的正确性，因此，具有较为广泛的使用场景。

4 实例分析

我们借助一个实例验证本文提出的方法。该实例来自 EIRENE（european integrated railway radio enhanced network）的功能需求规范。EIRENE 是一种铁路通信网络，其功能需求规范则定义了一个满足欧洲铁路无线通信需求的无线电系统，它包括地面列车的语音和数据通信，以及地面轨道工人、车站、仓库工作人员和铁路管理人员之间的移动通信。

Cab Radio 是这个功能需求规范中描述的一部分，包括 driver call 功能、与 driver 相关的其它功能、与 cab radio 相关的其它功能、环境需求和物理需求、driver 的人机接口、driver 的安全设备接口、列车车载记录器、控制/命令接口以及其它相关接口。我们以 driver call 功能相关的 Send Railway Emergency Call（发送铁路紧急呼叫）为例，这一功能涉及 3 个子系统：cab radio，MMI（man-machine inteface）和 driver，图 8 给出了它们之间的关系。该用例的场景如下：为了进行铁路紧急呼叫，driver 需要通过 MMI 设备发送（接收）信息，首先 driver 通过 MMI 初始化一个呼叫，呼叫信息会传递给 cab radio，同时该信息也会发送到与 cab radio 连接的列车车载记录器，cab radio 在接收到呼叫请求后将会开始进行连接，当 driver 接收到连接信息后将会向 MMI 发送一个消息，并且当 driver 拿起话筒时，呼叫连接将会接通到 LoudSpeaker。最后，当呼叫结束时，driver 将会终止至此呼叫，当 cab radio 接收到终止信号时，将会发送一条指示消息，这条消息将会通过 MMI 发送给 driver。

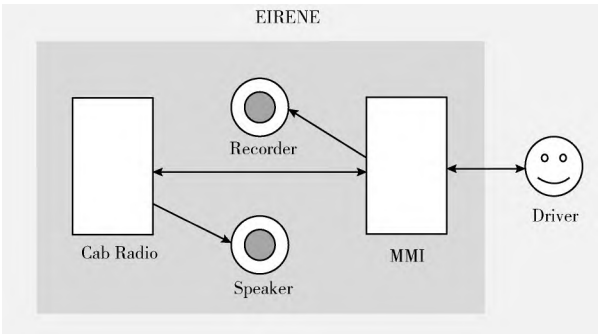

图 8 Send Railway Emergency Call 关系

我们使用 3 个 use case 描述这个场景，如图 9 所示。接下来，利用上文介绍的工具，并按要求以这些用例作为输入，让工具自动生成 HMSC 模型，并进行检测。其结果如图 10 所示。

在以上实例中，HMSC 图形中的每一个节点都是一个 MSC，我们可以通过点击每个节点以查看其具体内容。而在右侧的输出窗口中，可以发现 SPIN 的模型验证输出结果。根据 SPIN 输出结果分析，可知该模型不存在 invalid end states，即没有死锁等非法状态。同时，也可以验证 LTL 公式是否满足需求，例如对于公式“[] <> mmi_10 == 0”表示，mmi 没有收到 checks connection 请求（Use case 2 中的“6 System checks connection”）。将该 LTL 输入后对其进行验证，其输出结果如图 11 所示。根据 LTL 输

Use Case 1: Driver sends a railway emergency call Scope: ERTMS/ETCS Sud: EIRENE Primary Actor: Driver preCondition: null postCondition: null Main Scenario: 1 Driver initiates a call of railway emergency 2 System confirms the call to the Driver 3 Driver sends an indication 4 System confirms the indication to the Driver 5 System connects the call 6 System checks the connection 7 System provides an indication to the Driver 8 Driver confirms the connection 9 System connects the Loudspeaker to send an audible indication 10 System connects the Handset 11 Driver terminates the call 12 System provides termination information to the Driver Extension. 3a A train-borne Recorder is connected to the CR 3a1 Driver sends the message to the Recorder 9a Driver does not pick up the Handset 9a1 Go to step 9	Use case 2: MMI sends a railway emergency call Scope: EIRENE Sud: CR Primary Actor: MMI preCondition: Driver to MMI postCondition: null Main Scenario: 1 MMI sends an emergency call to CR 2 System confirms the emergency to MMI 3 MMI sends visual information to the CR 4 System confirms the information to MMI 5 System connects the call 6 System checks the connection 7 System sends a flag to the MMI 8 MMI confirms the flag 9 System connects the Loudspeaker to send an audible indication 10 System connects the Handset 11 MMI sends the terminate signal 12 System sends termination information to the MMI Extension. 3a A train-borne Recorder is connected to the CR 3a1 MMI sends the data to the Recorder 9a Driver does not pick up the Handset 9a1 Go to step 9 Variations. 6b The call is not connected 6b1 Use case aborted
Use case 3: Driver to MMI Scope: EIRENE Sud: MMI Primary Actor: Driver preCondition: null postCondition: MMI sends a railway emergency call Main Scenario: 1 Driver initiates a call of railway emergency 2 System confirms the call to the Driver 3 System sends an emergency call to CR 4 CR confirms the emergency 5 Driver sends an indication 6 System confirms the indication to the Driver 7 System sends visual information to the CR 8 CR confirms the information	9 CR sends a flag to the MMI 10 System confirms the flag to CR 11 System provides an indication to the Driver 12 Driver confirms the connection 13 Driver terminates the call 14 System sends the terminate signal to the CR 15 CR sends termination information to the MMI 16 System provides termination information to the Driver Extension. 5a A train-borne Recoder is connected to the CR 5a1 Driver sends the message to the Recorder 7a A train-borne Recorder is connected to the CR 7a1 MMI sends the message to the Recorder

图 9 Send Railway Emergency Call **的 3 个用例**

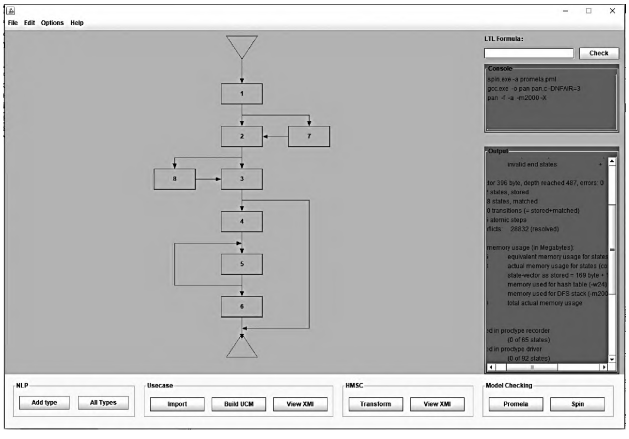

图 10 实例生成的 HMSC

出结果含义，可知该次模型验证过程启用了 never claim，**即输入的** LTL **公式。同时该次执行检测到了一个错误，该错误表示该** LTL **公式不能得到满足，即该模型不满足此** LTL **公式表达的性质。**

```
Full statespace search for:
        never claim             + (ltl_0)
        assertion violations    + (if within scope of claim)
        acceptance  cycles      + (fairness enabled)
        invalid end states      - (disabled by never claim)

State-vector 404 byte, depth reached 171, errors: 1
```

图 11 LTL 输出结果

同时，该工具可以得到自动生成的 HMSC **的** Promela **代码实现。首先点击“**Promela**”按钮然后点击“**Generate**”按钮，其自动生成的部分** Promela **代码如图** 12 **所示，其中所生成的** Promela **代码一共有** 824 **行之多。借助** SPIN **的可视化工具** ispin **还可以对该** Promela **代码进行执行模拟，观测其行为轨迹。图** 13 **显示了该** Promela **代码某次执行的轨迹图。**

5 结束语

本文针对由文本需求 use case **建立的** HMSC **模型的正确性检测与验证提出了一种方法。提出从** HMSC **模型到**

```
proctype recorder(){
  G1:
  if
  ::{
    loop_driver_0!yes;
    loop_loudspeaker_0!yes;
    loop_mmi_0!yes;
    loop_handset_0!yes;
    loop_cr_0!yes;
    atomic{
      bc5?[bm5]-> bc5?bm5;
      recorder_5 = 1;
      printf( " ?bm5\n " );
    }
    goto G2
  }
  ::{
    loop_driver_0!no;
    loop_loudspeaker_0!no;
    loop_mmi_0!no;
    loop_handset_0!no;
    loop_cr_0!no;
  }
  fi;
  G2:
  if
  ::{
    loop_driver_1!yes;
    loop_loudspeaker_1!yes;
    loop_mmi_1!yes;
    loop_handset_1!yes;
    loop_cr_1!yes;
    atomic{
      bc7?[bm7]-> bc7?bm7;
      recorder_7 = 1;
      printf( " ?bm7\n " );
    }
    goto G3
  }
  ::{
    loop_driver_1!no;
    loop_loudspeaker_1!no;
    loop_mmi_1!no;
    loop_handset_1!no;
    loop_cr_1!no;
  }
  fi;
  G3:
  atomic{
    if
    :: atomic{
      bflag22=1;
      goto END;
    }
    :: atomic{
      bflag22=2;
    }
    fi
  }
  G4:
  G5:
  if
  ::{
    loop_driver_2!yes;
    loop_loudspeaker_2!yes;
    loop_mmi_2!yes;
    loop_handset_2!yes;
    loop_cr_2!yes;
    goto G5
  }
  ::{
    loop_driver_2!no;
    loop_loudspeaker_2!no;
    loop_mmi_2!no;
    loop_handset_2!no;
    loop_cr_2!no;
  }
  fi;
  G6:
  END;
  skip;
}
```

图 12 工具生成的 Promela 代码

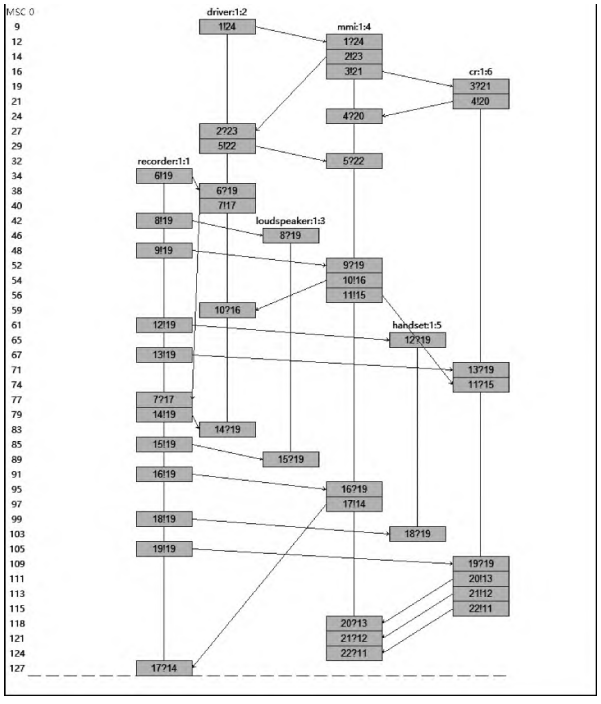

图 13 实例的 Promela 代码某次执行轨迹

Promela 模型检测语言的转换规则，以及其实现算法，进而将 HMSC 模型转换为 Promela 代码并进行模型检查，并借助 SPIN 完成需求的正确性验证。同时基于以上方法实现了一个文本需求 use case 自动建模及检测分析工具。我们的工作为 HMSC 模型的验证提供了完整可行的方法和工具。整个过程都是自动的，不需要过多的人为参与，缩短了建模和分析的时间，大大提升了软件开发效率。

本文提出的这些方法以及其工具实现能很好完成 HMSC 模型的检测。但仍然还有一部分工作，需要更进一步的研究与完善。其中模型检测的结果不能自动反馈至文本需求 use case 中，而需要人参与分析。后续我们还会围绕这些方面开展进一步的研究工作。

参考文献：

[1] LI Wanqian，HU Jun，CHEN Song，et al. A security analysis and verification method for SysML models [J]. Computer Science，2019，46 (11)：100-108 (in Chinese). [李宛倩，胡军，陈松，等. 面向 SysML 模型的安全性分析与验证方法 [J]. 计算机科学，2019，46 (11)：100-108.]

[2] Hamza Z A，Hammad M. Generating UML use case models from software requirements using natural language processing [C] //8th International Conference on Modeling Simulation and Applied Optimization. IEEE，2019：1-6.

[3] Al-qtiemat E M，Srinivasan S K，Dubasi M A L，et al. A methodology for synthesizing formal specification models from requirements for refinement-based object code verification [C] //The Third International Conference on Cyber-Technologies and Cyber-Systems. IARIA，2018：94-101.

[4] Hingmire S，Ramrakhiyani N，Singh A K，et al. Extracting message sequence charts from hindi narrative text [C] //Proceedings of the First Joint Workshop on Narrative Understanding，Storylines，and Events，2020：87-96.

[5] Chai M，Wang H，Tang T，et al. Runtime verification of train control systems with parameterized modal live sequence charts [J]. Journal of Systems and Software，2021，177：110962.

[6] Stutz F，Zufferey D. Comparing channel restrictions of communicating state machines，high-level message sequence charts，and multiparty session types [J]. Electronic Proceedings in Theoretical Computer Science，2022，370：194-212.

[7] Clarke E M，Henzinger T A，Veith H. Introduction to model checking [M] //Handbook of Model Checking. Springer，2018：1-26.

[8] Penczek W. Branching time and partial order in temporal logics [M] //Time and Logic. Routledge，2019：179-228.

[9] Li Y，Yin X，Wang Z，et al. A survey on network verification and testing with formal methods：Approaches and challenges [J]. IEEE Communications Surveys & Tutorials，2018，21 (1)：940-969.

[10] Ferrari A，Mazzanti F，Basile D，et al. Comparing formal tools for system design：A judgment study [C] //Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering，2020：62-74.

[11] Garavel H，Lang F. Equivalence checking 40 years after：A review of bisimulation tools [C] //A Journey from Process Algebra via Timed Automata to Model Learning，2022：213-265.

[12] Larsen K G，Lorber F，Nielsen B. 20 years of UPPAAL ena-

bled industrial model-based validation and beyond [C] //International Symposium on Leveraging Applications of Formal Methods. Springer, 2018: 212-229.

[13] Singh A, Parizi R M, Zhang Q, et al. Blockchain smart contracts formalization: Approaches and challenges to address vulnerabilities [J]. Computers & Security, 2020, 88: 101654.

[14] Gabmeyer S, Kaufmann P, Seidl M, et al. A feature-based classification of formal verification techniques for software models [J]. Software & Systems Modeling, 2019, 18 (1): 473-498.

[15] Krook J, Svensson L, Li Y, et al. Design and formal verification of a safe stop supervisor for an automated vehicle [C] //International Conference on Robotics and Automation. IEEE, 2019: 5607-5613.

[16] Bai X, Cheng Z, Duan Z, et al. Formal modeling and verification of smart contracts [C] //Proceedings of the 7th International Conference on Software and Computer Applications, 2018: 322-326.

[17] Yacoub A, Hamri M E A, Frydman C. DEv-PROMELA: Modeling, verification, and validation of a video game by combining model-checking and simulation [J]. Simulation, 2020, 96 (11): 881-910.

[18] Lucia A D, Deufemia V, Gravino C, et al. Detecting the behavior of design patterns through model checking and dynamic analysis [J]. ACM Transactions on Software Engineering and Methodology, 2018, 26 (4): 1-41.