

KTH Informations- och
kommunikationsteknik

基于组件的建模与仿真验证框架

“将各个部分组合起来”

Imran Mahmood

电子与计算机系统博士论文，斯德哥尔摩，瑞典，
2013

ISBN 978-91-7501-628-3 TRITA-ICT/ECS
AVH 13:01 ISSN 1653-6363 ISRN
KTH/ICT/ECS/AVH-13/01-SE

KTH 信息与通信技术学院,
SE-164 40，基斯塔，瑞典

经瑞典皇家理工学院许可，提交学术论文供公开评审，以取得技术博士学位。评审将于2013年2月26日星期二14:00在基斯塔 Isafjordsgatan 39号的 Forum E厅举行。

© Imran Mahmood，2013年2月。

印刷：Universitetservice US AB

致谢 Acknowledgement

我谨将这份手稿献给我所爱的人，包括几位贵宾、最近离世的父母、将我抚养成人、让我得以见证今日的监护人 Sajid Latif 先生和夫人，以及最重要的——我挚爱的妻子和年幼的女儿。他们忍受分离和我长期缺席所作出的牺牲，任何东西都无法弥补。

我谨向导师 Rassul Ayani 教授致以最诚挚的谢意，感谢他的悉心指导与支持。他不只是为我指明方向，更像一位真正的向导，亲自牵起我的手，带我抵达目的地。能在他的指导下学习和工作，我深感荣幸。我也感谢 Vladimir Vlassov 副教授，他为我的研究提出了中肯的建议，并作出了宝贵贡献。我谨向知识殿堂——瑞典皇家理工学院，尤其是信息与通信技术学院，致以深情的敬意。

感谢瑞典国防研究局（FOI）的 Farshad Moradi 博士给予我持续的支持与鼓励。我也感谢对手 Gary Tan 博士以及评审委员会成员 Oliver Dale 博士提出的建设性批评。

我非常感谢巴基斯坦高等教育委员会为我的学业提供全部经费支持。我感谢 Mumtaz Begum 女士给予我的支持。特别感谢 Awais Ali Sohrawardi 先生和 B. Muhammad 博士在我学习期间给予我的精神支持。我感谢所有同事和朋友，尤其是板球队，让我度过了美好时光。

最后，感谢瑞典的盛情款待、关怀以及留下的美好回忆。

Imran Mahmood
2013年1月，斯德哥尔摩

摘要 Abstract

基于组件的建模与仿真学科能够带来诸多有前景的收益，包括降低开发成本、时间和系统复杂性。这一范式极具价值，因为它推动了模块化组件的使用与复用，并有利于高效开发复杂仿真。然而，在实际践行这一方法论时，它也面临一系列研究挑战。其中一个重要问题是可组合性验证。在建模与仿真（M&S）中，可组合性是指选择并以各种组合方式组装组件，以满足特定用户需求的能力。 因此，为确保组合模型的正确性，需要根据其需求规格说明对其进行验证。

目前存在多种支持可组合性的方法和组件建模框架。据我们观察，大多数组件建模框架对可组合性验证的内置支持不足，甚至完全没有支持。基础对象模型（BOM）就是这样一种框架，它在模型可组合性和复用方面具有良好的潜力。然而，它缺少必要的语义、建模特征和内置评估技术，而这些对于建模复杂系统行为以及推理不同层次上的可组合性有效性至关重要。

本文提出了一个全面的验证框架，以应对可组合性验证中的一些重要问题，并给出了一套验证过程，用于验证不同类型系统模型的可组合性，例如反应式、实时和概率系统。假设所有这些系统在本质上都是并发的，即不同的组合组件会同时相互交互，那么对结构分析和行为分析所需的广泛技术提出的要求也日益严峻。所提出的验证框架为不同层次的可组合性验证提供了方法、技术和工具支持。这些层次被定义为模型一致可组合性的基础。本文详细讨论了每个层次，并提出了验证该层次可组合性的方法。我们特别关注动态语义可组合性层，因为它对整体可组合性正确性至关重要，而且在验证过程中也具有较高难度。为验证这一层次的可组合性，我们研究了三种不同方法的应用，分别是：（i）基于Petri网的代数分析；（ii）基于有色Petri网（CPN）的状态空间分析；（iii）基于通信顺序进程的模型检验。这三种方法从不同角度解决动态语义可组合性的验证问题，但目标相同，即确认组合模型是否符合其需求规范。除了在框架中对这些方法进行实际集成外，我们还改进了每种方法，使其能够有效应用于可组合性验证。例如，应用算法实现Petri网代数计算的自动化，在基于CPN的状态空间分析中引入 状态空间缩减技术，并引入函数库以执行验证任务，帮助建模者更轻松地完成可组合性验证。我们还通过详细示例，展示如何将每种方法用于不同模型，以说明验证过程及其功能。最后，我们对这些方法进行了比较，并提出了相关指南，以便

根据模型的性质和可用信息选择合适的方法。选择正确的方法并遵循我们提出的基于组件的建模与仿真生命周期指南，建模者便可轻松构建基于 BOM 的组合模型，并根据需求规范对其进行验证。

关键词： 建模与仿真，基于构件的开发，可组合性，语义可组合性，动态语义可组合性，验证，正确性，佩特里网分析，代数技术，有色*Petri*网，状态空间分析，通信顺序进程，模型检验。

目录 Table of Contents

致谢.3 摘要.4

目录.6
图目录.9 表格目录.11

缩略语表.13 第1章.16 引言.16

1.1 背景与开篇视角.16 1.1.1 基于组件的软件工程.17 1.1.2 基于组件的建模与仿真.18 1.1.3 使用 Petri网进行建模与分析.19 1.1.4 使用进程代数进行建模与分析.19 1.1.5 模型验证.19

1.2 开篇视角概述.20 1.3 预备知识.20

1.3.1 定义1：组件集.20 1.3.2 定义2：需求规范.20 1.3.3 定义3：组合与可组合性模式.21 1.3.4 定义4：可满足性算子.21 1.4 问题陈述.21

1.5 方法.22
1.5.1 问题域.22
1.5.2 解决方案域.22
1.5.3 解决方案陈述.24
1.6 论文范围.24
1.6.1 正确性.24
1.6.2 验证.24
1.6.3 涌现.25 1.6.4 泛化.25

1.7 贡献总结.25 1.8 论文结构.26

第2章.28
基于组件的建模与仿真.28
2.1 建模与仿真中的可组合性.28 2.2 可组合性简史及相关工作.29 2.2.1 起始阶段.29

2.2.2 理论演进.29
2.2.3 标准与框架.29
2.2.4 技术进展.29 2.2.5 可组合性验证与确认.30 2.3 可组合性理论.30 2.4 与可组合性相关的概念.32 2.4.1 可组合性与可重用性.32 2.4.2 可组合性与互操作性.33 2.5 可组合性层级.34 2.5.1 语法层面：.35 2.5.2 静态语义层面：.35

2.5.3 动态语义层：.35 2.5.4 语用层：.36 2.6 可组合性框架.36 2.6.1 离散事件系统规范（DEVS）.37 2.7 基础对象模型（BOM）框架.38 2.7.1 BOM结构.39

2.7.2 BOM组装.41

2.7.3 模型映射和对象模型定义.42 2.7.4 BOM的组合的形式化规范.42 2.7.5 总结.45

第3章.4 6
可执行建模形式.4
63.1 Petri网.46
3.1.1 PN定义与概念.47 3.1.2 Petri网图.47 3.1.3 PN的性质.49 3.1.4 PN分析.51 3.1.5 PN类别.58

3.2 通信顺序进程.62 3.2.1 基本概念与定义.62 3.2.2 CSP 分析技术.64 3.2.3 时序逻辑.65 3.2.4 时间CSP.66

3.2.5 概率系统.66 3.2.6 CSP实现工具.67 3.2.7 流程分析工具包（PAT）.67

3.3 总结.68
第4章.69 验证与分析.69

4.1 建模与仿真中的一些基本概念.70 4.1.1 建模过程中的验证与确认.71 4.2 自顶向下细化的原理.73 4.3 验证技术.74 4.3.1 非正式技术.74 4.3.2 静态分析：.75 4.3.3 动态分析：.76 4.3.4 形式化分析.76

4.4 总结.77
第5章.7 9
拟议方法与验证框架.7
95 79 5 1 2 . 基于组件的建模与仿真生命周期.初始阶段.80 53 . 建模.81 54 82 . 执行.

5.5 分析.82 5.6 可组合性验证框架.83 5.6.1 发现、匹配与组合（DMC）.83 5.6.2 结构与行为评估.84

5.6.3 静态分析.84 5.6.4 动态分析.90 5.7 PN代数技术.93 5.7.1 BOM到PNML的转换.93

5.7.2 PN代数计算.93 5.7.3 性质验证方法.95 5.8 基于CPN的状态空间分析技术.96 5.8.1 BOM扩展.97

5.8.2 E-BOM到CPN组件的转换.99 5.8.3 组合CPN模型的验证.105 5.9 基于 CSP 的模型检验技术.110 5.9.1 BOM扩展.110

5.9.2 E-BOM 到 CSP# 转换.112
5 113 5 93 10 115 . . 组合CPN模型的验证.总结.**第6章.116 可组合性验证过程.116**

6.1 可组合性验证过程.116 6.1.1 仿真对象、需求和概念模型的构建.126

6.1.2 句法匹配过程.127 6.1.3 静态语义匹配过程.127 6.1.4 状态机匹配过程.127 6.1.5 动态语义可组合性验证的方法选择.127 6.1.6 PN代数技术.128 6.1.7 状态空间分析技术.128 6.1.8 模型检验.128

6.2 总结.129
第7章.13 **0**
使用PN代数技术进行公平性验证.13
07.1 公平性.130
7.2 公平性验证.131
7.3 制造系统.132 7.3.1 场景 I.132

7.3.2 场景 II.138
7.4 总结.141 **第8章.142**

使用状态空间分析技术进行模型验证.14
28.1 作战建模.142 8 11 142 . . 情境环境.

8.1.2 移动.142 8.1.3 观察（或感知）.143 8.1.4 射击.143 8.1.5 通信：.143

8.2 野战炮兵.144
8.2.1 仿真对象.145
8.2.2 野战炮兵模型.145 8.2.3 需求规范.148

8.3 使用 CPN 状态空间分析验证野战炮兵模型.149
8.3.1 静态与动态分析.149 8.3.2 BOM到E-BOM扩展.149

8.3.3 E-BOM 到 CPN 转换.153
8.3.4 CPN组件的组合.158 8.3.5 状态空间分析.159 8.4 状态空间缩减.162 8.5 总结.164

第9章.16 **5**
使用基于CSP的模型检验技术进行模型验证.16 **5**
9.1 野战炮兵场景.165 9.2 需求规范.168 9.3 使用模型检验进行验证.169 9.3.1 静态与动态分析.169 9.3.2 BOM到E-BOM扩展.169

9.3.3 E-BOM 到 CSP# 转换.171
9.3.4 野战炮兵模型的模型检验.174
9.4 总结.176 **第10章.177**

总结与结论.177
10.1 方法选择指南.181 10.1.1 PN 代数技术.181 10.1.2 基于CPN的状态空间分析技术.182 10.1.3 基于CSP的模型检验技术.182 10.2 论文贡献.183

10.3 结论.185
10.4 未来方向.186
参考文献.187

图目录 Figures

图1：作为可计算函数的模型（取自 [34]）.30 图2：执行序列（取自 [50]）.31 图3：组合模型（取自 [50]）.31 图4：通用组件设计与特定组件设计.32 图5：黑盒、玻璃盒、白盒.33 图6 语法可组合性与语义可组合性（取自 [38]）.34 图7：Ping-Pong DEVS [Wikipedia] .38 图8：BOM结构.39 图9：BOM组装.41 图10：（a）BOM Works中的PingPong BOM.41 图11：组合BOM.44 图12：变迁触发序列（取自 [68]）.49 图13：Petri网分析技术.51 图14：生产者-消费者示例.53 图15：M_0 到 M_3 通过发射序列 σ = t2、t1、t2.54 图16：一年中的季节（取自 [68]）.54 图17：（a）Petri网模型（b）可达图（取自 [68]）.56 图18：生产者-消费者Petri网模型及其覆盖图.56 图19：CPN模型.59 图20：分层彩色Petri网.60 图21：建模过程（取自 [108]）.71 图22：建模过程（取自 [29]）.72 图23：仿真研究生命周期（取自 [28]）.73 图24：验证技术.74 图25：CBM&S 生命周期.79 图26：使用UML图表示的仿真对象.80 图27：实现与仿真.83 图28：发现、匹配与组合（DMC）.84 图29：语法匹配.85 图30：数据类型本体的部分子类.88 图31：语义匹配技术.88 图32：静态语义匹配.90 图33：SCXML格式.91 图34：状态机匹配过程.92 图35：BOM到PN的转换.93 图36：PN代数技术.96 图37：缓冲区扩展有限状态机 [120] .97 图38：BOM与E-BOM比较.99 图39：队列组件的CPN-CM表示.103 图40：CPN状态空间分析.106 图41：状态空间分析技术.108 图42：基于CSP的模型检验技术.115 图43：Simuland构造.117 图44：句法匹配过程.118 图45：静态语义匹配过程.119 图46：状态机匹配过程.120 图47：方法选择 | PN代数技术.121 图48：PN代数技术（续）.122 图49：实现.122 图50：状态空间分析技术.123 图51：状态空间分析技术（续）.124 图52：模型检验.125 图53：模型检验（续）.126 图54：制造系统（取自 [124]） .132 图 55：基于制造系统 BOM 的组合模型 .134 图 56：制造系统的状态机匹配 .136 图57：制造系统的Petri网模型.136

图58：修改后的制造系统组合BOM.139 图59：修改后的制造系统Petri网模型.140 图60：作战建模的活动.144 图61：野战炮兵与间接射击的要素.145 图62：野战炮兵组合物料清单.148 图63：野战炮兵模型的状态机匹配.149 图64：观察器 CPN 组件.154 图65：Field CPN组件.155 图66：BHQ CPN组件.156 图67：电池 CPN 组件.157 图68：FDC CPN组件.158 图69：野战炮兵CPN组合模型.159 图70：野战炮兵CPN模型的状态空间.160 图71：野战炮兵模型的约简状态空间图.163 图72：野战炮兵组合模型.168 图73：野战炮兵模型的状态机匹配.169 图74：野战炮兵模型的全局代码块.171 图75：观察者组件的CSP表示.172 图76：BHQ组件的CSP表示.172 图77：电池组件的CSP表示.173 图78：战场组件的CSP表示.173 图79：野战炮兵组合模型.174 图80：野战炮兵验证断言.174 图81：断言1的验证结果.175 图82：断言2的验证结果.175 图83：带有目标时间的野战炮兵验证断言.175 图84：断言3的验证结果.176

表格目录 Table of Tables

表1：实体 A.43 表2：实体 B.44 表3：组合BOM.44 表4：关联矩阵 A.53

表5：状态方程.53 表6：非形式化验证技术.75 表 7：静态分析技术.75 表8：动态分析技术.76 表9：形式化分析技术.77 表 10：可组合性验证中的强制约束.81 表11：语义匹配算法.89 表12：状态机匹配算法.91 表13：关联矩阵计算.94

表14：库所不变量.95
表15：转换规则.102
表16：组合状态空间生成算法.109 表17：E-BOM 中的时间函数.111

表18：概率分布函数.111 表19：E-BOM 到 CSP# 转换规则.113

表20：PAT 断言示例.114 表21：Machine1 基础 BOM 的形式化定义.133

表22：Machine2 基础 BOM 的形式化定义.133
表23：Robot 基础 BOM 的形式化定义.134
表24：制造系统组合BOM的形式化定义.134 表25：语法匹配.135 表26：静态语义匹配.135 表27：初始标识和关联矩阵（场景I）.137 表28：P-不变量和T-不变量（场景I）.137 表29：B-公平性验证.138

表30：Controller 基础 BOM 的形式化定义.139
表31：制造系统组合BOM.139 表32：初始标识和关联矩阵（场景 II）.140 表33：P-不变量和T-不变量（场景 II）.140 表34：观察员基础物料清单.146

表35：野战基础物料清单.146
表36：BHQ 基础物料清单.147 表37：FDC 基础物料清单.147

表38：炮兵连（1、2、3）基础物料清单.147 表39：野战炮兵组合物料清单.147 表40：观察员工程物料清单.150

表41：现场电子物料清单.151
表42：BHQ电子物料清单.152 表43：FDC E-BOM.152

表44：炮兵连工程物料清单.153 表45：约简统计.163

表46：观察员基础物料清单.166
表47：野战基础物料清单.166
表48：BHQ 基础物料清单.167 表49：炮兵连（1、2、3）基础物料清单.167 表50：野战炮兵组合物料清单.167 表51：观察员工程物料清单.169

表52：现场电子物料清单.170
表53：BHQ电子物料清单.170 表 54：BHQ电子物料清单.171 表 55：可验证的属性类型.178 表 56：可验证的模型类型.179 表57：可扩展性.179

表 58：无限模型验证.179
表59：可用性.180 表 60：自动化.180

缩略语表 Acronyms

ABV	基于断言的验证
Ac	通信弧
ACP	通信进程代数
ALSP	聚合级仿真协议
AOI	关注区域
AP	原子命题
API	应用程序接口
ARC	阿德莱德精化检查器
A_{SV}	状态变量弧
A_T	过渡弧
BB	基本BOM
BDD	二元决策图
BHQ	营部
BHQSM	营部状态机
BID	电池编号
BL	行为层
BOM	基础对象模型
CB	组合BOM
CBM&S	基于组件的建模与仿真
CBSE	基于组件的软件工程
CBT	可组合行为技术
CCA	通用组件架构
C_{CP}	通信端口颜色集
CCS	米尔纳通信系统演算
CL	通信层
CM	概念模型
CODES	可组合离散事件可扩展仿真
COST	面向组件的仿真工具包
CP	通信端口
CPN	有色Petri网
CPN-CM	彩色 Petri 网组件模型
CPN-ML	有色Petri网的 ML 脚本语言
CSP	Hoare 的通信顺序进程
C_{SV}	状态变量的颜色集
CTL	计算树逻辑
DEDS	离散事件动态系统
DES	离散事件系统
DEVS	离散事件系统规范
DIS	分布式交互仿真
DMC	发现、匹配与组合
DOT	DOT文件格式
EC	事件控制器
EFSM	扩展有限状态机
EIC	DEVS 输入端口耦合
EOC	DEVS 输出端口耦合
EXPR	表达式
FA	野战炮兵
FD	战场数据
FDC	火力指挥中心
FSM	有限状态机
HLA	高层体系结构
HPC	高性能计算
IC	DEVS 内部耦合
IDE	集成开发环境

INT	整数
ISV	状态变量的初始化函数
JCSP	基于 Java 的通信顺序进程
JSIMS	联合仿真系统
JUNG	Java 通用网络图函数库
LCIM	概念互操作性层级
LTL	线性时序逻辑
LVC	Live, 虚拟，或构造性仿真
MBSC	基于模型的仿真组合
MCT	模型耦合工具包
MDF	矩阵定义形式
MDP	马尔可夫决策过程
MOCCA	基于组件的网格环境
MPD	马尔可夫决策过程
MUSCLE	多尺度耦合库与环境
NET	网络
OMT	高层体系结构对象模型模板
OSA	开放仿真体系结构
OWL	网络本体语言
PAT	进程分析工具包
PIPE	平台无关 Petri 网编辑器
PLTL	概率线性时序逻辑
PN	Petri网
PNML	Petri网标记语言
POI	BOM 交互模式
RS	需求规范
SAT	布尔可满足性
SCT	语义可组合性理论
SCXML	状态图可扩展标记语言
SE	软件工程
SIMNET	仿真网络
SISO	仿真互操作标准组织
SL	结构层
SM	语法匹配
SML	脚本语言
SMM	状态机匹配
SSM	静态语义匹配
SV	状态变量
SV_{IN}	输入状态变量
SV_{OUT}	输出状态变量
TCSP	定时通信顺序进程
TENA	测试与训练使能架构
TOT	目标时间
UML	统一建模语言
V_{CP}	通信端口变量
V&V	验证与确认
VVA	验证，验证与认证
VVT	验证、验证与测试
XML	可扩展标记语言
XMSF	可扩展建模与仿真框架
X_T	发射向量

第一部分

Part I

知识

Epistemé

知识在希腊语中意为“知道”。它是理论知识；一种有原则的理解体系；一个由基本思想和集体预设构成的体系，这些思想和预设决定了在特定时期内哪些知识在智识上是确凿的；纯科学；知识探讨主题的“是什么”与“为什么”。

第一部分涵盖所讨论研究的认识论，概述问题域和解决方案域的理论、概念、原理、范式、哲学与理论依据。从本质上说，第一部分包含理解第二部分所讨论的问题及拟议解决方案所需的理论知识和背景信息。

“如果你不能简单地解释它，那就说明你还没有充分理解它。”

- 阿尔伯特·爱因斯坦

第1章 引言 Chapter 1 Introduction

本章介绍本论文所呈现研究的开篇陈述和一般信息。本章概述了背景、历史、所研究问题的形式化定义与基本哲学，并涵盖研究的动机、目标和范围，以及本论文的贡献。最后，本章介绍论文的组织结构。

1.1 背景与开篇视角

过去五十年来，几乎所有科学领域都经历了一场革命性的发展。这一进步主要源于数字计算机应用的惊人增长，以及随之而来的计算机仿真时代的兴起 [1]。正是计算能力和资源的涌现及广泛普及，使人们能够通过复杂模型及其仿真开展全新的实验 [2]。如今，计算机仿真已广泛应用于各种科学学科和应用领域。它们被用于研究复杂系统，并在不干扰实际系统的情况下深入了解现有系统的运行。此外，它们还用于在系统实现之前测试新的系统概念、对未来系统的行为进行可视化和预测；也用于分析和解决问题、得出结论，并辅助关键决策过程 [3]。因此，计算机仿真被视为科学的第三大分支 [4]，与前两个分支——理论和实验——并列。

建模与仿真（M&S）是一门拥有自身知识体系、理论和研究方法论的学科 [4]。M&S 的目标与系统理论一致，包括对真实系统进行建模与分析、设计与综合、控制、性能评估和优化。建模与仿真领域长期以来一直致力于为实现这些目标提供支持。随着设计复杂仿真系统的方法和技术进入网络中心化时代，M&S 行业的关注重点转向了人们普遍认可的潜在收益，即降低开发成本、时间和系统复杂性 [5]。这是因为建模与仿真开发过程成本高、耗时长且资源密集。模型可能规模庞大、结构复杂，其开发需要投入大量时间、资源和领域专业知识。此外，还需要付出巨大的努力来评估模型是否正确以及是否满足其需求。因此，建模与仿真领域高度重视质量设计原则及其基础支持理论，以缓解这些挑战。人们已经认识到，每次需要模型时都从头开始构建模型是低效的。相反，模型复用的实践日益受到重视，这一实践源于软件复用的愿景，而软件复用最初是在 1968 [6] 中提出的。显然，这种方法看起来非常具有吸引力，但在实施过程中会面临许多障碍，例如设计缺乏灵活性和适应性、集成困难、接口不匹配、规范不完整等 [7]。这些障碍是

公认的难以解决的研究挑战，如今已成为软件工程和建模与仿真领域的主要研究兴趣 [8]

1.1.1 基于构件的软件工程

基于构件的软件工程（CBSE）已被视为构建复杂系统的关键推动因素，其方式是组合已开发并准备集成的软件构件 [8]。

> **软件构件**
> 软件构件被定义为一种组合单位，它独立开发，并可与其他组件组合以构建更大的单位。它必须具有明确规定的接口，以便与其环境通信；同时，实现必须封装在组件内，不能从环境中直接访问 [9]，，因此第三方无需了解实现细节即可轻松使用它[8], [10]。

通过组件构建软件，有助于推动软件工程领域发生重大范式转变。基于构件的开发这一理念的基本哲学，是通过组装预制的软件构件，快速构建软件应用程序，从而完成软件开发过程，并将这些可互操作的软件构件存档到日益庞大的存储库中，以供后续重用 [11]。基于组件的软件工程倡导模块化原则。这从根本上有助于通过将现实世界的复杂性分解为若干部分来加以掌控 [12]，并使设计人员能够针对不同目的使用和重用适当的部分。这些部分是以基于组件的方式构建的子系统。这些子系统组件可能由不同团队分别开发，也可能是为与当前使用语境无关的不同目的而开发的。基于组件的软件工程具有诸多优势，例如有效管理复杂性、逻辑分离、缩短时间并降低成本、提高生产力、改善质量、提升一致性、增强可靠性，以及扩大可用性范围。此外，现实世界问题日益增强的关联性，体现为对组合跨领域解决方案的需求 [13]，，因此也需要支持在更广泛的用户群体中共享知识。因此，基于组件的软件工程是软件工程的一个分支，致力于通过组合组件来构建能够按照用户需求执行功能的软件系统 [14]。

在基于组件的软件工程（CBSE）中，构件集成和构件组合是两个截然不同的术语。构件集成仅仅是将组件连接在一起，而组合还包括对所得组件组合的语义行为进行推理 [14]。随着构件技术的出现，集成问题正逐渐成为过去。相反，如何预测组件组合的涌现行为，以及如何推理组件能否良好协同等更为关键的问题，如今正受到广泛讨论。构件组合支持这类推理，并为基本推理提供依据，以证明所得组件组合的有效性、运行时兼容性及涌现行为。集成与组合之所以不同，主要是因为组件接口未能提供足够的信息，无法确定组合后的组件能否良好协同[14]。接口只能帮助确定某个组件能否与另一个组件连接，

但无法支持对组件组合涌现属性的推理 [14], [15]。构件组合有望提供这样的理论依据；然而，这仍是一个开放研究课题。

1.1.2 基于组件的建模与仿真

受基于组件的软件工程这一学科的启发，建模与仿真领域也开始通过重用先前开发并验证过的“仿真组件”，并根据用户预期目标将它们组合成新的仿真模型，来开发仿真模型 [16], [17], [18], [19], [20]。应对任何大型复杂仿真问题的基本且有效的策略都是“分而治之”。基于组件的仿真开发中的一个主要理念，是创建自包含且可独立部署的模型。因此，不同的仿真研究人员能够各自独立地处理不同组件，无需彼此进行太多通信（尤其是无需共享保密领域知识），而这些组件仍能无缝协同工作。此外，在维护阶段，可以修改某些组件而不影响其他所有组件 [21]。

在仿真领域，关于基于组件的开发的研究归属于可组合性范畴 [22], ，其中仿真模型被视为构建块，并称为“模型组件”。

模型组件[1]

> 模型组件是仿真模型中的一个独立元素，符合特定的组件标准，具有定义明确的功能（输入/输出）和行为。这些功能和行为通过其接口呈现，接口描述了该组件与其他组件之间的通信方式，并提供其内部行为的形式化描述。模型组件并非独立组件，但可以独立部署，并可由第三方进行组合，无论是否经过修改 [19]。

在基于组件的开发中，人们将一些可复用的基本模型组件组合起来，以创建复杂而精密的仿真，而无需从头开始构建。如果模型组件的输入和输出在物理上彼此匹配，就可以将它们组合起来，但很难判断这种组合是否有意义。此外，除非检查可组合性的正确性，否则无法确定它是否会按照预期需求运行。

可组合性是模型的一种性质，因为它本质上涉及建模层上的问题对齐 [13], ；在这一层，可组合性被视为通过选择和集成可复用的基本模型组件来创建复杂模型。如果一组组件的输入和输出相互兼容，就可以将它们集成起来；但为了确保它们的组合在所需的可执行场景中有效，我们需要研究其可组合性程度。

随着组件数量略有增加，且这些组件本身具有一定复杂性，可组合性会逐渐成为更具挑战性的问题。在

[1]应当将“模型组件”一词与“组件模型”一词区分开来；本文中的后者指的是基于组件的软件工程平台（如 CORBA、EJB 等）所采用的底层技术。

存在功能性和非功能性应用需求的情况下，这会对需求验证所需的工作量以及日益增强的动态性产生重大影响。尽管单个组件已通过预验证，但通常是在有限的语境中进行验证，并依赖一些组合后可能不再成立的假设。因此，系统验证的复杂度会随着应用数量 [23][2] 呈指数级增长。

1.1.3 使用 Petri网进行建模与分析

Petri网（PN）是一种用于对复杂系统建模的机制，其中的状态和事件可以按照特定规则和明确条件进行操控。PN 形式化方法由 Carl Adam Petri 于1962年提出。它为并发系统及其行为的建模提供了一种优雅且实用的图形和数学形式体系 [24]。

Petri网图非常适合表示离散事件系统（DES），其操作取决于可能很复杂的控制方案。Petri网图直观易懂，能够捕捉系统的大量结构和行为信息。考虑使用 Petri网对 DES 建模的另一个原因，是经过三十多年的发展，已经形成了大量分析技术，可用于推理 Petri网模型的结构和行为属性。这些技术包括可达性分析、状态空间分析、模型检验以及线性代数技术 [25]。

过去三十多年来，Petri网研究沿着两个方向发展：（i）佩特里网理论，侧重于开发 Petri网应用所需的基本工具、技术和概念；（ii）应用佩特里网理论，主要关注将 Petri网应用于系统建模及其分析。要在这一方向上取得成功，需要充分了解 Petri网所应用的领域，以及 Petri网理论和技术 [26]。

1.1.4 使用进程代数进行建模与分析

进程代数是一种用于建模和分析研究并发进程的代数方法。它包含多种用于建模并发系统的代数形式化方法。这些形式化方法由用于规范进程的代数语言构成，并以代数定律的形式提供演算规则，从而能够操作和分析进程描述，并对其正确性和等价性进行形式化推理 [27]。主要的进程代数形式化方法包括：

- CCS，米尔纳通信系统演算
- CSP，Hoare 的通信顺序进程
- ACP，通信进程代数
- LOTOS，时序规范语言

1.1.5 模型验证

在建模与仿真中，验证关注的是正确地构建模型。通常，验证被定义为一个过程，用于确定模型是否已正确实现 [28] 以及是否与其规格一致 [29]。原则上，

[2] 尽管所提及的文本对应的是在物理上可组合的电子组件，但可组合性复杂度问题仍然相同，并且不同社群对此有着共同的理解。

验证关注的是将模型需求转换为概念模型，再将概念模型转换为可执行模型的准确性 [29]。概念模型与可执行模型之间的区别至关重要，也是建模与仿真中的一项基本原则。概念模型是对真实系统的抽象描述 [30], , 基于给定的需求和建模目标构建。随后，这一模型会经过细化并实现为更具体的可执行模型。从这些角度来看，概念建模是模型设计的一个子集 [31]。概念建模的过程是从问题情境出发，经由模型需求，最终定义要建模的内容；它独立于模型的实现细节 [30], , 而这些细节随后会通过可执行模型加以处理。

1.2 开篇视角摘要

从本质上讲，基于组件的方法深受建模与仿真领域青睐，常用于构建大型复杂模型。然而，为确保模型正确并符合其需求规范，需要投入大量精力评估其可组合性程度。在建模与仿真领域，模型验证学科提供了基本概念和基本原理，用于全面研究可组合性程度，并推理模型相对于给定规范的正确性。然而，现有的基于组件的仿真框架所提供的内置验证技术有限，甚至完全没有。因此，人们会考虑采用第三方方法，例如 PN 分析技术和进程代数，从不同深度全面检查组合模型。

这些子主题：（i）基于组件的建模与仿真、（ii）PN /CSP分析和（iii）模型验证，是本论文的基础支柱和理论基础，并将在第2、3和4章中分别详细阐述。

1.3 预备知识

基于前文的讨论，本节给出本文所研究问题的形式化定义。为陈述问题，使用以下定义：

1.3.1 定义1：组件集

令$\mathbf{C = \{c_1, c_2, c_3 \ldots, c_n\}}$为从组件库 R 中发现并选取的给定组件集，该选取基于对真实系统的抽象。

1.3.2 定义2：需求规范

系统模型的需求规范定义为一个元组：

$$\mathbf{RS = \langle O, S \rangle} \text{ where:}$$

$\mathbf{O = \{o_1, o_2, o_3 \ldots, o_n\}}$ 是一组目标（或目的），并且
$\mathbf{S = \{s_1, s_2, s_3 \ldots, s_n\}}$ 是一组系统约束（或系统性质） .

目标：

目标 $o_i \in O$ 可定义为组合模型的一个可达“最终状态”，或组合模型产生的聚合期望输出（数据值或事件），而单个组件无法产生该输出。

系统约束：

在建模术语中，系统约束 $s_i \in S$ 定义为必须满足的系统属性；例如，必须达到的良好状态，或必须避免（永远不能达到）的不良状态。

约束的概念不同于目标，因为约束可能是必要的需求，但并非最终目标。例如，制造系统不仅应生产所需产品（目标），还应满足安全需求（约束）。

1.3.3 定义 3：组合与可组合性模式

设 CM〈$c_1, c_2, c_3 ..., c_n$〉为给定组件集 C 的一种组合，该组合采用特定的可组合模式 P 构成。模式描述组件如何相互连接，即组件的拓扑结构，并为可组合性验证提供重要信息。可组合模式可以是顺序、并行、Fork、Join、迭代或复合。

1.3.4 定义 4：可满足性算子

对于需求规范 RS 中的每个元素，可满足性算子 $\models$ 将给定的组合模型 CM 映射为布尔值（真或假），形式化描述如下：

- $CM\langle c_1, c_2, c_3 \ldots, c_n\rangle \models_i o_i \in O \rightarrow true \mid false$
- $CM\langle c_1, c_2, c_3 \ldots, c_n\rangle \models_j s_j \in S \rightarrow true \mid false$

对于每个关系 $\models_i$，我们定义一个验证函数（算法或定理），可满足性算子据此映射其结果值。该验证函数用于确定给定的组合模型是否满足所需属性。

1.4 问题陈述

根据上述定义，问题陈述如下：

“给定一个组合模型 CM，该模型由组件集 C 中的组件通过模式 P 组合而成，并给定需求规范 RS，我们能否验证 CM 满足需求规范中给出的所有目标和约束？”

形式化：

$$CM=Compose([c_1, c_2, c_3 ..., c_n], P) \wedge RS=\langle O, S\rangle \rightarrow \{CM \models_{\boldsymbol{i}} \forall o_i \in O \wedge CM \models_{\boldsymbol{j}} \forall s_j \in S\} \quad (1.1)$$

本论文提案将这一问题陈述视为本研究的起点和基础。本文将展示建模者如何借助我们提出的验证框架，正确组合组件模型，并在不同层次上验证这种组合。

1.5 方法

本节概述所采用的方法和方法论。根据软件工程原则，本节分为两个主要部分：（i）问题域和（ii）解决方案域。

1.5.1 问题域

> 问题域（或问题空间）是一个工程术语，指定义问题及其约束的所有信息。它包括问题所有者[3]希望实现的目标、问题存在的语境，以及定义任何解决方案产品所需基本功能或其他方面的所有规则。它代表解决方案必须运行的环境 [Wikipedia]。

本文中提供的所有与建模与仿真、基于组件的模型开发、概念建模、模型组件、可组合性、模型验证以及可组合性正确性问题相关的信息，都属于问题域。尤其是，第2章涵盖了问题域的主要方面，并详细讨论基于组件的建模与仿真。以下小节简要介绍了所选的问题域规范方法。

基础对象模型（BOM）作为可组合性框架

本文选择 BOM 作为一种组件规范标准，可用作在概念层开发模型组件的基础。这些组件经过组合，并接受可组合性验证过程的评估，以确认它们满足给定需求，从而构成问题域的组件框架。

需求规范模板

本文定义了“需求规范模板”，用于编制需求规范。该模板主要包含一组目标和约束（针对标准属性或特定场景属性），组合模型必须满足这些目标和约束，才能证明其正确性。

1.5.2 解决方案域

> 解决方案域（或解决方案空间）是指定义问题拟议解决方案的所有信息。它包括有助于解决所研究问题的概念、原理、方法、技术、算法、程序、软件架构师、框架、过程和推荐实践。

以下小节将简要概述本论文采用的方法：

[3] 问题所有者可以是客户、解决方案购买者、组织或潜在目标群体。问题所有者将问题视为机遇，而解决方案工程师则看到自己必须为之提供解决方案的问题。

多层级可组合性验证

组合模型会在不同层级进行多次迭代，以验证其可组合性。每个层级对应验证过程中的一个阶段。成功验证某一层级的可组合性后，再迭代下一层级。完成所有层级后，即可认为这些组件完全可组合。第2章将详细讨论这些层级，第5章将讨论这些层级的验证。

PN 形式化方法

PN 形式化方法（尤其是有色Petri网扩展）被选用于创建基于 BOM 概念模型的可执行模型。所提出的框架会自动将 BOM组件转换为 PNML[4] 形式的可执行组件，或基于 CPN 的组件；这些组件可在相应的 PN 执行环境中执行或进行验证。

CSP形式化体系

选择带有 Timed-CSP 扩展的 CSP[5] 形式体系，作为另一种适用于基于 BOM 的概念模型的可执行建模语言。所提出的框架将 BOM组件转换为可执行的 CSP 进程组件，并将其组合起来以供执行和验证。

自动转换工具

提出了一种自动转换工具，可将 BOM 组件模型转换为所选的可执行建模形式体系，例如基于 PNML、CPN 或 CSP 的可执行模型。此外，可能还需要提供 BOM 无法建模或表示的额外细节。

动态分析方法

在动态语义可组合性层面，选择了三种主要的动态分析方法，用于验证基于 BOM 的组合模型的可组合性：

代数分析方法

该方法使用 PNML 格式，将 BOM 组合转换为经典 PN 模型，并使用 PN 代数验证其属性。

状态空间分析方法

该方法基于有色Petri网和状态空间分析。CPN工具是一种强大的仿真和验证工具。状态空间分析是一种非常精确的正确性推理技术，但在计算能力和内存方面成本高昂。因此，还提出了一种约简技术，用于约简组合模型的状态空间图，以避免状态空间爆炸。

模型检验方法

基于 CSP 的模型检验用于对基于 BOM 的组合模型进行形式化验证。在形式逻辑中，模型检验指的是这样一个问题：判断公式或正确性性质 ϕ（使用线性时序逻辑、计算树逻辑[6] 或类似的性质规约形式体系定义）在某个系统

[4] Petri网标记语言 [5] 通信顺序进程 [6] 线性时序逻辑、计算树逻辑

K 的解释中是否为真，记作 $K \models \phi$。选择高效算法来判断 $K \models \phi$ 是否成立[32]。

总之，数十年来，这三种方法一直广泛应用于形式化验证，因此具备扎实的理论基础以及丰富的实用工具和技术。然而，我们认为，本论文首次将这些方法用于基于BOM的模型（或者说，任何建模与仿真组合框架）的可组合性验证，并且这些方法将展现出极大的应用前景和有效性。本论文运用这些方法构建了基本基础，并通过恰当的示例展示了它们的用法。此外，论文还提供了必要的指南，说明如何运用这些方法和工具开发新的验证方法，以应对各种验证问题。
本研究旨在提出一个多层级验证生命周期，用于定义、开发、归档、发现、匹配、选择和组合模型，并对模型进行转换、执行和验证，最终推理组合模型的正确性。该生命周期广泛依托于本研究提出的集成式组件开发、组合与验证框架。它遵循我们提出的过程，在不同层级对组合模型进行验证。M&S 实践者可采用这一生命周期，快速构建、分析、细化并复用模型，从而推动复杂动态系统的建模与仿真进程。

1.5.3 解决方案陈述

基于所提出的方法，解决方案陈述如下：

> “经过验证的组合模型可确保所选组件在所有可组合性层级上均可组合，并通过满足给定目标和符合所需约束来满足需求规范”。

正确组合的模型有助于复用基础组件，从而支持快速模型开发；该模型日后还可作为另一个组件再次复用。

1.6 论文范围

本节概述论文的范围和边界。

1.6.1 正确性

在本论文中，“正确性”是研究的重点。该方法、方法、过程和框架主要处理可组合性验证中的正确性问题。解决方案的性能、效率和成本估算等其他问题目前超出本论文的范围，将作为未来工作加以研究。

1.6.2 验证

验证是模型评估的重要组成部分，且始终与验证密切相关。不过，这超出了本论文的范围。尽管如此，我们认为该框架灵活且开放，因此只需付出少量工作，就能进行必要的扩展以支持验证。

1.6.3 涌现

子系统组合所产生的涌现行为，是可组合性领域中一个重要且开放的研究课题。不过，本论文不讨论这一问题，并将其视为未来工作。

1.6.4 泛化

目前，所提出的方法以基础对象模型为基础，仅用于演示如何将我们的方法应用于现有的组件标准。然而，本论文提出的框架具有开放性，可以泛化以适应其他任何组件标准。此外，也支持异质可组合性。不过，本论文不讨论泛化问题。

1.7 贡献总结

现有的基于组件的建模与仿真领域研究从本质上看较为零散，尤其是在概念层关注模型组件可组合性验证时更是如此。此外，尽管存在不同的可组合性验证方法，但尚未在不同粒度层次上对其进行深入研究。

本研究深入研究基于BOM的模型的可组合性，主要聚焦于不同层次。本文提出了一种多层级基于组件的验证生命周期，用于处理模型开发、发现、选择、匹配、组合、需求规范、转换、实现、执行、分析，尤其是验证等关键问题。

在验证方面，本文的主要贡献包括开发一种可组合性验证框架，该框架整合了不同的方法和技术，以支持组合模型可组合性验证过程中的不同任务。这些不同任务被归类到所提出的基于组件的建模与仿真（CBM&S）生命周期的不同阶段中。我们提出了用于评估组合BOM的结构一致性和行为一致性的方法。在结构评估方面，我们提出了一组静态分析技术，用于验证组件是否能够正确连接，以及它们的通信在语义上是否一致、有意义并且能够按预期被理解。
为确保组合的行为一致性，我们提出一种状态机匹配技术。该技术用于验证组件能否按照正确的因果顺序相互交互并到达最终状态。为进一步评估行为可组合性，我们的框架纳入了三种主要方法：(a) PN代数技术；(b) 基于CPN的状态空间分析技术；(c) 基于 CSP 的模型检验。针对每种方法，我们都开发了自动转换工具，可将基于 BOM 的组合模型转换为相应方法的可执行模型。我们通过三个不同的案例研究进行概念验证，并评估我们的验证框架。
我们还针对每种方法提出了多种扩展，以满足可组合性验证的需求。例如，我们提出了用于自动化 PN代数方法的算法。此外，我们还为状态空间代数方法提出了一种基于CPN的组件模型，用于将 BOM组件（或任何其他仿真组件）描述为可执行模型，并可使用

CPN 执行环境执行。我们还为基于CPN的状态空间分析方法引入了一种状态空间缩减技术，以避免状态空间爆炸的风险。对于基于 CSP 的模型检验方法，我们提出了一个外部函数库，其中包含用于支持各种建模任务的方法，例如为概率系统模型定义概率分布函数。

1.8 论文结构

本论文分为两个主要部分：

第一部分：认识论： 本部分主要介绍理论概念、原理和讨论，包含第1、2、3和4章。

第1章：引言：
第1章概述了本论文所呈现的研究内容。该章介绍了可组合性的概念、历史背景和基本哲学，定义了问题，并简要介绍了研究方法。此外，还介绍了论文范围和主要贡献。

第2章：基于组件的建模与仿真：
第2章详细介绍并讨论了基于组件的建模与仿真，因为它是问题域的基础。本章主要涵盖模型组合的理论、相关问题、不同层次、框架和形式化方法。此外，本章还详细介绍了基础对象模型（BOM），将其作为本论文所采用的一种模型组合标准。

第3章：可执行建模形式
本章介绍了作为可执行建模形式化方法并被视为解决方案域的 PN 和 CSP 的基本理论、定义和分类，同时还描述了本论文后续所采用的分析技术的基本概念。

第4章：验证与分析
第4章讨论验证的理论与原理，并对本论文中使用的一些重要验证技术进行了分类。

第二部分：Techne： 本部分介绍实践方面的内容，包括方法、方法、工具、开发框架和生命周期，也包含与我们为概念验证提出的解决方案相关的示例。本部分由第5、6、7、8和9章组成。

第5章：拟议方法与框架

第5章是论文的核心，提供了我们贡献中最重要的细节。本章介绍了所提出的验证框架和验证生命周期，涵盖了我们在不同阶段提出的方法、技术、算法和程序，作为我们的

贡献，服务于可组合性验证过程的不同阶段。这些阶段及其相关活动概述为可组合性验证生命周期。

第6章：可组合性验证过程

本章介绍所提出的可组合性验证过程，并提供了如何使用我们提出的可组合性验证框架（见第5章）的必要指南。本章使用工作流图描述整个过程，并在每个步骤为建模者提供必要的指导。

第7章：使用PN代数技术进行公平性验证

第7章以一个制造系统为例进行案例研究，说明所提出的框架如何帮助验证组合系统的公平性。本章旨在实际展示代数验证方法。

第8章：使用状态空间分析技术进行模型验证

第8章介绍了一个验证野战炮兵模型的示例，实际演示了如何使用状态空间分析来验证组合系统。文中详细介绍了野战炮兵模型及其需求规范，并展示了所提出的方法如何帮助验证其可组合性。

第9章：模型检验

本章演示了一个使用基于 CSP 的模型检验进行验证的示例。第8章讨论的野战炮兵模型经过修改后成为实时概率系统，并使用基于 CSP 的模型检验进行验证。

第10章：结论与未来工作

本章对论文进行总结与结论，并讨论未来工作。

第2章 基于组件的建模与仿真

Chapter 2
Component Based Modeling and Simulation

可组合性是一项重要的质量特征，也是实现建模与仿真（*M&S*）领域诸多优势的有效手段，但在现实中，它是一个颇具挑战且难以解决的问题。学界围绕其理论与实践层面的复杂问题开展了积极研究。理论上，研究者从多个方面和视角对可组合性进行研究，其主要区别在于不同研究团队所界定的不同“层”或“级别”。实践方面，则对与之相关的各种实际挑战进行了探讨。其中最重要的问题包括组件规范、开发、集成、可组合性验证与确认，这些统称为基于组件的生命周期的各个阶段。本章将详细讨论可组合性的理论与实践层面。

2.1 建模与仿真中的可组合性

在建模与仿真应用中，可组合性的定义各不相同。A. Tolk 在其文章 [13] 中汇总了其中许多定义。Harkrider 和 Lunceford 将可组合性定义为：

> 通过从可复用系统组件池中按逻辑组装出一次独特的仿真执行，从而创建、配置、初始化、测试并验证一项演练，以满足一组特定目标的能力 [33]。

Kasputis 和 Ng 将可组合性定义为：

> 能够跨越各种应用领域、分辨率级别和时间尺度组合模型 [16]

Petty 和 Weisel 在其关于可组合性理论的文章中提出了以下定义，后来由 P. K. Davis 补充：

> 可组合性是指以各种方式选择并组装仿真组件，构成有效的仿真系统，从而有意义地满足特定用户需求的能力[17] [34]。

人们已经认识到，模型组合比通用软件构件的组合更加困难。这一观点基于以下假设：模型更加复杂；模型是为特定目的而开发的，并且依赖于上下文敏感假设 [8] [17]。模型开发是一项艰巨的设计任务，主要是因为这一过程涉及复杂性。如今，这种复杂性日益增加，因此利用预定义模型被认为非常有助于缩短开发时间。因此，模型组合是一种范式，其中现有组件作为构建块，用于构建更大型、更复杂的新模型。组合模型时，必须根据其需求评估其

正确性。简而言之，保证模型组合正确性的可预测性称为可组合性。

2.2 可组合性简史及相关工作

2.2.1 起始阶段

国防研究领域主要研究建模与仿真中的可组合性。在国防仿真的语境下，最早使用“可组合性”一词可追溯至20世纪90年代中期的可组合行为技术（CBT）项目[35]。随后，联合仿真系统（JSIMS）项目将可组合性作为系统目标进行研究[36]。1998年，启动了一个关于基于模型的仿真组合（MBSC）的项目，并在其中为JSIMS开发了一个原型组合环境。1999年，Page和Opper从可计算性和复杂性理论的角度研究了可组合性问题[35]。可组合性成为OneSAF项目的一项关键系统目标1999 [22]。

2.2.2 理论演进

随后陆续发表了一系列文章，探讨可组合性的各种问题与方法，并为后续研究奠定了理论基础。这一系列的重要成果包括：Kasputis 和 Ng [16]; Davis 等人 [37]；Petty 和 Weisel [38]。Petty 和 Weisel 扩展了 Page 和 Opper 的研究成果，广泛考察了“可组合性”一词的用法，并在自动机理论和可计算函数的语境下探讨了组合验证问题。之后，Davis 和 Anderson 在 2003 [17] 中发表了一份综合报告，广泛综述了可组合性，并提出了其在这一领域对国防部[7] 的应用。

2.2.3 标准与框架

此后，关于可组合性的研究仍聚焦于标准组合框架的开发，以及其在建模与仿真各领域的实际应用。2005年，美国海军研究生院启动了可扩展建模与仿真框架（XMSF），旨在开发基于 Web 的仿真环境 [39]。建模与仿真技术的进展催生了不同的分布式仿真标准和协议，例如仿真网络（SIMNET）、分布式交互仿真（DIS）、聚合级仿真协议（ALSP）和高层体系结构（HLA）。Moradi [19] 对这些标准的详细信息作了充分记录。由于这些标准以及分布式仿真本身具有复杂性，人们引入了不同的可组合性框架，以应对这些需求。更通用的框架，例如离散事件系统规范（DEVS）[40], 、开放仿真体系结构（OSA）[41], 、基础对象模型（BOM）[42], 和面向组件的仿真工具包（COST），相继出现，并以不同方式推动了可组合性相关问题的解决。

2.2.4 技术进展

随着计算机工程技术的进步，出现了许多方法，旨在解决建模与

[7] 美国国防部

仿真中的问题及高端需求，例如复杂、动态和自适应系统的表示、庞大相互依赖系统的集成、多分辨率和多尺度建模 [43], 等。在此期间，许多工具和技术基于可组合性范式开发。模型耦合工具包（MCT）旨在支持并简化并行耦合模型的构建 [44]。MUSE 是另一种用于天体物理应用的可组合仿真环境，其中将恒星系统的不同仿真模型整合到一个框架中 [45]。一些框架，如通用组件架构（CCA） [46] 和基于组件的网格环境（MOCCA） [47], , 被提议用于高性能计算，科学组件通过其用户端口和提供者端口直接连接。多尺度耦合库与环境（MUSCLE）提供了一个软件框架，可依据复杂自动机理论构建可组合仿真 [48]。Compo-HLA 是一种用于支持 HLA 组件的环境 [49]。

2.2.5 可组合性验证与确认

这些框架大多缺乏强大的内置可组合性评估支持。因此，一些研究团队开发了第三方组合、验证与确认框架，例如可组合离散事件可扩展仿真（CODES） [20] 和基于语义网的BOM组合框架 [19], , 它们重点关注可组合性的验证与确认。

2.3 可组合性理论

可组合性形式化理论由 Petty 和 Weisel 开创

[34], [38], , 其源于弗吉尼亚建模、分析与仿真中心（VMASC）开展的一项计划。该理论也称为“语义可组合性理论”（SCT）。SCT 旨在通过形式化描述和推理，检查并证明组件的语义可组合性。 模型定义为一个可计算函数：y = f(x)，其中函数可通过有限过程计算，并将每个输入映射到唯一的输出，如图 **Figure 1**

[50]

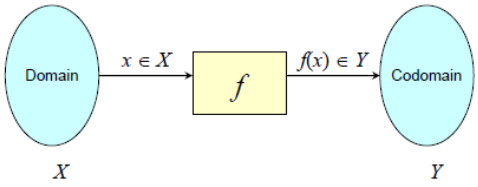


图 1：作为可计算函数的模型（取自 [34]）

仿真是模型f(x)的一系列执行，其中每个执行步骤的输出都是该执行下一步骤的输入：

$$(m_n, o_n) = f(m_{n-1}, i_{n-1}) \quad (2.1)$$

其中 i = 表示输入值；m=表示内存值；o=表示输出值；n=表示当前迭代，如图 **图 2**

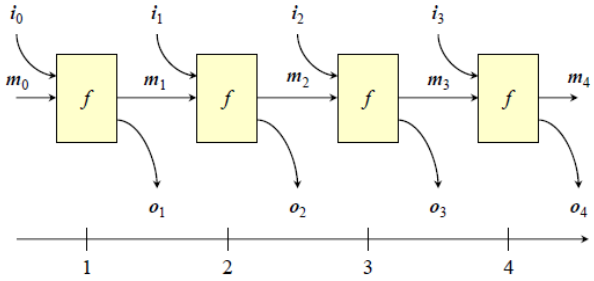


图2：执行序列 （摘自 [50]）

组合定义为将一个函数的输出作为另一个函数的输入：

$$h(x) = f(g(x)) \tag{2.2}$$

图3展示了组合模型的表示形式，该模型通过组合其他模型（f1、f2 和 f3）构建而成。如**图3**所示，组合模型整体也具有一组输入、输出、当前状态和下一状态。

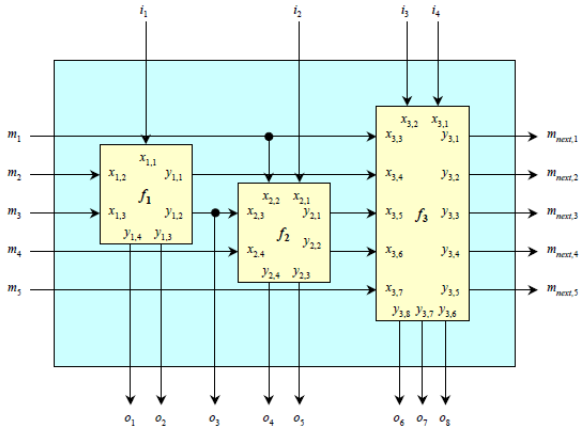


图3：组合模型 （摘自 [50]）

SCT 中的模型组合实际上是函数的组合。由于可计算函数集在组合运算下是封闭的，只要组合存在，任意一组模型都可以组合起来；但无法保证组合后的模型是有用的。因此，SCT 关注的是语义可组合性，即模型组合在语义上是否有效这一问题。

有效性

如果模型能够针对预期用途准确地表示现实世界，则称其为有效模型。对于形式化验证，组合的仿真表示为标记转换系统，其中节点是模型状态，边是函数执行，标签是模型输入。当且仅当组合的仿真接近完美模型的仿真时，该组合才有效。

完美模型

当且仅当模型表示对自然系统 [50] 的完美观测系统时，该模型相对于自然系统 N [8] 才是完美的。

[8] 自然系统 N 是一个真实或想象中的系统。

有关不同模型类别、其等价关系、形式化定理及等价性证明的详情，感兴趣的读者请参阅 [50]。语义可组合性的形式化理论的基本概念包括对模型、仿真、有效性和组合的形式化定义。可组合性理论有助于便捷地重用仿真组件，从而有望缩短仿真开发所需的时间并降低成本 [34] [38] [50]。

2.4 与可组合性相关的概念

本小节将讨论一些与可组合性相关的概念和观点。

2.4.1 可组合性与可重用性

可组合性在许多方面不同于可重用性。Balci 等人将 可重用性 定义为某种制品、方法或策略能够再次或重复使用的程度 [5]。另一方面，Robinson 等人认为，仿真模型复用这一术语可以有多种含义，从复用少量代码，到组件复用，再到完整模型的复用 [51]。可组合性为实现可重用性提供了手段，但模型组合的最终目标并不总是可重用性。例如，在某种特定情况下，一组模块化组件被有目的地构建并组合起来，以构造一个大型模型，但由于其设计高度特定，这些组件无法在另一个项目中复用。要实现广泛的可重用性，组件必须足够通用、可扩展且可适应。对可重用性的要求可能会促成另一种开发方法，例如，在更抽象的层次上进行设计 [9]。对可组合组件的可用性与可重用性进行比较，会在以下两者之间形成权衡：一方面，组件在功能和行为上非常具体，使其能够用于特定情形以满足特定的用户需求；另一方面，组件非常通用且抽象，使其能够在不同情形下反复复用。

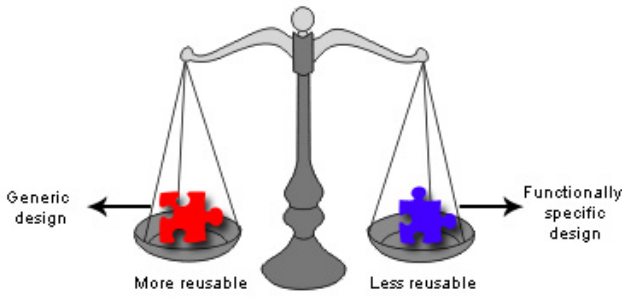


图 4：通用设计与特定组件设计

图 4说明，如果组件采用通用设计，通常会更具可复用性；如果采用功能特定的设计，可复用性则较低。

可组合组件的使用和复用都具有三个层次的透明性。[7]。组件可以看作一个盒子，其中包含接口和内部实现。可组合性透明度分为三个层次：

黑盒组合

在黑盒组合中，用户可以看到接口，但看不到组件的实现。系统会提供用户文档，其中包含组件的输入和输出、需求及限制的详细信息。所有实现细节均被隐藏。客户端将获得契约所承诺的内容。在部署端无法进行更改。黑盒组合的优势在于，开发端完成的测试得以保留，部署端无需进一步测试。

玻璃盒组合

在玻璃盒组合中，可以查看组件的内部结构，但无法对其进行修改。与黑盒复用相比，这种方案具有优势，因为建模者可以更好地了解该盒及其用途。然而，无法对实现进行任何更改。这一层级的优势与黑盒组合相同，但还有一项额外好处：用户可以了解内部实现，并理解组件的运行机制。

白盒组合

在白盒组合中，可以查看并更改盒子的内部及其接口。白盒可以通过继承或委托，与另一个盒子共享其内部结构和实现。此层级的优势在于能够进行修改，因此具有更高的灵活性。不过，此层级会增加部署端的测试负担。

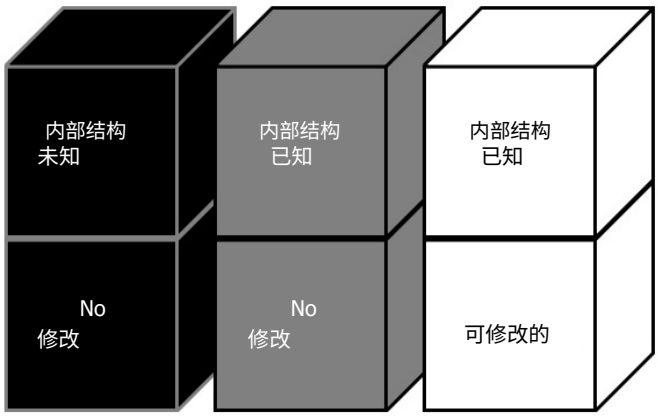


图5：黑盒、玻璃盒、白盒

图5展示了黑盒、玻璃盒和白盒组合之间的区别。

2.4.2 可组合性与互操作性

考虑到前文提及的可组合性定义，IEEE 对互操作性的定义如下：

> 两个或多个系统或组件交换信息，并使用所交换信息的能力

互操作性的概念主要涉及将为不同目的开发的各种类型的系统相互连接，使其能够在不同平台上运行，并就通信方式达成语法和语义上的共识 [**13**]。在

建模与仿真领域，互操作性是指分布式系统中相互连接的不同仿真协同仿真共同场景的能力 [19]。Page 等人 [52] 对可组合性和互操作性的区别如下：

> 可组合性关注建模层面各项问题的协调。底层模型是对现实的有目的抽象，用于实现最终系统所承载的概念化；而互操作性关注的是互操作的软件和实现细节，包括通过接口交换数据元素、使用中间件，以及映射到通用信息交换模型。

2.5 可组合性层级

Petty 和 Weisel 在其可组合性理论中强调了可组合性的两种基本类型：句法可组合性和语义可组合性 [38] [50]。根据该理论，句法可组合性要求可组合组件采用兼容的实现细节构建，例如参数传递机制、外部数据访问和时序假设。句法可组合性关注的问题是组件能否连接。相比之下，语义可组合性关注的是模型能否有意义地组合成一个组合仿真系统，以及组合计算在语义上是否有效。两个组件在句法上可能已连接，因此一个组件可以向另一个组件传递数据，但它们在语义上仍可能无效。**图 6**以隐喻方式展示了句法可组合性与语义可组合性的区别。

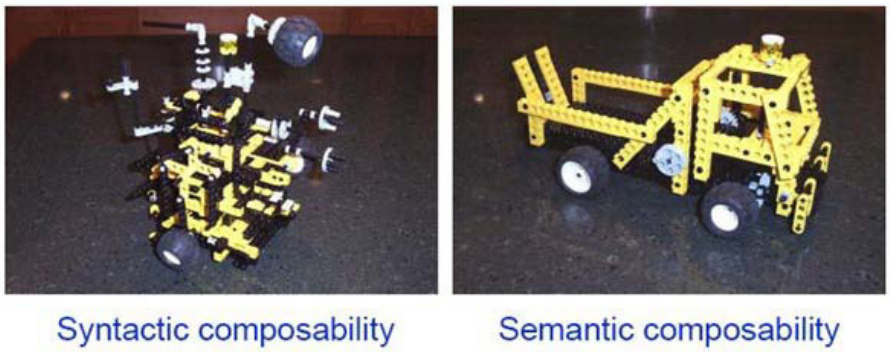


图 6 语法可组合性与语义可组合性（摘自 [38]）

不同研究团队将可组合性划分为不同层级，并对其进行了更深入的研究。模型之间需要在多个层面上达成理解与共识，才能有意义地组合在一起，也就是说，组合后才能产生有意义的结果 [17]。

Davis 建议采用五个层次的区分，即：句法、语义、语用、假设，以及有效性，以研究可组合性 [43]。他将这些层次描述为可组合性的不同一致性，它们共同用于检验模型可组合性的正确性。Petty 和 Weisel 从组合单元的角度提出了九个可组合性层次，分别是：应用、联邦成员、包、参数、模块、模型、数据、实体以及行为 [38]。Tolk 提出了一种名为概念互操作性层级（LCIM）的六层模型，用于研究可组合性和互操作性。该模型包括：技术层、句法层、语义层、语用层、动态层和概念层 [13]。类似地，Medjahed 和 Bouguettaya 提出了可组合性层级，用四个层次检验语义网服务的可组合性：句法、静态语义、动态语义和定性层 [53]。前三个层次的

Moradi 等人采用了 Medjahed 和 Bouguettaya 的可组合性层级，以研究基础对象模型（BOM）组件的可组合性程度 [54]。在本文中，这些层次被视为评估模型可组合性的基本基准。模型可组合性及其正确性的概念，在很大程度上取决于这些层次的一致性，以下各小节将对此进行说明。

2.5.1 语法层面：

在这一层面，需要研究组件的结构，以确定它们是否能够彼此适配；也就是说，一个组件的输出能否被另一个组件作为输入读取，以及相连组件的语法信息（例如消息名称、作用方式和参数数量）是否相互匹配。例如，在军事训练仿真中，“客运飞机”组件在语法上并不适配，因为此处需要的是“战斗机”组件：它以“地面站”组件发出的信号作为输入，以攻击目标，并向“目标”组件输出空袭。客运飞机既不能将目标指示作为输入，也不能向地面目标开火。因此，该组件在语法层面不具备可组合性。

2.5.2 静态-语义层面：

这一层面关注组合组件之间有意义的交互。可组合性的静态语义层面要求对参与组合的组件所交换的数据形成简明且一致的理解。在这一层面，需要确保所有组件对术语、参数、数据类型和单位的理解一致，因此，这一层面主要涉及对组合组件之间交换的信息中各概念含义的相同解释。例如，如果两个组合组件以不同方式解释量的单位，就会在信息交换过程中错误地处理数据值，进而导致用户意料之外的情况。例如，若一个整数数据值表示目标的方位角（单位为度），而另一个组件却将其解释为目标距离（单位为千米），那么这就是语义不匹配。

之所以加上“静态”这一前缀，是因为评估这一层所需的所有信息都是静态的，并且在整个组件交互过程中不会发生变化。

2.5.3 动态语义层：

动态语义可组合性意味着组件在动态上是一致的，也就是说，它们具有适当的状态行为，这是实现预期目标并进而满足用户需求所必需的。可组合性的动态层面确保参与的组件在实现共同目标时具有行为一致性和连贯性。只有当组件在交互过程中处于恰当的状态时，才能实现动态语义可组合性。此外，组件还应具备所需的行为，以推动整体进展。例如，在餐厅的组合模型中，服务员组件可能有两种不同的行为：（i）传统餐厅中，服务员向客户点单、上菜，然后收款；或（ii）快餐店中，服务员先接单、收款，然后上菜。选择正确的行为和正确的顾客组件（即能够与传统餐厅服务员或快餐服务员正确交互的顾客组件），会影响模型的整体可组合性。这个例子说明，组件必须处于恰当的状态，才能实现

进展。一个（期待经典处理的）顾客会永远等着（快餐服务员）上菜，反之亦然。

即使组件处于正确的状态，但如果其行为不正确，组合也可能无法实现其目标。例如，在一个制造系统中，两个机器组件生产两种不同的零件，之后将它们组合成成品；这两个机器组件共用一个机器人组件来输入原材料，因此要求该机器人组件公平地为两台机器提供原材料，使它们获得大致相等的运行机会。如果机器人不公平，产出的合格品比例就会失衡，因此，即使组件处于正确的状态并持续运行，系统仍无法实现其目标。

之所以加上“动态”一词，是因为在组件交互过程中，组件的当前状态等信息会动态变化。

2.5.4 语用层：

意义一致性并不总是显而易见的，因为同一个词会因语境不同而有截然不同的含义 [43]。语用一致性指的是基于语境、具有意义的组件组合。在语言学中，对语言现象与语言使用语境各方面之间关系的研究称为语用学，而语境被定义为由相关的思想、情境、事件或信息构成，并使人能够充分理解某事物的内容[55]。

可组合性的语用层评估通信过程中消息的实际效果与预期效果之间的差异 [43]。对可组合性语用层的研究涉及对计算语言学、认知技术和情境计算的深入研究 [55]。这一层面的一个重要问题是语用歧义。当消息不够明确，且语境未能提供澄清陈述所需的信息时，就会产生语用歧义，从而导致组件未能按照预期目标进行交互。语用歧义的一个例子是克罗伊斯国王与德尔斐神谕的故事（改编自 [56]）：

> “克罗伊斯国王在与波斯居鲁士开战之前，曾向德尔斐神谕求教。神谕回答说：‘如果克罗伊斯与居鲁士开战，他将摧毁一个强大的王国。’克罗伊斯大喜，进攻波斯，结果他的军队和王国惨遭覆灭。克罗伊斯愤怒地向神谕的祭司抱怨，祭司回答说，神谕的预言完全正确。克罗伊斯与波斯开战，确实摧毁了一个强大的王国——他自己的王国。”

本质上，一组组件可能在语法上能够组合在一起，且其通信在语义上有意义并能被理解；但除非所有组件都能动态地保持其基本行为以实现期望的组合目标，并共享正确的语境知识（在实用层面），否则就不能认为其可组合性符合给定的需求规范。

2.6 可组合性框架

可组合性本质上依赖于一个合适的组合框架，该框架能够对其正确性进行准确推理，并支持利用特定组件标准的方法。为了支持建模与仿真中的可组合性，人们开发了各种组件标准及其相应的框架。其中一些框架通过提供所需的形式化方法，为概念建模做出贡献，并影响在概念层开发和组合模型

组件的能力，而另一些则支持在可执行层进行模型组合。这些框架在实践中支持可组合性，因为它们通常提供模型规范、开发和执行等功能。下面简要介绍一些可组合性框架:

2.6.1 离散事件系统规范（DEVS）

DEVS [57] 是一种基于动力系统理论的基于组件的形式主义，旨在描述系统的结构和行为。它通过组件的耦合支持分层模块化模型构建这一概念 [19]。DEVS 本质上是一种模型规范形式体系，但它包含不同的实现框架，例如 DEVS-Java、DEVS-C++ 和 DEVS-Sharp，这些框架用于将 DEVS模型实现为可执行形式。

DEVS模型有两种类型，即原子型和耦合型 [20]。

原子型 DEVS 是一个元组 $\mathbf{M} = \langle \mathbf{X, S, Y,} \delta_{int}, \delta_{ext}, \lambda, \tau \rangle$ ，其中:

$X = \{(p, v) \mid p \in$ 输入端口, $v \in Xp\}$ 是输入端口和值的集合

$Y = \{(p, v) \mid p \in$ 输出端口, $v \in Yp\}$ 是输出端口和值的集合

S 是状态的集合

$\delta_{int} : S \rightarrow S$ 是内部转移函数

$\delta_{ext}: Q \times X \rightarrow S$ 是外部转移函数，其中

$Q = \{(s, e) \mid s \in S, 0 \leqslant e \leqslant \tau(s)\}$ 是总状态集

e 是自上次变迁以来经过的时间

$\lambda : S \rightarrow Y$ 是输出函数

$\tau : S \rightarrow R^{+}_{0,\infty}$ 0, ∞ 是时间推进函数

DEVS 原子组件具有输入 X、输出 Y 和状态集 S。在给定时刻，DEVS模型处于状态 $s \in S$。若没有外部事件，它将在该状态停留由 $\tau(s)$ 定义的寿命。当 $\tau(s)$ 到期时，模型通过端口 $y \in Y$ 输出值 $\lambda(s)$，然后转移到由 $\delta_{int}(s)$ 给出的新状态。由 $\tau(s)$ 所指示的时间流逝而发生的转移称为内部转移。另一方面，外部转移由外部事件的发生引起。在这种情况下，外部转移函数会确定新状态，即 $\delta_{ext}(s, e, x)$，其中 s 是当前状态，e 是自上次转移以来经过的时间，而 $x \in X$ 是已接收的外部事件。时间推进函数可以取介于 0 和 ∞ 之间的任意实数值。若某个状态满足 $\tau(s)=0$ ，则称其为暂态（这将触发一次瞬时内部转移）。相反，若 $\tau(s)=\infty$，则称 s 为被动状态，系统将永远停留在该状态，除非接收到外部事件。

耦合型 DEVS 是一个元组：$M = (X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC, Select)$，其中：$X = \{(p, v) \mid p \in$ 输入端口$, v \in Xp\}$ 是输入端口和值的集合；$Y = \{(p, v) \mid p \in$ 输出端口$, v \in Yp\}$ 是输出端口和值的集合；D 是组件名称的集合；M_d 是一个 DEVS模型，其 $X_d = \{(p, v) \mid p \in$ 输入端口$_d, v \in Xp\}$、$Y_d = \{(p, v) \mid p \in$输出端口$_d, v \in Yp\}$；EIC 是输入端口耦合的集合；$EIC \subseteq \{((N, ip_N), (d, ip_d)) \mid ip_N \in$ 输入端口$, d \in D, ip_d \in$ 输入端口$_d\}$；EOC 是输出端口耦合的集合；$EOC \subseteq \{((d, op_d), (N, op_N)) \mid op_N \in$ 输出端口 $, d \in D, op_d \in$ 输出端口 $_d\}$；IC 是内部耦合的集合；$IC \subseteq \{((a, op_a), (b, ip_b)) \mid a, b \in D, op_a \in$ 输出端口$_a, ip_b \in$ 输入端口$_b\}$；Select 是平局裁决函数

使用 DEVS 建模的系统可描述为原子组件和耦合组件的组合。耦合模型由一组输入和输出端口、一组组件名称 D、一组 DEVS 组件 M_d、输入端口 EIC 和输出端口 EOC 的耦合，以及一组将内部组件彼此连接起来的内部耦合 IC 构成。当两个或多个组件的内部转移安排在同一时间发生时，平局裁决函数将决定由哪个组件继续执行。

图7展示了一个 DEVS 示例。在此示例中，两个原子组件 A 和 B 耦合在一起。两个组件都有两个状态：Send $\tau(s)=0.1$ 和 Wait $\tau(s)=\infty$. 输入端口：*?*接收 和输出端口：*!*发送 在耦合型 DEVS 中定义并彼此连接。

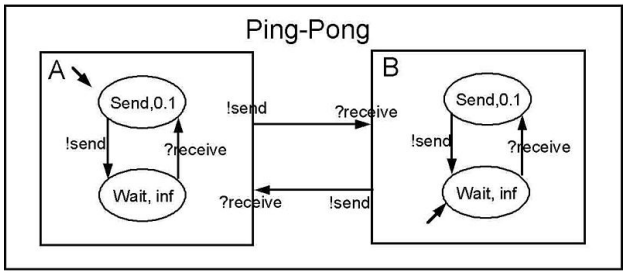


图7：Ping-Pong DEVS [Wikipedia]

2.7 基础对象模型（BOM）框架

SISO [9] 标准将 BOM 定义为：“概念模型的一个组成部分，由一组相互关联的元素构成，可用作开发和扩展仿真及仿真环境的构建块” [58]。

BOM 提供了一种仿真标准，使模型开发人员和仿真工程师能够以可组合对象的形式创建模块化概念模型，

[9] 仿真互操作标准组织

这些对象可作为仿真或仿真环境的基础 [59], [60]。BOM 的概念基于这样的假设：模型、仿真和联邦的组件可以在开发新的仿真或联邦时作为构建模块重复使用 [54]。

BOM具有独特性，因为它提供了一种方法，用于表示概念模型的各个方面；该概念模型捕捉从真实系统（仿真对象）中抽象出的项目的结构与行为描述。随后，BOM允许将这些概念模型映射到一个或多个类定义，这些定义可用于软件设计、各种编程语言，或诸如HLA或TENA之类的分布式仿真架构[10] [61], [62]。

BOM标准还提供了一种通用建模架构，用于定义将在实况、虚拟或构造型（LVC）仿真环境中表示的组件。它非常适合描述模型，包括交互系统、个体及其他实体的结构和预期行为。BOM框架的主要优势在于，它为概念模型在语法和语义层面的有效可组合性提供了良好基础，从而形成一个用于组装系统（即仿真）或系统之系统（即分布式仿真环境）的框架[62]。

尽管 BOM 框架具有这些合理的特性，但它仍缺少所需的行为语义和必要的内置评估技术，而这些对于
建模复杂系统行为，以及推理各个不同层级上模型可组合性的正确性至关重要。因此，对于深入研究模型可组合性，并将所提出的方法应用于基于 BOM 的组合体以阐释该方法而言，BOM 框架成为最合适的候选项，也是首选的组合框架（本论文中）。

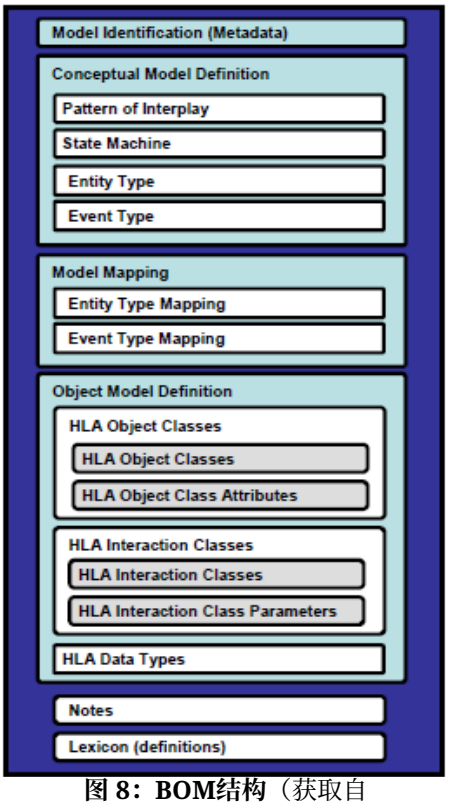


图 8：BOM结构（获取自 [59]）

2.7.1 BOM结构

BOM 由用于指定元数据信息、概念模型和类结构信息的元素构成，这些信息使用 HLA OMT 构造定义 [59]。图 8 展示了 BOM 的不同组成部分，说明如下：

模型标识

模型标识将元数据信息与 BOM 关联起来。其目的是记录 BOM 描述中的某些关键标识信息。它为 BOM 提供最低限度但足够的描述性信息

[10] 测试与训练使能架构

概念模型定义

从可组合性的角度来看，这是 BOM 中最重要的部分，因此也是本论文的主要关注点。要理解这一部分，首先应考虑概念模型的定义：

> **概念模型**
>
> 概念模型是对真实（或拟议）系统的抽象描述或适当简化，之后会进一步细化并实现为更具体的可执行模型（或仿真模型）。就此而言，概念建模是模型设计的一个子集，它根据系统建模的目标，通过迭代过程形成[**63**], [**64**]。

文献中对“概念模型”一词的用法各不相同。概念模型可以是特定的图表，例如 UML 类图，也可以是对仿真对象某个特定方面的文档[11] [**29**]。为了更好地理解 BOM 的概念，不妨考虑房屋建造类比。建造新房时，人们会将建筑特征的概念构想绘制成建筑图纸，这与概念模型（BOM）类似[**60**]。BOM 概念模型定义包含以下部分：

交互模式（POI）

POI 对特定目的或能力进行建模，并通过一个或多个模式动作来表示。对于每个模式动作，都会指定一个或多个发送方和接收方，以便理解概念实体之间的行为关系。POI 通过 UML时序图 [**60**] 表示。

状态机

状态机用于对 BOM 概念实体的行为进行建模。状态机由一组状态构成；每个状态在发生交互模式中标识的退出动作时，都可以转移到后续状态，即下一状态。BOM 的状态机通过 UML状态机图 [**60**] 表示。

实体类型

概念实体是现实世界实体的抽象。它定义了与交互模式中其他实体的关系，并充当事件的发送方或接收方 [**60**]。

事件类型

概念事件包含有关消息或触发器的源、目标和内容（参数）的信息。触发器与消息的区别在于，触发器用于广播信息，而消息则是定向的信息交换，发送方知道消息的预期接收方 [**60**]。

实体与事件表示有关现实世界对象及其交互的数据（物理描述），而交互模式与状态机共同表示组件的动态行为。

[11] 仿真对象是感兴趣的现实世界系统。它是要仿真的对象、过程或现象 [**29**]。

2.7.2 BOM组装

BOM 概念提供了一种组合 BOM 并创建高层 BOM 的机制，这类高层 BOM 称为 BOM组装，如图9所示。表示 BOM组合 的 BOM组装以层次化方式构建，包含有关组合BOM的信息；这些信息进而用于确定复合接口，并表示仿真空间中的联邦成员和联邦[12]。通常，仿真开发人员会在 BOM仓库 中查找适用于仿真的 BOM候选项，并将其组合成 BOM组装（即仿真模型），然后用它来创建实际的仿真 [19]。

BOM组装包含模型标识，以及所表示的概念实体之间相互作用的模式；该模式通过将 BOM 与概念模型视图 [60] 中 BOM组装所标识的各种模式描述动作相关联来提供。

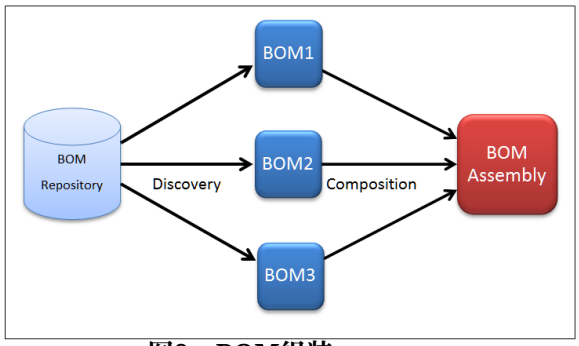


图9：BOM组装

BOM 模型可以使用 XML脚本创建。不过，若要以图形方式构建 BOM 模型，可以使用一款名为 BOM Works 的免费集成开发环境工具 [65] 。图 10 展示了一个使用 BOM Works 开发的示例。该示例与图7所示的 DEVS 示例类似，可用于比较两者的差异。

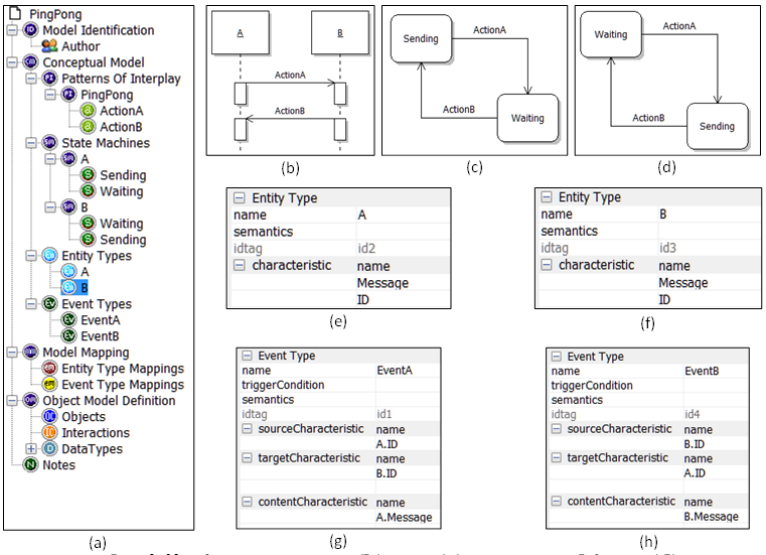


图10：(a) BOM Works 中的 PingPong BOM (b) POI (c) State-machineA (d) State-machineB (e) EntityA (f) EntityB (g) EventA (h) EventB

[12] 尽管后续步骤并非必须使用 HLA，但 BOM 组件组合很可能旨在支持基于HLA的联邦 [59]。

2.7.3 模型映射与对象模型定义

模型映射提供了一种机制，用于将概念模型的元素映射到对象模型定义的类结构元素；这些元素使用 HLA OMT [13] 规范构造进行描述。对象模型定义描述了对象和交互类的结构，以及它们关联的属性和参数。HLA对象类包含HLA属性，HLA交互类包含 HLA 参数。本论文未使用 BOM 的这些部分，但感兴趣的读者可以在 [58], [59], [60], [61], [62] 中找到更多详情。

2.7.4 BOM组合的形式化规范

与 DEVS 不同，BOM 并没有用于指定组件组合方式的图形化和数学形式体系（尽管 BOM 的某些部分，如状态机和 POI，可以用统一建模语言表示，BOM 文档也可以用 XML 描述）。因此，需要一种图形化且形式化的表示方法来描述 BOM 组合。

本节介绍 BOM[14] 的形式化图形规范。我们定义两种 BOM 类型：(i) 基本 BOM 和 (ii) 组合BOM。基本BOM 是不可再分的原子BOM组件，并假设它最多只表示一个概念实体。组合BOM 是由基本BOM 和其他组合BOM 构成的层次组合。

基本BOM

我们提出，基本BOM（BB）可形式化定义为：

$$BB = 〈 EnT、EvT、AcT、S 〉 \tag{2.3}$$

其中：

- **EnT** 是一种实体类型。我们假设基本BOM仅包含一个实体。EnT 定义如下：

  EnT = 名称 {特征：类型}

  其中，名称是由标识符唯一确定的实体名称[15]，特征是实体属性的集合。每个特征都由标识符唯一确定，并具有一个类型[16]

- **EvT** 是事件类型的集合，每种类型均包含发送方、接收方和内容 Evt = {(名称, 发送方，接收方，{内容：类型}) | 名称 ∈ 标识符，发送方和接收方 ∈ EnT，内容 ∈ 标识符：类型 ∈ 类型}

- **S** 是状态的集合，每个状态都有一个退出条件和一个下一状态：

  S = {(名称, 退出条件{动作, 下一状态})} | 名称 ∈ 标识符，动作 ∈ Act, 下一状态 ∈ S

[13] 高层体系结构对象模型模板 [14] 这些概念并非新概念，在其他基于组件的方法的文献中也有介绍 [21]。在本论文中，这些概念在 BOM 中的应用旨在便于规范制定并提升可理解性 [15] 标识符是由字母和数字组成的唯一序列，且以字母开头。 [16] 类型：= 整数 | 字符串 | 双精度浮点数 | 复数

▪ **AcT** 是一个动作集，每个动作都有名称、发送方、接收方和关联的事件：AcT= {(名称, 发送方, 接收方, 事件) | 名称 ∈ 标识符，发送方和接收方 ∈ EnT，事件 ∈ EvT}

组合BOM

组合BOM（CB）可形式化定义如下：

$$\text{组合式基础对象模型} = \langle AcT_{IN},\ AcT_{OUT},\ POI \rangle \quad (2.3)$$

其中：▪ $\mathbf{AcT_{IN}}$ 是一个输入动作集，这些动作从其他 BOM 接收。如果组合BOM是封闭的，该集合可以为空。AcT_{IN} = {(名称, 发送方, 接收方, BOM) | 名称 ∈ 标识符，发送方和接收方 ∈ EnT，BOM ∈ 文件}▪ $\mathbf{AcT_{OUT}}$ 是一个输入动作集，这些动作发送给其他 BOM。如果组合BOM是封闭的，该集合也可以为空。AcT_{OUT} = {(名称, 发送方, 接收方, BOM) | 名称 ∈ 标识符，发送方和接收方 ∈ EnT，BOM ∈ 文件}▪ **POI** 是定义基本BOM或组合BOM如何（通过动作）彼此连接的交互模式。它将发送动作列表映射到接收动作列表。“！”符号表示发送，“？”符号表示接收。

$POI = \{(\{!AcT_{SEND}\}, \{?AcT_{RECV}\})\} \mid AcT_{SEND}\ \&\ AcT_{RECV} \in AcT$

示例

例如，图10中的 BOM 可以形式化地表示为：

$BB_0 = \langle EnT, EvT, AcT, S \rangle$ 其中：

$EnT = EntityA\ \{C_0(\text{消息:字符串})\}$

$EvT = \{E_0(EventA, BB_0, BB_1, BB_0.C_0), \{ E_1(EventB, BB_1, BB_0, BB_1.C_0)\}$

$\text{动作} = \{ A_0(ActionA, BB_0, BB_1, E_0), A_1(ActionB, BB_1, BB_0, E_1)\}$

$S = \{ S_0(\text{发送}, A_0, S_1), S_1(\text{等待}, A_1, S_0)\}$

表 1：实体 A

BB_1 = 〈 EnT, EvT, S, AcT 〉 其中:
EnT = EntityB {C_1(消息:字符串)}
EvT = {E_2(EventA, BB_0, BB_1, $BB_0.C_0$), { E_3(EventB, BB_1, BB_0, $BB_1.C_1$)}
动作 = { A_2(ActionA, BB_0, BB_1, E_2), A_3(ActionB, BB_1, BB_0, E_3)}
S = { S_2(等待, A_2, S_3), S_3(发送, A_3, S_2)}

表 2：实体 B

类似地，组合BOM CB_0 可以形式化地描述为：

组合式基础对象模型$_0$ = 〈 AcT_{IN}, AcT_{OUT} , POI 〉其中:
AcT_{IN} = ∅ (因为没有来自其他任何 BOM 的传入动作)
AcT_{OUT} = ∅
POI = {I/O_0(!A_0 , ?A_2), I/O_1(!A_3, ?A_1)}

表3：组合BOM

我们提出了一种用于表示基本BOM及其组合的图形表示法，如**图11**所示。在该图中，两个基本BOM实体 EntityA 和 EntityB 进行了组合。

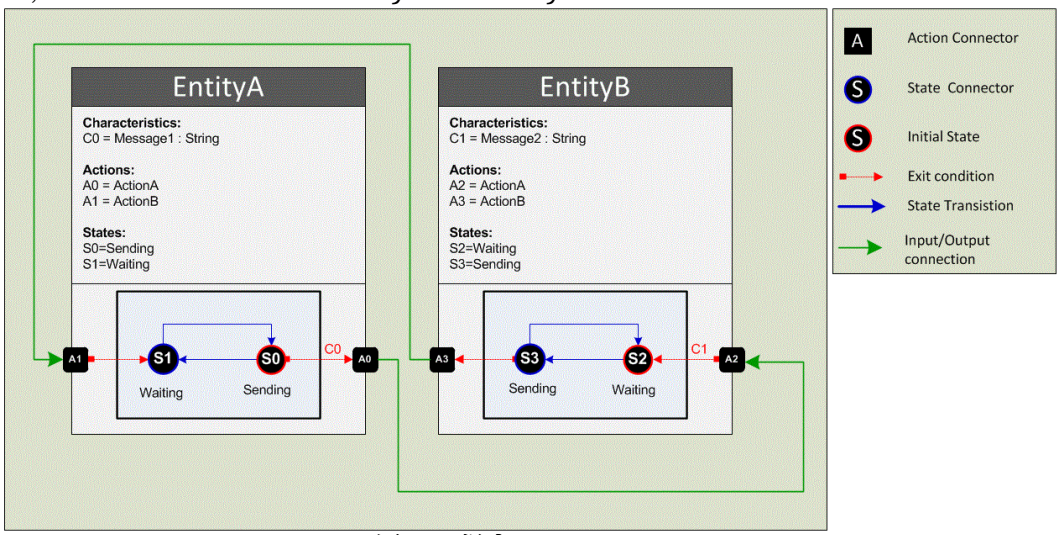


图11：组合BOM

组件的一般信息（如实体名称、特征、动作和状态）定义在主块中。下方的块显示状态及其变迁（以蓝色箭头表示）。每个变迁都映射到相应的动作（以红色箭头表示），并带有参数标签（特征ID）。箭头的方向表示关联动作的类型（发送或接收）。BOM组合通过连接器（绿色）表示。

2.7.5 总结

在本文中，我们利用 BOM 作为概念建模框架的能力，因为它提供了采用 XML 规范的组件标准；为可执行模型的进一步开发提供指南，并有助于确定模型或其部分是否适合复用；最重要的是，它对句法和语义可组合性提供了有力支持。本文将展示，如何通过可组合性评估，发挥 BOM 现有的潜力，从而在基于 HLA 的仿真中准确、快速地构建和修改相应的联邦成员，进而推动分布式仿真领域的发展。

第3章 可执行建模形式 Chapter 3 Executable Modeling Formalisms

本章将讨论两种流行的模型描述形式化方法，即 *Petri*网和通信顺序进程（*CSP*）[17]，，它们通常用于并发系统的建模、执行（或仿真）和验证。本章介绍 *PN* 和 *CSP* 的基本内容、理论、属性、分类、建模方法和分析技术。由于 *PN* 和 *CSP* 在并发建模及分析技术方面积累了大量知识，因此本文将二者都视为解决方案域的一部分。本文借鉴并运用这些方面进行可组合性验证。

PN 和 CSP 形式化体系相互关联，因为它们都用于对同一类系统——并发系统——建模。与迁移系统或自动机等其他系统不同，并发系统的形式化体系严格建立在并发理论的基础上。并发理论的主要贡献者包括：Carl Adam Petri，他开创了交互顺序进程的概念并提出了 Petri网；C. A. R. Hoare，他专注于为并发系统开发编程语言（CSP）；以及 Robin Milner，他提出了通信系统演算（CCS）和 π-演算。这些是对并发系统进行形式化建模的不同方法，属于并发数学理论家族，统称为进程代数或进程演算。CSP 也是进程代数的一员。PN 和 CSP 的主要区别在于，前者基于图，而后者基于文本描述。然而，两者都为可执行系统建模提供了强大的形式语义，并共享大量理论原理和实用技术知识，可用于分析和验证复杂行为模型。本文提出使用这两种形式化体系对组件的可执行形式建模，并研究其可组合性。

3.1 Petri网

Petri 网由 Carl Adam Petri 于 1962年提出（并以他的名字命名）。它们提供了一种优雅且实用的图形化和数学形式体系 [24]。Petri 网的核心思想是分别表示各子系统的状态。通过这种方式，可以非常有效地表示系统的分布式活动。Petri 网广泛应用于各种系统的建模和控制。尤其是在离散事件动态系统（DEDS）中，[18] 许多属性，例如同步、顺序性（生产者-消费者问题）、并发以及

[17] 如今，CSP 名称中的“顺序”一词多少有些名不副实，因为现代 CSP 允许将组成进程定义为顺序进程，也允许将其定义为并行进程 [Wikipedia]。[18] DEDS 的例子包括空中交通管制系统、自动化制造系统、计算机与通信网络、嵌入式与网络化系统以及软件系统等。这些系统中的活动受人类制定的操作规则支配，其动态行为通常由离散事件的异步发生驱动 [67]。

冲突（互斥）、并发和选择，都可以使用 Petri 网得到良好的表示和分析 [66]。Petri 网的结构属性和行为属性已成功用于解决复杂动态系统中的各种问题。过去三十多年间，这些方面取得了显著进展。Petri 网最重要的特征是局部性、并发、图形表示和代数表示原理。它们不仅可用于规范和分析结构性系统设计，也可用于设计系统行为。[66], [67]。

Petri网具有两个有趣的特征。首先，它们能够对具有复杂行为的系统进行建模和可视化，这些行为包括并行性、并发、同步和资源共享。其次，这些网的属性、分析方法和定理已得到广泛研究 [68]。

3.1.1 Petri网定义与概念

在Petri网中，建模的两个基本元素是库所和变迁。 **事件**与变迁相关联，当满足某些条件时，变迁便会发生。与这些条件相关的信息包含在库所中。库所分为两种：输入库所和输出库所。输入库所与变迁发生所需的条件相关联。输出库所与受变迁发生影响的条件相关联 [25]。变迁、库所及它们之间的某些关系定义了Petri网图的基本组件。Petri网图有两种类型的节点：库所和变迁，并由弧连接。它是一个二部图，因为弧不能直接连接相同类型的节点；相反，弧连接库所节点与变迁节点，以及变迁节点与库所节点[25]。

3.1.2 Petri网图

从数学上来说，PN 是一个五元组：$PN = \langle P, T, F, W, M_0 \rangle$，其中：

- P 是有限库所集 $P = \{p_1, p_2 \ldots p_m\}$，在 PN 图中表示为椭圆形节点；
- T 是有限变迁集 $T = \{t_1, t_2 \ldots t_n\}$，在图中表示为线条或矩形节点；
- F 是流函数，使得 $F \subseteq (P \times T) \cup (T \times P) \rightarrow N$ [19]
- $W: F \rightarrow N^+$，其中 $N \in \{1, 2, 3 \ldots\}$ 是弧权函数。
- $M_0: P \rightarrow N$ 是一个称为初始标识的函数，其中每个元素 $M_0(p)$ 表示库所 p 初始时有 N 个**令牌**[20]，其中 N 是非负整数集。
- 对于每个变迁 $t \in T$，记作 $\bullet t$ 的输入库所集，是指通过输入弧与 t 相连的库所：

$$\bullet t = \{p_i \mid (p_i, t) \in F\} \tag{3.1}$$

- 类似地，对于每个变迁 $t \in T$，输出库所集合记为$t\bullet$，即通过出弧与 t 相连的库所：

$$t\bullet = \{p_i \mid (t, p_i) \in F\} \tag{3.2}$$

[19]使得 $P \cap T = \varnothing$（即 P 和 T 是互不相交的集合），且 $P \cup T \neq \varnothing$（即 P 和 T 都不是孤立的）。此外，弧可以从库所连接到变迁（输入弧），也可以从变迁连接到库所（输出弧），但不能连接到同类型的节点。[20] 在经典Petri网中，令牌用黑点表示。令牌被分配到 Petri网的库所中，也可以认为它们驻留在这些库所中。

定义：标识

标识是对 PN 中各库所分配令牌的一种方式。令牌的数量和位置定义了系统状态，并且令牌移动时，系统状态可能随之改变。由于变迁射击而导致的令牌移动会引起 PN 的执行 [26]。标识 M 可定义为一个 n 维向量，$M = (m_0, m_1, m_2 \ldots m_n)$，其中 $n = |P|$（库所数量），且每个 $m_n \in N$，$i = 1 \ldots n$。向量 M 给出了 PN 中每个库所 p_i 所包含的令牌数量。

定义：Petri网状态空间

Petri网模型的状态由其标识定义。变迁的发射表示网中标识的变化。具有 n 个库所的 Petri 网的状态空间是所有标识的集合。本章后面将详细讨论状态空间。

定义：变迁使能

给定 PN 中的变迁 t 称为在标识 M_i **当且仅当**对每个输入库所 $p \in \bullet t$，其标识都大于或等于从该库所指向 t 的弧的权重时使能或可发射（或者 t 没有输入库所）。用数学形式表示，变迁 t 可发射**当且仅当**

$$\forall p \in \bullet t \mid M(p) \geq W(p, t) \vee \bullet t = \varnothing \tag{3.3}$$

定义：变迁发射

如果变迁 t 已使能，就可以发射：从每个输入库所 p 中移除 W(p, t) 个令牌，并向每个输出库所 p’ 放入 W(t, p’) 个令牌，由此生成一个新标识 M_{n+1}。

$$M_{n+1} \xrightarrow{t} M_n \mid M(p') = M(p) - W(p, t) + W(t, p') \ \forall p \in P \tag{3.4}$$

M_{n+1} 可从 M_n立即到达。如果通过发射使能变迁序列$\sigma = t_1, t_2 \ldots t_k$，使 M_0 到达 M_n，则称 M_n 可从 M_0 到达，记作 $M_0 \xrightarrow{\sigma} M_n$

示例[21]

考虑图**图12**所示的 PN 模型 PN = 〈P, T, F, W, M_0〉，其中：$P = \{p_1, p_2, p_3, p_4, p_5\}$，$T = \{t_1, t_2, t_3, t_4\}$。设 W = 1 对所有弧，初始标识为 $M_0 = [1\ 0\ 0\ 0\ 0]$

[21] 此示例的灵感来源于 [**68**]

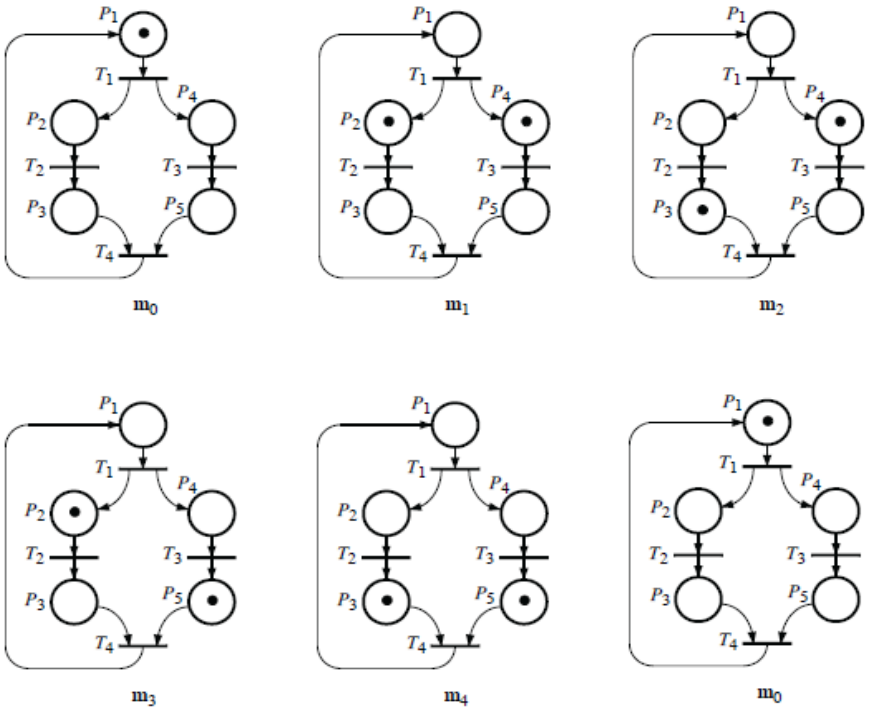


图12：变迁触发序列 （引自 [68]）

σ1: $M_0 = [1\ 0\ 0\ 0\ 0] \xrightarrow{T1} M_1 = [0\ 1\ 0\ 1\ 0] \xrightarrow{T2} M_2\ [0\ 0\ 1\ 1\ 0] \xrightarrow{T3} M_4\ [0\ 0\ 1\ 0\ 1] \xrightarrow{T4} M_4\ [1\ 0\ 0\ 0\ 0]$

σ2: $M_0 = [1\ 0\ 0\ 0\ 0] \xrightarrow{T1} M_1 = [0\ 1\ 0\ 1\ 0] \xrightarrow{T3} M_3\ [0\ 1\ 0\ 0\ 0] \xrightarrow{T2} M_4\ [0\ 0\ 1\ 0\ 1] \xrightarrow{T4} M_4\ [1\ 0\ 0\ 0\ 0]$

在此示例中，有两种可能的变迁发射序列σ1= T1、T2、T3、T4 和 σ2 = T1、T3、T2、T4

3.1.3 PN的性质

与其他模型一样，Petri网是根据非形式化的需求规范构建的。这并非一项简单的任务，需要大量建模经验。如果所建模的系统非常复杂，Petri网模型可能与其原始规范大相径庭。只有当模型相对于其规范在逻辑上正确时，它才有用 [69]。正确性有多种不同的概念。当规范与实现等价，或者系统满足一组期望属性时，我们就称该系统是正确的。这些期望属性使系统设计者能够判断系统是否正确 [69]。

在 PN文献中，“Petri网基本属性集”是指一组与复杂系统的逻辑结构和行为相关的、经常出现的问题或关键问题所对应的属性，因此可将其分为两大类，即（i）结构属性和（ii）行为属性。需要注意的是，这些属性的满足能够回答许多关于

系统正确性的问题，因此有助于 PN 模型的分析。下面以非形式化的方式列出并简要讨论其中一些选定的 Petri网行为属性[22] 。

可达性

可达性是研究系统动态行为的一项基本性质。在Petri网中，研究可达性性质是为了分析某个特定系统状态（以标识表示）是否能够到达。若存在一个变迁发射序列能够将初始标识 M_0 转换为 Mn，则称标识 Mn 可从初始标识 M_0 到达。在可达性分析中，所有可能从 M_0 出发的发射序列都会汇集在可达图$R(N, M_0)$中，而Petri网的可达性问题就是判断给定标识 $Mn \in R(N, M_0)$ [70]是否可达。

有界性

在经典系统理论中，允许无限增长的状态变量通常是系统不稳定的指标[25]。因此，系统具有有界性是理想的。如果对于任意可从初始标识到达的标识，每个库所中的令牌数都不超过有限数 k，即对于每个库所 p 以及每个标识 $Mn \in R(N, M_0)$ [70]，均有 $M(p) \le k$，则称Petri网是有界的（或 k-有界的）。

无死锁性与活性

如果从任何可达标识出发，始终至少有一个变迁可以发生，则称 PN 是无死锁的。活性是比无死锁性更强的条件。如果某个变迁在所有可达标识下都有可能发生，则称该变迁是活的。换言之，如果某个变迁永远不会失去发生的可能性，则称该变迁是活的。如果网中的所有变迁都是活的，则称该网是活的 [69]。

可逆性

如果从每个标识 Mn 出发，都能到达初始标识 M_0 ，则称 PN 是可逆的。因此，在可逆网中，总能回到初始标识或状态 [70]。

公平性

文献中对公平性有不同的含义和理解。具体而言，公平性意味着给予若干竞争者相同数量的机会，使任何一方都不能连续进行超过“k 次”，而不让其他方轮到。在 PN 中，如果变迁 t_1 和 t_2 中任一个在另一个不发生时能够发生的最大次数有界，则称它们之间存在有界公平（或 B-公平）关系。如果网中的每一对变迁之间都存在 B-公平关系，则称该 PN 为 B-公平网 [70]。

互斥

此性质描述了诸如两个或更多进程无法同时访问临界区（资源）之类的约束。在 Petri网中，互斥可以用库所或变迁来定义。若库所 p 和 q 的令牌数在同一标识中不能同时为正，即 {v1}m {v2}RS m(p)·m(q) {v3}，则称它们在 Petri网中互斥

[22] 文献中对这些属性进行了详细讨论，并给出了数学定义和证明 [70]。本章仅将这些属性作为背景概念进行讨论。论文后续部分会用到其中的一些属性。

。类似地，若 Petri网中的两个变迁在任何标识下都不能同时使能 [71]，则称它们互斥。

以下定义了 PN 的一些重要结构属性：

可控性：
如果任意一个标识都能从任何其他标识到达，则称 PN 是完全可控的 [70]。

守恒性：
如果存在一个正整数 y(p)，对于每个库所 p，令牌的加权和在每个标识下均为常数，则称 PN N 是（部分）守恒的。给定一个 Petri网模型，我们通常需要确保其相对于某些权重保持守恒，以表示资源不会损失或增加。

持久性
如果对于任意两个使能变迁，一个变迁的开火不会使另一个变迁失能，则称 PN 具有持久性。在持久网中，变迁一旦使能，就会一直保持使能状态，直到它开火 [25]。

3.1.4 PN分析

PN 的主要优势在于能够对具有并发性的系统进行建模。然而，仅仅建模本身并没有多大用处，还必须能够分析所建模的系统。分析有助于深入了解所建模系统的结构和行为 [26]。目前有许多用于分析 PN 模型的技术，可根据模型的性质用于验证。每种技术也可能有不同的变体。本节将讨论两种最常用的 PN 模型分析技术：

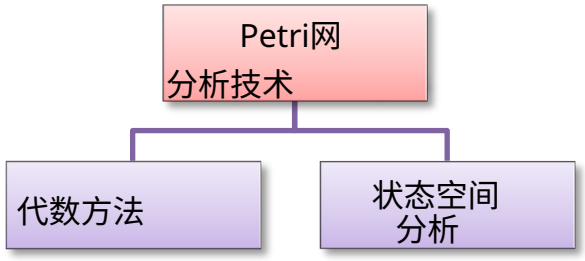


图 13：Petri网分析技术

这些技术为验证上一节提到的属性提供了解决方案和机制。在本论文中，我们选用这些技术进行可组合性验证，并在第二部分通过恰当的示例展示其应用。本章将对这些技术及其优势和局限性进行简要说明和讨论。

代数方法

这种技术也称为线性代数技术（由于大量使用不变量，也称为线性不变量）。在运用代数技术对 Petri 网进行推理的框架中，求解 Petri 网问题可归结为求解与 Petri 网 [24] 相关的代数方程。由于这种技术的特性，该方法通常效率较高（在大多数情况下，复杂度关于 Petri 网的规模为多项式级）。Petri 网模型的动态行为可以用代数方程描述。要使用代数方法，需要应用以下基本概念:

矩阵定义形式 (MDF)

Petri网模型具有矩阵定义形式 (MDF)，由三个 $n \times m$[23] 矩阵组成:

(i) 输出矩阵 A+

$$A^+ = [\boldsymbol{a}_{ij}^+]^{n \times m}, \text{ where } \boldsymbol{a}_{ij}^+ = w(t_i, p_j); \text{ if } p_j \in t_i\bullet, \text{ and } i \in n; j \in m \quad (3.5)$$

也就是说，如果$\mathbf{p}_j$ 连接到$\mathbf{t}_i$ 的输出端，那么$\boldsymbol{a}_{ij}^+$ 等于输出弧的权重；否则为 0[70]。

(ii) 输入矩阵 A-

$$A^- = [\boldsymbol{a}_{ij}^-]^{n \times m}, \text{ 其中 } \boldsymbol{a}_{ij}^- = w(p_j, t_i); \text{ 如果 } p_j \in \bullet t_i \quad (3.6)$$

也就是说，如果$\mathbf{p}_j$ 连接到$\mathbf{t}_i$ 的输入端，那么$\boldsymbol{a}_{ij}^-$ 等于输出弧的权重；否则为 0[70]。

(iii) 关联矩阵 A

$$A = A^+ - A^-, \text{ 其中 } [\boldsymbol{a}_{ij}] = [\boldsymbol{a}_{ij}^+ - \boldsymbol{a}_{ij}^-] \quad (3.7)$$

在关联矩阵 A 中，每个元素 a_{ij} 表示变迁 i 开火一次时库所 j 中令牌数量的变化[70]。

发射次数向量

标识 M_k 是一个 $m \times 1$ 维列向量。M_k 的第 j^{th} 个分量表示某个发射序列中第 k^{th} 次射击后库所 j 中的令牌数。非负整数构成的$\mathbf{n \times 1}$ 维列向量$\mathbf{X}$ 称为发射次数向量，其中$\mathbf{X}$ 的第 i^{th} 个分量表示要将$\mathbf{M}_{k-1}$ 转换为$\mathbf{M}_k$ [70]，变迁$\mathbf{t}$ 必须开火的次数。

状态方程

Petri网的状态方程写作:

$$M_k = M_{k-1} + A.X \quad (3.7)$$

其中:
$\mathbf{M}_{k-1}$ 是当前标识

[23] $n \times m$ 表示 n 个变迁和 m 个库所。

$\mathbf{M}_k$ 是新的标识**A** 是发生
矩阵**X** 是发射次数向量

示例
图14展示了一个生产者-消费者Petri网模型的示例。

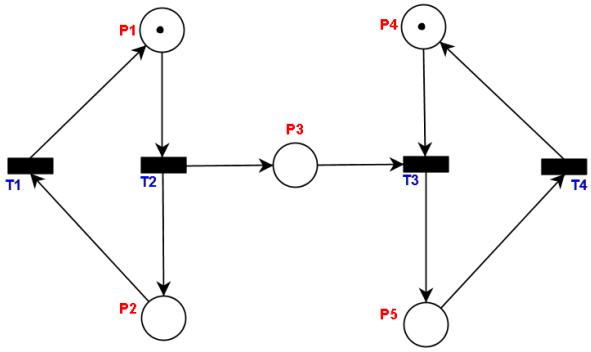


图14：生产者-消费者示例

使用公式3.7，可计算出该模型的关联矩阵A，如下所示：

A+	T1	T2	T3	T4
P1	1	0	0	0
P2	0	1	0	0
P3	0	1	0	0
P4	0	0	0	1
P5	0	0	1	0

-

A-	T1	T2	T3	T4
P1	0	1	0	0
P2	1	0	0	0
P3	0	0	1	0
P4	0	0	1	0
P5	0	0	0	1

=

A	T1	T2	T3	T4
P1	1	-1	0	0
P2	-1	1	0	0
P3	0	1	-1	0
P4	0	0	-1	1
P5	0	0	1	-1

表4：关联矩阵 A

在此模型中，初始标识为 [1 0 0 1 0]。按照发射序列 σ = t2, t1, t2，发射次数向量将为 [1 2 0 0]。利用状态方程，可按如下方式生成标识 M_x：

M0	
P1	1
P2	0
P3	0
P4	1
P5	0

+

A	T1	T2	T3	T4
P1	1	-1	0	0
P2	-1	1	0	0
P3	0	1	-1	0
P4	0	0	-1	1
P5	0	0	1	-1

·

X	
T1	1
T2	2
T3	0
T4	0

=

Mx	
P1	0
P2	1
P3	2
P4	1
P5	0

表5：状态方程

图15以图形方式说明，σ = t2、t1、t2 的发射序列如何使 M_0 转变为 M_3。绿色部分表示变迁发射。可以看到，右下角的标识 M_3 与**表5**中通过矩阵状态方程生成的标识相符。

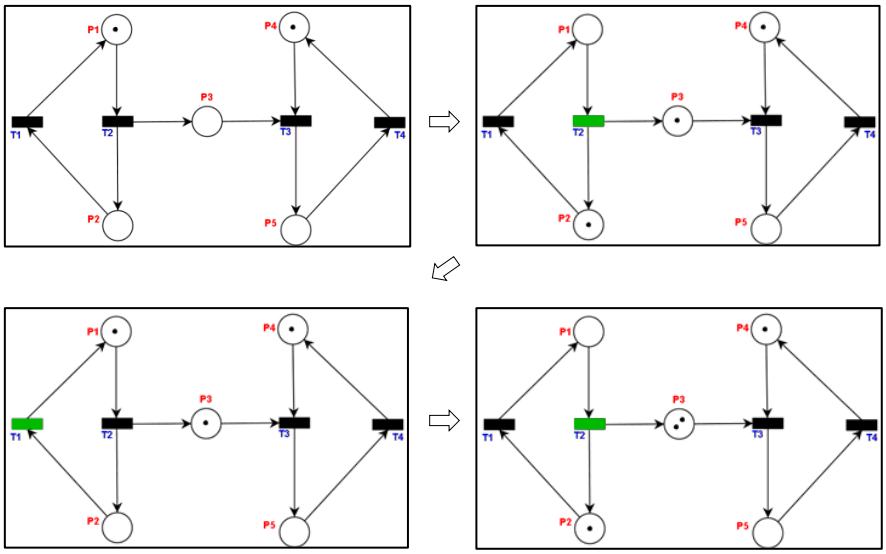


图15：M_0 通过发射序列σ = t2, t1, t2到达M_3

状态方程本身只能用于代数计算未来的标识。要对模型进行代数分析，还需使用其他概念，例如 Petri网不变量。

Petri网不变量

变迁的发生会改变网中的令牌分布，但通常会遵循标识的一些全局属性，这些属性称为线性不变律。对于分析Petri网的结构属性和行为属性，不变量非常有用。从初始标识出发，Petri网的标识可以通过变迁的射击而演化（如果不存在死锁），射击次数可以无限。然而，并非任意标识都可达；所有可达标识都有一些共同属性；在变迁射击过程中保持不变的属性称为不变量。同样，并非任意变迁序列都可以射击；可能的发射序列也具有一些共同的不变属性。因此，无论系统如何演化，不变量都能用于描述可达标识和可发射变迁的某些属性。

图 16展示了一年中不同季节的 Petri网模型。可以看出，无论季节如何变化，4 个库所中始终只有一个令牌。因此，在任何时刻，$M(p_1) + M(p_2) + M(p_3) + M(p_4) = 1$。这一不变量性质的含义显而易见：任何时刻都只有一个季节 [68]。这也意味着该网是结构有界的。

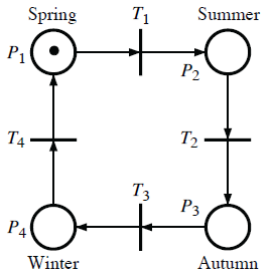


图16：一年中的季节（摘自 [68]）

Petri网有两种重要的不变量类型:

P-不变量

库所不变量用于形式化描述 PN 中与库所有关的不变属性。例如，如果某个库所集合中的令牌总数不受任何射击影响而保持不变，那么该集合就可以定义一个库所不变量。它们有助于评估 PN 的结构属性。简而言之，属于 P-不变量的库所有界 [24], [70]。

如果 PN 中存在 P-不变量，则

$$\sum_{p=1}^{n} m \,.\, y_p = \sum_{p=1}^{n} m_0 \,.\, y_p \quad \forall \mathrm{m} \in \mathrm{R}(\mathrm{N}, \mathrm{m}_0) \tag{3.8}$$

其中，y 是一个由整数构成的 $\mathrm{m} \times 1$ 列向量，满足 $\exists\, \mathrm{y} = (\mathrm{y}_1, \mathrm{y}_2 \ldots \mathrm{y}_n) > 0$ 即至少有一个正的非零元素 [71]。这意味着任何变迁的射击都不会改变 PN 中令牌的加权和。更一般地，如果向量 y 满足

$$\mathrm{A} \,.\, \mathrm{y} = 0$$

很容易看出，如果对所有 $\mathrm{p} \in \mathrm{P}$ 都存在一个 P-不变量，那么就能保证该 Petri 网具有结构有界性。因此，库所不变量可用于推理结构有界性 [24]。如果 P-不变量的每个元素都非负，则它是一个 P半流 [67]。

T-不变量

另一方面，变迁不变量对适用于 Petri 网的变迁发射序列的属性进行了形式化描述。它们有助于评估活性和公平性等行为属性 [24], [70]。

一个 $\mathrm{n} \times 1$ 维发射次数向量 X，若满足以下条件，则称为 T-不变量:

$$\mathrm{A} \,.\, \mathrm{X} = 0$$

也就是说，按照 X 指定的次数触发每个变迁后，Petri 网会回到其初始标识 M_0 [24]。如果 J 的每个元素都非负，则 T-不变量是一个 T半流 [67]。

如果不存在其他 T-不变量 X′，使得对所有 $\mathrm{i} \in \mathrm{T}$ 都有 $\mathrm{x'}_i \leq \mathrm{x}_i$，则称 T-不变量 X 为最小 T-不变量。一个 Petri 网可以有多个 T-不变量。最小 T-不变量称为该网的再生向量 。

P-不变量与 T-不变量的本质区别在于：如果 Petri网中的所有库所都被 P-不变量覆盖，这就是有界性的充分条件；而 T-不变量的存在仅是 Petri网模型能够返回初始状态的必要条件，因为无法保证计数向量等于 T-不变量的变迁序列实际上能够被触发 [71]。

优缺点

代数分析的优势在于，网结构的规模远小于可达标识的数量，因此不存在状态空间爆炸的风险。因此，可以使用线性代数方法证明 PN 的各种性质。然而，这种方法的弱点是，它只能处理有限种性质，并且只能给出充分条件或必要条件。此外，这种方法

涉及复杂的底层数学定理，而针对不同性质的验证所需的定理各不相同，因此无法将其推广到自动化推理中。

状态空间分析

状态空间分析是开展形式化分析与验证最重要的方法之一。与代数技术相比，这是一种相对简单的模型行为分析方法。该方法的基本思路是计算系统所有可能的状态，以及导致状态变化的事件，并将它们表示为有向图。图构建完成后，可以应用不同的搜索技术来分析模型。

在 Petri 网术语中，这种方法通常也称为可达图分析。Petri网模型的状态空间分析通过穷举生成给定初始标识所能到达的所有可达标识，然后检查可达图的结构，以推理模型的 PN 属性。

可达图由顶点和弧组成，其中顶点对应可达标识，弧对应变迁的射击，表示从一个标识到另一个标识的变化。图17展示了一个简单的可达图示例。

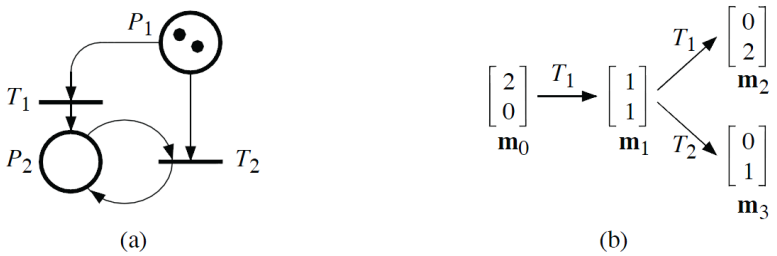


图17：(a) Petri网模型 (b) 可达图 （获取自 [68]）

在某些情况下，如果 Petri网或其某些部分具有重复性且网是无界的，换句话说，Petri网拥有无限多个可达标识，那么可达图的构建就会变得无限。因此，不再无限地持续构建图中的节点，而是采用一种替代技术：通过抽象掉某些细节并插入符号 ω （“无穷大”符号），来表示无界库所的标识。这种图称为覆盖图。生产者-消费者Petri网模型的覆盖图如 **图18 所示**

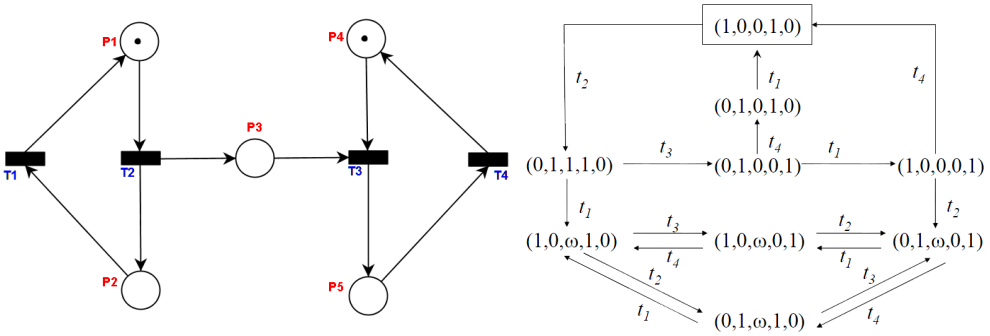


图18：生产者-消费者Petri网模型及其覆盖图

可以看出，库所 P3 为无界的标识包含 ω 个符号 .

构建的状态空间有助于回答有关模型结构和行为的大量分析问题，例如验证无死锁、无活锁；确认活性、能否到达良好状态、无法到达不良状态，以及确保实现目标。以下是状态空间分析如何帮助进行模型验证的一些示例：

有界性

有界性问题可以通过可覆盖树轻松解决，前提是：如果符号 ω 从未出现在 PN 的可覆盖树中，则该 PN 有界。由于 ω 表示某个库所中有无限多个令牌，因此它的缺失可以保证 PN 在结构上有界 [25]。

无死锁性

如果图中不存在这样的节点（非终止节点）：它没有出弧，意味着不存在进一步使能的变迁，那么死锁自由问题就得到了解决。存在一个或多个这样的节点，表明模型可能发生死锁，也有助于找出死锁的确切原因。

无活锁

同样，可以使用状态空间分析检测活锁。对于并发系统，某个过程需要执行一些特定动作 [72]。这些动作通常旨在推动进程，因此称为<风格 id='2'>进展动作</风格>。如果可达图中存在一个环，且其中没有执行任何进展动作，就可以检测到活锁。

状态可达性

可以使用状态空间分析来保证良好状态（或不良状态）的可达性。如果存在一个有效发射序列，能够从初始标识到达某个状态，则该状态是可达的。（在图中，这意味着从初始节点到目标状态对应的节点存在一条路径。）图中可能有多条路径可以到达目标状态。最短路径分析有助于分析到达该状态所需的最少步数。

关于如何进行状态空间分析，建议感兴趣的读者参阅一篇内容详尽的 PN 状态空间分析分步教程 [73]。

优缺点

状态空间方法的主要优势在于，它能够探索系统的所有可能状态。此外，它还能提供反例，说明预期性质为何不成立。再者，状态空间的自动计算与生成提升了易用性，因为计算机工具隐藏了底层复杂数学中的很大一部分，用户只需构造待研究的性质以及用于评估该性质的合适查询函数 [74]。

使用状态空间的主要劣势是状态爆炸问题。从计算角度来看，构建可达图的成本非常高，计算负担也很重。这是因为状态空间的规模可能会随着 Petri 网模型的规模（例如，以库所数量衡量）呈指数级增长。即使是相对较小的系统，也可能具有数量庞大到难以想象甚至无限多个可达状态。当模型包含时间时，这一问题会变得更加严重。人们投入了大量精力开发约简方法，以缓解这一问题。约简方法以紧凑的形式表示状态空间。约简不应影响系统的性质；这些性质应得到保留，并且仍可从约简后的状态空间中推导出来。然而，由于验证领域的复杂性和多样性，不存在一种能在所有情况下都表现良好的单一约简方法。因此，约简方法的选择完全取决于待验证系统的性质 [74]。一些重要的约简方法包括扫描线方法 [75], 哈希压缩法 [76], 对称性方法 [77] 和等价方法 [78]。

在本文中，我们提出了另一种适合我们需求（可组合性验证）的约简方法。如果所研究的模型规模较大且资源消耗较高，该方法有助于缓解状态爆炸问题。

3.1.5 PN类别

基本Petri网或经典Petri网的计算能力较弱。已有研究表明，Petri网的表达力不如图灵机，因此不适合对某些现实世界系统进行建模。为克服这一缺陷，人们引入了许多扩展Petri网，以增强Petri网的表达能力。Petri网有多种分类方式。从结构角度看，它们可分为三个主要类别 [79]:

一级Petri网： 其特征是包含“布尔令牌”，即库所中至多标记一个非结构化令牌。

二级 Petri 网： 其特点是使用“整数令牌”，即库所中标记着多个无结构令牌——它们表示计数器。

三级 Petri 网： 其特点是使用高级令牌，即库所中标记着附带信息的结构化令牌。

Petri 网形式化方法有许多扩展。本节仅讨论本文中使用的部分 Petri 网扩展。

彩色佩特里网（CPN）

CPN 是 PN 的三级扩展，其中库所由表示数据结构的令牌标记。CPN 是一种用于构建并发系统模型并分析其属性的图形语言。CPN 是一种通用离散事件语言，它结合了 PN 的功能（PN 是图形表示法的基础）和一种编程语言（CPN ML）的功能；CPN ML 基于 Standard ML [80] 函数式编程语言，为数据类型的定义和数据操作例程的指定提供基本构件 [78]。

CPN 通过元组 [81]进行形式化定义：

CPN = (P, T, A, Σ, V, C, G, E, I) where:

P 是由有限个库所组成的集合

T 是由有限个变迁组成的集合，其中：$P \cap T = \varnothing$

A ⊆ P×T ∪T ×P 是有向弧的集合。

Σ 是由有限个非空颜色集组成的集合。

V 是一个有限的类型化变量集，满足：对于所有变量 v∈V，Type[v] ∈ Σ

C: P→Σ 是一个颜色集函数，为每个库所分配一个颜色集。

G: T → 表达式是一个守卫函数，为每个变迁 t 分配一个守卫

E: A→ 表达式是一个弧表达式函数，为每条弧 a 分配一个弧表达式

I: P → 表达式是一个初始化函数，为每个库所 p 分配一个初始化表达式。

普通 Petri 网中的令牌没有类型。使用 CPN 可以利用数据类型和复杂的数据操作来定义令牌，即每个令牌都附有一个称为令牌颜色的数据值。发生的变迁可以检查并修改令牌颜色[81]。

“CPN Tools”是一款用于编辑、仿真、状态空间分析和性能分析 CPN 模型的软件包 [82]。该工具作为集成开发环境（IDE），用于构建 CPN 模型。它提供画布以创建 Petri 网图，并具备编写 CPN ML 代码及增量语法检查功能。它还内置了仿真器，能够高效执行非定时网和定时网。从我们的角度来看，CPN 工具最重要的功能是生成和分析状态空间。状态空间分析包括各种内置的状态空间查询函数，并支持生成分析报告，这些功能共同对验证过程做出了重要贡献。有关 CPN 形式化体系及其应用的更多详情，请参阅 [78], [81] 。

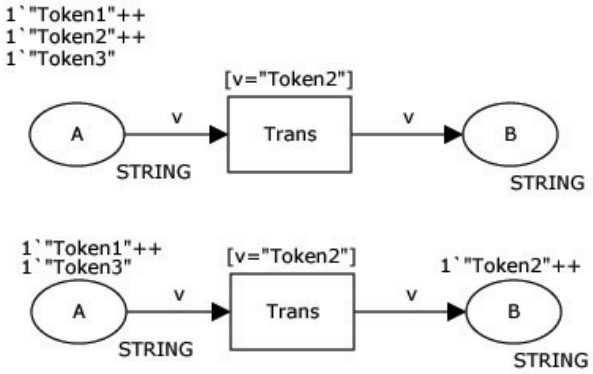


图 19：一个 CPN 模型

图 19 展示了一个 CPN模型的基本示例。椭圆形节点 A 和 B 表示库所。该库所初始化时包含三个字符串类型的令牌。矩形节点表示变迁。输入弧将库所 A 与变迁相连，并带有一个类型为字符串的弧变量 v（用于承载相同类型的令牌）。

类似地，输出弧将变迁与库所 B 相连。该变迁带有一个守卫表达式，用于检查令牌的值。只有当表达式为真时，变迁才能开火。图 19 的第二部分展示了变迁开火的结果，即令牌“Token2”被放入库所 B。

分层 CPN

CPN模型可以组织为一组模块；这些模块可以视为黑盒，使我们能够在不同的抽象层次上工作，并一次专注于一个层次。

替代变迁

CPN工具提供了构建分层 CPN 模型的功能。在分层网中，一个变迁可以代表一整段网结构。这种变迁称为替代变迁 [82]。

子页面／超页面

包含替代变迁的页面称为超页面。当 CP 网使用替代变迁时，该变迁所代表的逻辑会保留在一个称为子页面的页面上 [82]。

端口和插口

超页面和子页面通过端口和插口连接。插口是超页面中的一个库所，且替代变迁与该插口之间至少有一条弧。另一方面，端口是子页面中的一个库所，并标记有以下端口类型标签之一：(i) 输入端口 (ii) 输出端口或 (iii) 输入/输出端口。它通过端口与插口分配与主页面中的插口绑定。此关系用于定义子页面应如何与其超页面的周围环境连接。以下是部分分配规则：

- 带有输入标签的端口必须分配给一个插口，该插口是替代变迁的输入弧。
- 输出标签表示该端口必须与一个作为输出弧的插口关联，
- 输入/输出标记表示该端口必须同时是输入弧和输出弧 [82]。

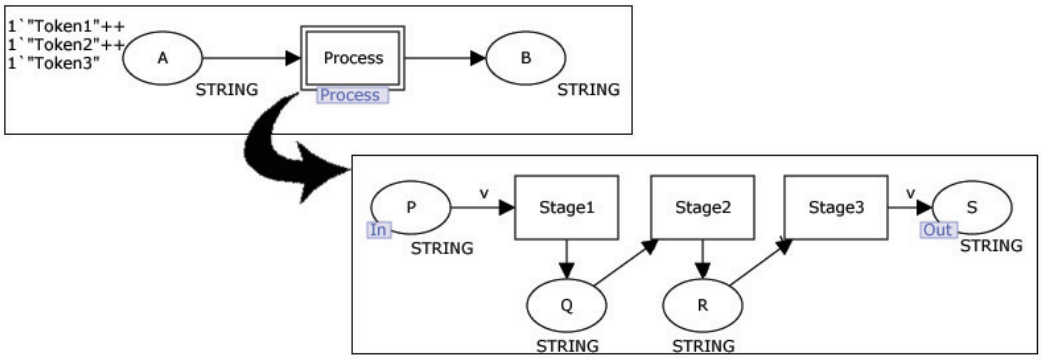


图20：分层彩色Petri网

图20展示了一个分层 CP 网的示例。在超页面（上方）中，显示了一个名为 Process 的替代变迁，它代表一个子模块（下方）。A

过程有三个阶段，以及一个标记为输入端口 In 和输出端口 Out 的输入和输出，它们与超页面中的 A 和 B 套接字库所相连。

时间Petri网

带有时间依赖关系的Petri网可根据时间约束的指定方式进行分类。这些约束可以是时间区间或单个数值，也可以与这些约束相关联的网元素，即库所、变迁或弧 [83]。下一个分类标准是对时间约束的解释。当约束与变迁相关联时，可以将其视为

(i) 发射时间

变迁在变为使能时消耗输入令牌，但只有在与其相关联的延迟时间经过后才会生成输出令牌 [83]。

（ii）保持时间

变迁开火时，令牌的移除和创建动作会瞬间完成，但新创建的令牌必须在其输出库所中停留达到创建它们的变迁所指定的持续时间后，才能用于使能新的变迁 [83]。

（iii）使能时间

变迁必须先持续使能指定的一段时间，之后才能开火；令牌会在同一时间区间内被移除和创建 [83]。

高级Petri网也有已知的时间扩展。其中一种是时间彩色Petri网 [78], ，其时间概念基于引入一个用于表示模型时间的全局时钟。令牌带有时间戳，用于描述它们最早可用于触发变迁的模型时间。时间戳会根据与变迁或其输出弧相关联的表达式进行修改。时间区间可以解释为令牌的非活动时段，变迁则按照强最早发射规则 [78] 开火。

形式化地说，时间 Petri网是一个元组：

$$N = (P, T, F, m_0, Eft, Lft)$$

其中：(P, T, F, m_0) 是一个 PN,
Eft = 每个 t 的最早发火时间$\in$T
Lft = 每个 t 的最晚发火时间$\in$T

3.2 通信顺序进程

CSP 是本文为可执行组件建模而选用的第二种形式化方法。CSP 是由 Sir Charles Antony Richard Hoare [84] 开发的一种语言，旨在用于规范系统进程，并对其并发交互进行推理。CSP 的构想源于这样一种研究需求：使用具有所需表达力和代数定律的形式化表示法来研究并发进程。形式化表示法及其相关代数定律使进程模型能够得到控制和分析。它们还支持对进程正确性进行形式化推理，并证明进程之间的等价性。此外，它们还为开发实现这些目的所需的工具提供了充分的理论基础。

3.2.1 基本概念与定义

CSP形式化体系的主要基元是：(i) 进程、(ii) 事件和 (iii) 代数运算符。

过程

在 CSP 术语中，进程是对实体的独立、自包含、模块化描述，也是捕获行为的基本单元。进程具有特定的接口，由用于与环境交互的事件来表示；环境本身也是一个进程，称为系统全集（Σ）。环境可以看作由并发演化的进程构成的系统。在任何一次运行中，进程都会执行一系列事件。进程包含名称、参数列表和决定其计算逻辑的表达式：

$$\text{进程（参数）} = \text{表达式}$$

表达式描述进程的行为，可以表示为某个事件的发生或一系列事件，这一系列事件称为迹。进程在任何有限时间内只能执行有限数量的事件，因此所有迹的长度都是有限的 [85]。

事件

进程行为的基本单位是事件 [85]。事件体现通信或交互。事件是对观察结果的抽象。每个事件都构成进程与其环境之间的一次交互。如果交互没有发生，进程就会被阻塞。事件可以定义为不携带数据，也可以定义为携带具有特定类型的值。进程 P 的所有事件构成的集合称为 P 的字母表（αP）。下面一行描述了一台简单的自动售货机，它每次接收一枚硬币并出一杯咖啡 [84]。

$$\text{VM}() = \text{insert-coin} \rightarrow \text{coffee} \rightarrow \text{VM}();$$

其中，VM() 是一个过程（不带参数），其表达式包含一系列原子事件：投币和取咖啡，随后该过程被自引用（递归）。事件可以采用复合形式书写，即带有参数，如下一行所示：

$$\text{VM}() = \text{insert-coin}.10 \rightarrow \text{coffee}.1 \rightarrow \text{VM}();$$

此外，还可以在事件主体中使用语句块执行数据操作：

$$\text{VM}() = \text{投币}.10\{\text{余额} = \text{余额} + 10\} \rightarrow \text{咖啡}.1\{\text{咖啡}\text{--}\} \rightarrow \text{VM}() \quad ;$$

语句块可以是一个完整的顺序程序，其中包含赋值语句、条件分支语句、for 或 while 循环以及数学函数等。

输入/输出通道

进程也可以通过通道进行通信。通道是一种特殊类型的事件，称为通信事件。通常，通道上的通信是由并行发生的输入和输出产生的。输入通道用符号'?'表示，而输出通道用符号'!'表示。通道参数可以使用以下形式发送或接收：c ! x 或 c ? y

代数运算符

有许多不同且实用的运算符，用于表示进程行为及其组合体的不同概念 [85]。下面介绍其中一些：

- **前缀算子** $a \rightarrow P$

  前缀算子将一个事件与一个过程组合，生成一个新过程。

- **顺序组合** P ; Q

  它按顺序组合两个过程 P 和 Q，即只有当前一个过程终止后，后一个过程才会开始。

- **确定性选择** P □ Q

  确定性（或外部）选择运算符允许在两个组成进程之间进行选择，并允许环境在外部决定该选择。

- **非确定性选择** P ⊓ Q

  非确定性（或内部）选择运算符允许在两个组成进程之间进行选择，但不允许环境控制选择哪一个组成进程。

- **条件选择** if c o n d P e l s e Q

  选择取决于对条件的求值，以决定选择 P 还是 Q。

- **交错** P ||| Q

  交错算子表示进程 P 和 Q 之间完全独立的并发活动，即不存在屏障同步。

- **并行组合** P || Q

  并行组合算子表示 P 和 Q 之间需要组成进程进行屏障同步的并发活动。如果某个事件同时属于 P 和 Q 的字母表，那么只有当两个进程都能够参与该事件时，该事件才能发生。

3.2.2 CSP 分析技术

人们已开发出许多用于分析 CSP 模型的技术。不过，在许多方面，模型检验都已胜过其他技术，并且普遍受到大多数基于 CSP 的建模环境青睐。本节将简要介绍模型检验技术。

模型检验

> “模型检验是一种自动化技术，它以系统的有限状态模型和一个形式化性质为输入，系统地检查该性质是否适用于该模型 [86]”

模型检验方法的兴起与快速发展，是基于模型的软件验证领域的一项重大成就。尤其是计算领域在攻克顺序程序验证的难题后，随即面临验证并发程序这一更加艰巨的挑战 [87]。这类程序的错误追踪日益困难，原因在于系统行为的复杂性不断增加，以及涌现系统状态导致大量行为具有任意性，而普通的测试和调试方法难以应对这些问题。

自 20 世纪 70 年代末以来，模型检验及其他类似的算法和自动机理论方法，源自杰出研究人员的不懈努力。他们率先提出了不同的标准，这些标准共同构成了一套原理基础，塑造了现代模型检验技术 [87]。

模型检验之所以在不同领域取得成功，原因如下：

▪ 与传统测试方法不同，模型检验是一种穷尽式方法，可对系统模型进行深入分析，以确认不存在错误（而非仅通过调试发现少数错误）。▪ 模型检验会给出答案——要么是成功的结果，要么是反例；反例会展示错误的精确迹及其成因 ▪ 。模型检验技术的改进有效降低了状态空间爆炸问题的风险 [87]。▪ 模型检验具有坚实的数学基础，建立在图算法、数据结构和逻辑理论之上 [86]。▪ 模型检验支持对输入模型（如有限状态机、Petri网、CSP 等）以及待验证的系统性质（通常以线性时序逻辑、计算树逻辑或其扩展形式表示）进行形式化规范。因此，任何 3rd 相关领域的群体都可以将模型检验器当作黑盒使用，而无需了解其过程的原理和复杂性。

除了具备诸多优势外，它的一些不足之处包括：

▪ 大多数模型检验器要求模型简化细节，采用紧凑且表达力较弱的状态，并避免指定枚举，以降低状态爆炸的风险。因此，降低系统表达能力可能会增加额外工作，也可能导致忽略重要特性，最终得到不充分的验证结果。

▪ 尽管已经开发出多种非常有效的方法和改进的数据结构来应对状态爆炸问题，但现实系统的模型仍可能过于庞大，以至于无法进行验证。

模型检验的类型

根据状态的枚举方式，模型检验方法分为两种类型：（i）显式式和（ii）基于符号的 [88]。
显式模型检测技术会将已探索的状态存储在哈希表中，其中每个条目对应一个系统状态。即便只有几百个状态，状态空间图中的节点也会多达 ~10^{11} [88]。另一方面，显式模型检验器支持状态枚举，能够详细表达系统状态。

符号模型检测技术使用高效的数据结构（以二元决策图（BDD）等规范结构表示） [89], ，以符号方式存储已探索状态的集合，并通过单步探索一组状态来符号化遍历状态空间。与显式状态枚举技术相比，这些基于 BDD 的方法显著提升了可扩展性；然而，由于符号遍历过程中构造的 BDD 会变得极其庞大，而且 BDD 的大小严重依赖于变量排序，因此其性能会下降。这促使研究出现了一个新趋势：将布尔推理与表示分离。因此，研究人员对布尔可满足性（SAT） [90] 进行了研究和探索，以实现布尔推理和高效的半规范表示，并由此开发出 SAT 求解器。与 BDD 相比，SAT 求解器效率更高，表示也更紧凑。SAT 结合高效表示，已成为模型检验应用中 BDD 的一种可行替代方案 [88]。

有界模型检测是一种模型检验方法，它限制对状态空间进行正向遍历的步数，并检查是否会在 k 步或更少步数内违反某项性质 [88]。该方法会报告“发现违反”或“在有界深度（即 k 步）内不可能发生违反”；通过增加步数，可以进一步查找可能的性质违反。这种方法前景良好，因为它不会导致状态空间爆炸，或者至少能让用户控制发生状态空间爆炸的可能性。

在本文中，这三种模型检验方法均配有用于验证基于 CSP 的模型可组合性的工具。

3.2.3 时序逻辑

逻辑提供了形式语言，其中包含用于表示某个应用领域 [91] 内的陈述及其逻辑推理的公式。通常，逻辑语言由不同符号组成的字母表，以及公式集的定义给出；公式是字母表上的字符串 [91]。在逻辑中，时序逻辑这一术语用于表示和推理按时间限定的命题。时序逻辑在形式化验证中有着重要应用，用于规定系统需求。线性时序逻辑（LTL）和计算树逻辑（CTL）是其两个主要变体。LTL 公式在计算路径上进行解释。设 A 和 B 为原子谓词，

¬ ∧ 、 、 、
∨ ↔ ○ □ ◇
和 True 为经典逻辑的运算符，而 、 、和 U 则是线性时序逻辑的运算符，分别称为下一、始终、最终和直到。

一些 LTL 陈述的直观含义如下：

- ¬A：A 不成立
- A ∧ B：A 和 B 均成立
- ◯A：A 在下一状态成立
- □A：A 在所有状态下均成立
- ◇A：A 最终成立 • A U B：A 将一直成立，直到 B 成立。

在 CTL 中，还有额外的路径量词“∃”和“∀”，分别表示“存在一条路径”和“对于所有路径”。CTL 公式在计算树上进行解释。对于一棵树，上述公式的直观含义如下：

- ∃◯A：存在一条路径，使得 A 在下一状态成立 • ∀□A：对于所有路径，A 在所有状态中始终成立

3.2.4 时间CSP

CSP 已历经数十年的发展。CSP 的一项重要扩展是加入定时原语，称为 TCSP，以支持时间敏感的进程建模 [92]。在 TCSP 中，每个无时间 CSP 运算符都在带时间语境下进行解释，并添加了两个基本定时运算符：(i) 超时和 (ii) 中断，同时采用牛顿时间假设（即所有进程共用一个全局时钟，且时钟进度速率相同）。

▪ **超时** $P \rhd^{d} Q$超时算子可用于在进程中引入延迟。▪ **定时中断** $P \vartriangle_{e} Q$

如果允许进程运行的时间不超过特定时长，则使用中断。

本论文后续将使用 TCSP 的概念对实时系统进行建模和验证。

3.2.5 概率系统

对于设计随机算法、建模不可靠或不可预测的行为，或规范基于模型的性能评估至关重要的概率特性，具有此类特性的系统称为概率系统 [86]。为对此类系统中的随机现象建模，迁移系统被赋予概率。概率系统可以用不同方式进行规范。两种非常常见的方式是：(i) 马尔可夫链（MC）和 (ii) 马尔可夫决策过程（MPD）。在本文中，我们将 MPD 作为概率系统的规约形式体系，因为它既支持非确定性选择，也支持概率选择；不同于 MC，它还能以恰当的方式对并发进程的交错行为建模 [86]。

马尔可夫决策过程是一个元组 〈S, Act, P, l_{init}, AP, L 〉 [86]。

其中:
S = 状态集
Act = 动作集
P: S × Act × S → [0, 1] 是转移概率函数，使得对于所有状态 s∈S 和动作 α∈ Act:

$$\sum_{s' \in S} P(s, \alpha, s') \in \{0,1\}$$

l_{init}: S → [0, 1] 是初始分布，满足:

$$\sum_{s' \in S} l_{init}(s) = 1$$

AP 是原子命题的集合
L: S → 2^{AP} 是标记函数

本文后续将使用马尔可夫决策过程的概念来对概率系统建模并进行验证。

3.2.6 CSP实现工具

开发 CSP 模型时，可以使用多种实现支持工具和语言，例如 CTJ（Java）、CSP++（C++）、CSP.NET、PyCSP（Python）、JCSP（Java）和 CSP#（C#）[93]。

同样，CSP 分析也有多种技术，例如:

• FDR2 模型检查器由 Formal Systems Europe Ltd 开发[94]。• ARC（阿德莱德精化检查器）是一种 CSP 验证工具[95]。• ProB 是一款动画器和模型检查器，支持 CSP 的精化检查和 LTL 模型检查[96]。• PAT 是一款用于 CSP 的模型检查器、仿真器和精化检查器[97]。

本论文选择 PAT 模型检查器，是因为它提供了便于用户使用的 CSP 模型建模环境、快速的仿真器和模型检查器，最重要的是，它支持实时 CSP、概率 CSP 和实时概率 CSP 等 CSP 扩展。

3.2.7 流程分析工具包（PAT）

PAT 是由新加坡国立大学开发的成熟工具，专用于并发系统验证，并已应用于现实世界的工业项目中。PAT 旨在使用一种称为 CSP-Sharp（或 CSP#[24]）的 CSP 形式化体系扩展来开发、组合、仿真和分析基于事件的系统模型。该扩展包含一些新增功能，例如共享变量和异步消息传递。此外，它支持使用复杂数据类型（例如集、队列和栈）以及以 C# 编写的外部库中的函数，从而能够

[24]它采用类似 C# 的句法来描述 CSP 进程

对复杂的进程行为进行建模。PAT 还支持自动化精化检查，以及对扩展了事件[98]的 LTL 进行模型检验。
PAT 是一个适用于基于 CSP 的进程模型的建模、组合、仿真、验证和推理框架。这些模型可具有不同性质，例如并发系统、实时系统和概率系统。该框架的主要优势在于实现了多种模型检验技术，并为不同属性提供验证支持。这些属性包括无死锁性、无发散性或可达性等一般系统性质，以及用 LTL 断言定义的用户特定属性。它还支持精化检查，以及对实时系统和概率系统进行模型检验。为实现良好的性能，PAT 还实现了先进的优化技术，例如使用 BDD 进行偏序约简、对称性约简和并行模型检查 [97]。

3.3 总结

本章讨论了两种可执行建模形式，即：（i）Petri网和（ii）通信顺序进程，以及与之相关的概念、工具和技术。本论文使用这两种形式化方法来描述可执行模型。要理解本论文后文提出的方法，需要掌握 PN 和 CSP 的概念背景。

第4章 验证与分析

Chapter 4
Verification and Analysis

验证与确认是任何软件工程实践中的重要环节。二者是彼此独立的程序，具有不同的特征，用于检查程序、服务、模型或系统是否正确、是否满足需求规格，以及是否实现了预期目的。它们是实现必要质量保证水平的关键组成部分，也是可信、可靠地使用交付产品的必要前提。本章主要关注验证及其不同的分析技术。本章旨在概述软件验证的基本概念、原理、问题和不同方法。本章可视为一份指南，帮助读者理解本论文后文提出的验证过程。

程序的正确性是一个相对概念，意味着程序所执行的内容不少于其规范所规定的内容 [99]。验证、确认与测试（VVT）相结合，构成了系统工程中一门更为广泛且复杂的学科。在建模与仿真（M&S）中，通常会提及验证、确认与认证（VVA）的组合，其中“认证”是指对模型或仿真是否适用于特定目的进行正式认证 [100]。然而，其目标都是保证产品质量；当系统具有高度关键性时，这种质量保证的驱动力会进一步增强：这可能是因为系统的生产成本极高，例如用于探测外行星的月球车；也可能是因为人类的生命依赖于这些系统，例如控制飞机和汽车的计算机，以及医院中的生命支持实时系统 [101]。这些系统必须具备正确性，因为其失效可能导致人员死亡或巨大的经济损失。此外，正确的系统也可能被错误地使用，从而导致故障。当系统以模块化方式设计，并基于对新环境的假设进行实现时，这便是一个普遍问题。阿丽亚娜-5一次性火箭发射系统的发射失败就是一个类似案例：由于从阿丽亚娜-3中复用了一个软件模块，而其中某些假设并不适用于阿丽亚娜-5，导致后者自毁；其原因仅仅是一个64位浮点值变量被错误地转换为16位整数，进而造成系统崩溃 [102]。因此，对于关键系统而言，确保其正确且不存在错误，是值得付出努力的。

验证与确认旨在通过提供正确性和适用性的证据与指示，提升模型和仿真结果的可信度。具体而言，验证关注从真实系统角度看模型是否正确，而确认则关注模型相对于真实系统是否适用或合适。另一方面，测试旨在揭示系统中的错误。下一节将讨论这些相互关联术语的定义和概念。

4.1 建模与仿真中的一些基本概念

最早开始探讨验证与确认（V&V）方法论和术语的应用技术学科，是运筹学（OR）领域，也称为系统分析或建模与仿真（M&S）领域 [**103**]。

验证
根据国防部国防建模与仿真办公室（DoD）的定义，验证是：确定模型实现是否准确呈现开发者的概念描述和规范的过程 [**104**]。

一般来说，验证是指评估产品是否符合其规格或适用法规的过程。在建模与仿真中，验证通常定义为确定模型是否符合其规范的过程 [**29**]。验证涉及模型正确性，关注的是正确地构建模型 [**28**], ，也就是说，模型能够正常运行且没有错误。原则上，验证关注的是将模型需求转换为概念模型，以及将概念模型转换为可执行模型的准确性 [**29**]。

为清晰起见，正确性的概念定义如下：
正确： 没有错误；准确；符合事实、真理或理性；符合某种方法、惯例或行为公认的标准 [牛津词典]
正确性
程序、模型或整个系统在其规范、设计和实现中不存在缺陷的程度 [**105**]

软件产品（或仿真模型）按照其规范所定义的内容，精确执行任务的能力 [**106**]。

我们将结构和行为符合其规范的组合模型定义为正确模型。因此，组合模型的正确性取决于其规范。软件实体的正确性可呈现为三种状态，即：**(i)** 正确：已根据其规范确认正确；**(ii)** 有缺陷：已根据其规范确认不正确；**(iii)** 未知：尚未根据规范确认其正确性 [**107**]。在软件工程中，软件实体的规范是其所有通过的单元测试之和 [**107**]。我们将规范定义为一组目标（或目标）和属性约束（见 1.3.2），组合模型必须满足这些要求，才能被确认为正确。

验证
根据国防部（DoD）国防建模与仿真办公室的定义，验证是指：从模型预期用途的角度，确定模型在多大程度上准确地代表现实世界的过程 [**104**]。

相反，模型验证关注的是构建正确的模型，即能够准确表征真实系统的模型 [**28**]。模型验证通常被定义为“证实计算机化模型在其适用范围内具有令人满意的准确度，且与模型的预期应用一致

适用性具有令人满意的精度范围，与模型 [108] 的预期应用一致。

测试

另一方面，模型测试用于确定模型中是否存在不准确之处或错误。测试的目标是表明模型（或系统）存在错误（而不是证明其正确）。测试只能发现错误，却无法保证不存在错误；因此，这是一种必要但较为临时且成本低廉的方法，其正确性仅依据所有测试均已通过这一事实来判定，这种做法并不充分，也不可靠。测试失败时，便成功揭示了一个错误；测试通过时，则未能检测出错误。如果多项测试都未能检测出缺陷，即使无法保证正确性，也会提高对系统的信心 [99]。

4.1.1 建模过程中的验证与确认

Sargent [108] 定义了一个建模过程，如图21所示。在此过程中，验证指的是一项活动，用于确保概念模型的计算机编程和实现是正确的。

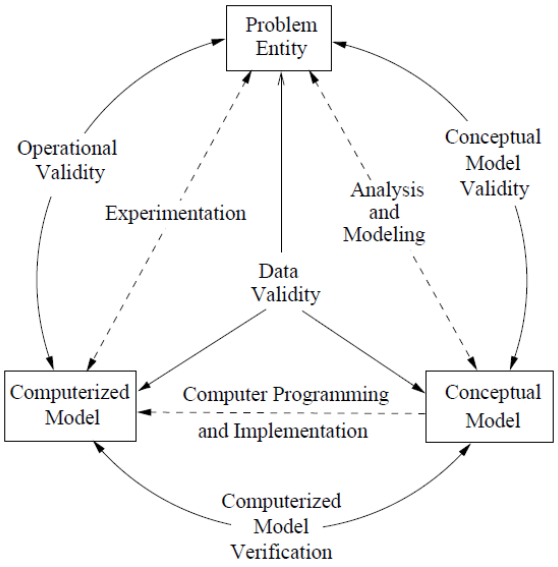


图21：建模过程（摘自 [108]）

验证从三个角度定义：

概念模型有效性 是指确定概念模型所依据的假设是否正确，以及问题实体（仿真对象）的模型表示对于其预期目的是否“合理”。

运行有效性是指确定模型的输出行为对于模型的预期目的是否足够准确。

数据有效性是指确保解决问题所需的模型执行和模型实验数据充分且正确 [108]。

Mike Petty 在他的文章 [29] 中也阐明了这两个术语在模型评估过程不同阶段的区别，如图22所示。

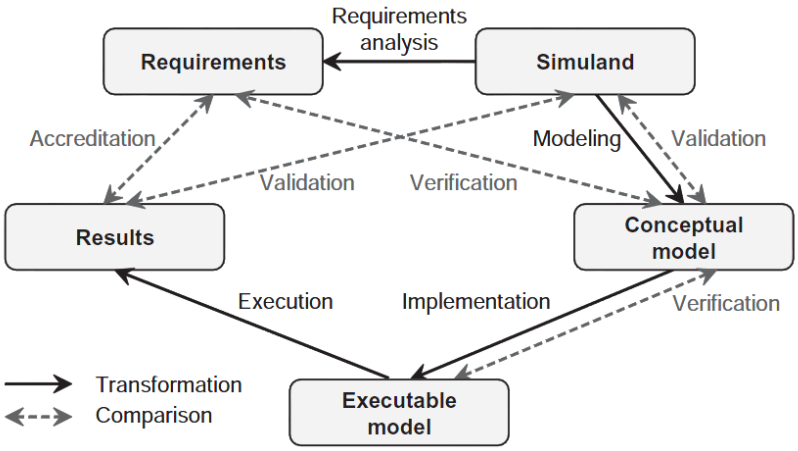


图22：建模过程 （摘自 [29]）

仿真对象是要进行仿真的真实系统，而模型则是对仿真对象的表示。模型以预期应用为导向开发，因此只捕捉仿真对象必要的抽象，省略其他部分。需求由预期应用决定。概念模型记录仿真对象的各个方面，包括结构和行为方面，例如对象、实体、事件、功能、环境现象等。可执行模型是能够执行的计算机程序，旨在按照概念模型中的详细描述仿真该仿真对象。因此，概念模型可视为可执行模型的设计规范。结果是模型在仿真过程中产生的输出。

图22将验证与确认描述为比较两项内容的活动。验证是将需求与概念模型进行比较。在此比较中，验证旨在确定概念模型是否满足给定需求。第二项比较是在概念模型与可执行模型之间进行，目标是确定已实现的可执行模型是否与概念模型一致。确认是将仿真对象与概念模型进行比较，以确定概念模型是否准确描述了仿真对象。第二项比较是在仿真对象与结果之间进行，用于确定仿真的输出相对于仿真对象的实际行为是否足够准确 [29]。

Balci [28] 提出了另一个全面的 VV&T 模型，它以仿真研究生命周期的形式呈现，如 **图23**。所示。各阶段用椭圆符号表示。虚线箭头描述了各阶段之间的关联过程。实线箭头表示可信度评估阶段。生命周期的每个阶段都对应一项 VV&T 活动。问题构建（或问题定义）是构建问题的过程，所构建的问题应有足够明确的定义，以便采取具体的研究行动并调查合适的求解技术。系统调查的输出是系统与目标定义，这进一步有助于模型构建。模型构建是定义概念模型的过程，该模型对所研究的真实系统进行抽象或构想。概念模型进一步以交流模型的形式表示；交流模型是一种模型表示，可与其他设计人员交流，并可与系统及研究目标进行比较。随后，它进一步

通过编程过程转化为可执行模型。实验模型是经过编程的模型，其中包含对操作的可执行描述以及实验设计，用于以特定目的对仿真模型进行实验。实验过程会产生仿真结果，并将其提交给决策者，以供接受和实施；如有需要，也可以进行细化。

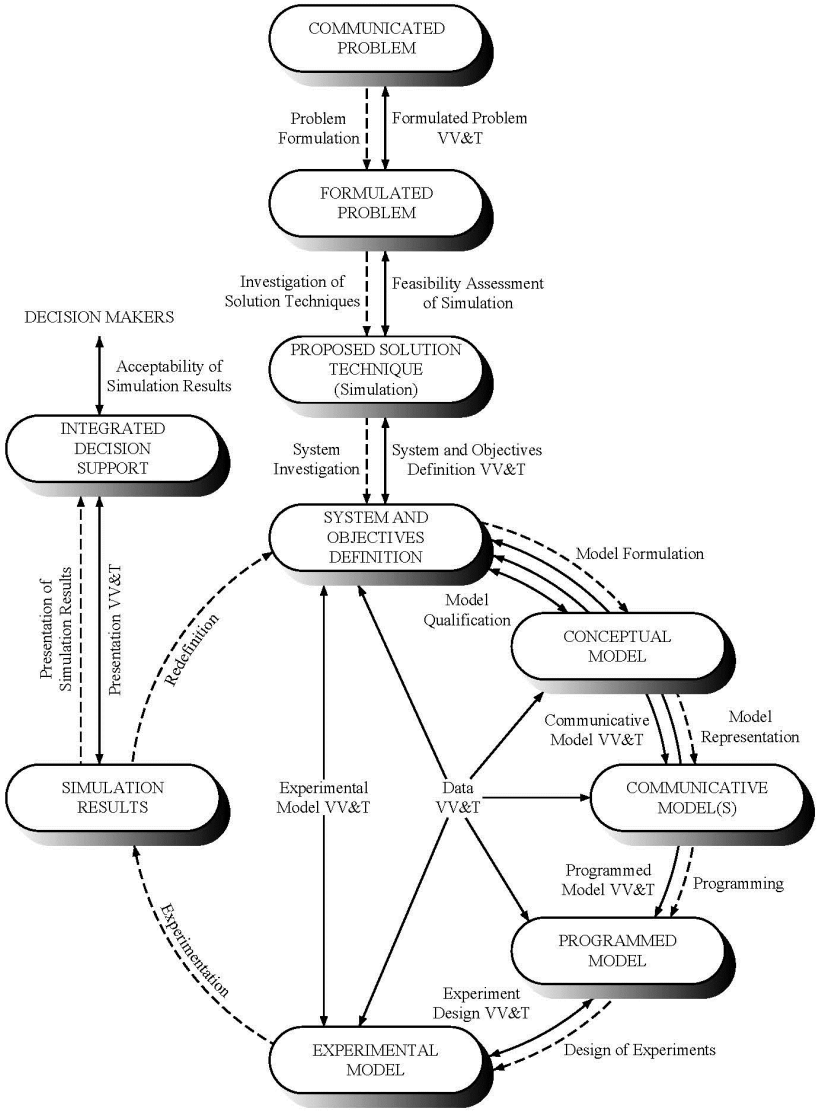


图23：仿真研究生命周期 （引自 [28]）

图21、**图21**、**图22**和**图23**所示的模型评估生命周期被视为指南，并为本文稍后提出和介绍的验证生命周期提供了灵感。

4.2 自顶向下细化的原理

自顶向下细化的原则在模型验证领域备受重视。一次性构建一个满足所有正确性层级的高度详细模型非常困难。相比之下，先构建一个细节较少的抽象模型则容易得多。设 S_1 为初始模型。为了将 S_1 推进到模型的最终形态，自顶向下细化范式主张推导出

S 的模型序列 S_1, S_2...S_f （按顺序排列）。对于 i = 1...f，若满足以下条件，则模型 S_{i+1} 是其直接前驱模型 S_i 的细化：

(i) S_{i+1} 的表达力强于 S_i

(ii) S_{i+1} 的抽象程度低于 S_i

(iii) 基于已验证的 S_i，评估 S_{i+1} 相对容易

因此，细化序列中的最后一个模型应当通过构造保证正确。以下是自顶向下细化范式的一些推论。首先，S_{i+1} 比 S_i 更难理解，因此也更难单独证明；正是细化步骤使得我们能够在假设 S_i 已被证明正确的前提下，验证 S_{i+1} [99]。

在本论文中，提出的验证过程基于这一基本原则：验证以迭代方式进行，且针对模型相对精细的形态开展。

4.3 验证技术

验证方法种类繁多。这种多样性源于仿真项目类型、研究对象（仿真对象）以及数据类型各不相同。大多数验证方法都受到软件工程领域的启发，因为仿真项目中的可执行模型几乎始终以软件形式实现 [29]。

在文献中，验证技术通常分为四大类，如**图24**所示。

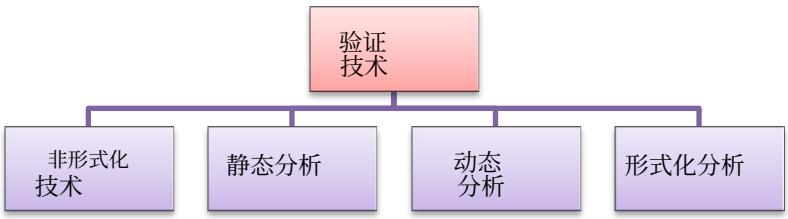


图24：验证技术

4.3.1 非正式技术

这些技术是最常用的。由于所用工具和方法高度依赖人类推理和检查，且没有任何底层数学形式体系，因此称为非形式化技术 [28]。这些技术结构良好，并按照标准政策和程序遵循适当的指南，但它们十分繁琐，效果也不够理想 [109]。

表6列出了几种常用的非形式化方法。

审计	审计旨在评估系统研究是否充分 按照既定计划、政策、程序开展， 标准和指南 [28]。
Desk 检查	桌面检查或自我检查是由个人执行的全面检查 作为第一步。在此方法中，语法 检查、规范比较、代码、控制流图 分析均会执行 [28]。
检查	检查由团队开展，并在不同的 开发阶段进行，例如问题定义、概念 建模、执行等。检查旨在发现并 记录故障 [28]。
图灵测试	图灵测试由被研究系统的领域专家进行 。他们会看到两组输出数据， 一组来自模型，另一组来自规范（ 但不告知哪组对应哪一方），并被要求区分两者， 然后根据他们的反馈修正模型 [28]。

表6：非形式化验证技术

4.3.2 静态分析：

这些技术用于评估静态模型设计和实现（源代码），而无需执行模型。其目的在于检查模型结构、数据流和控制流、句法准确性以及一致性。表 7 列出了一些常用的静态分析方法。

结构 分析	结构分析用于检查模型结构。它通过构建模型的 控制流图来进行 结构分析 [28]。
Data 分析	它涉及数据依赖测试和数据流分析，以确保 模型使用的数据都已正确定义，并且适当的 操作应用于数据对象 [28]。
因果- 效应 绘图	因果图绘制通过以下方式协助评估模型正确性： 在模型表示中回答“什么导致什么”的问题。 其方法是识别模型中的原因和结果，并 检查它们是否在规范 [28] 中得到准确体现。
语法 分析	语法分析通常由所使用的 仿真语言编译器执行。语法分析也可以通过 应用于模型表示的规则集来执行，以 验证其是否满足给定的规范。
语义 分析	此技术用于确定建模者的意图并验证 真实意图是否准确地反映在模型表示中 [28]。

表 7：静态分析技术

4.3.3 动态分析：

动态分析技术基于模型的执行，以评估其行为。它们不仅检查执行的输出，还会在模型执行过程中观察模型。为了在执行期间收集或监控行为，需要在模型中插入称为插桩的额外代码[109]。表8列出了一些重要的动态分析验证技术。

断言检查	断言是在.....期间应当为真的语句。模型执行期间应为真的语句。断言会放置在模型的各个部分，并在执行期间进行监控 [28]。
自底向上检查	此技术与自底向上模型开发策略。子模型会逐个检查。然后检查更高层级的父节点 [28]。
故障/失效注入	此方法用于在模型中注入故障或失效并观察是否产生了预期的错误行为。这种方法能有效检测无法解释的行为，从而发现错误 [28]。
功能测试	此技术用于评估模型输入的准确性输出转换，用于评估模型的准确性将给定输入转换为一组输出数据 [28]。
敏感性分析	通过改变以下各项的值进行敏感性分析。模型输入变量和参数在一定距离范围内的值兴趣 and 正在观察 效果 on 模型行为。意外的效果可能会揭示错误 [28]。

表8：动态分析技术

4.3.4 形式化分析

形式化分析是指通过数学方法分析系统相对于某个明确无歧义的规范或性质是否正确。用于分析的方法称为形式化验证方法，而明确无歧义的规范称为形式化规约。形式化验证可以对系统的抽象模型进行全面验证，该模型可使用有限状态机、Petri网或其他规约形式体系建模。不过应注意，形式化验证只能确保设计相对于其能够证明的某些性质是正确的 [88]。形式化分析技术有很多，我们将其分为四大类：

等价性检查	它也称为参考模型检查，这是一种广泛使用的验证技术，可用于对两个行为模型彼此进行比较。一般而言，二者之一会被视为参考模型，并代表所谓的黄金模型（或完美模型）。它用于验证在所测试的场景中，两个模型的行为是否相同。这种技术存在局限性：它实际上并不能验证设计不存在缺陷，只能提供相对正确性的证明[**109**]。
定理证明	此方法涉及验证数学在设计过程中提出或推导出的定理，并使用形式化规范语言。该过程涉及两个主要组件：**(i)** 证明检查器（在大多数情况下可完全自动化）和 **(ii)** 推理引擎（可能需要偶尔的人工指导）**{v1}**。引擎（可能需要偶尔的人工指导）[**109**]。
属性验证	形式化属性规定了正确性的需求系统设计。此方法的目标是检查实现是否满足这些需求。静态基于断言的验证（ABV）和动态 [**110**]。
模型检验	模型检验为可靠的验证与确认提供了坚实的信心过程。模型检验是一种自动化且全面的验证技术，可用于验证针对给定对象所指定的属性（通常使用时序逻辑描述）对于所有合法的设计输入是否均得到满足。模型检验也存在局限，因为它受到众所周知的状态爆炸问题的影响。在最坏情况下，设计的状态空间可能随着状态变量数量的增加而呈指数级增长。模型检验可以完全自动化地用于设计验证，并且能够得出结果，速度远快于定理证明 [**109**]。

表 9：形式化分析技术

我们提出的验证框架采用了其中一些技术。

4.4 总结

本章讨论验证、确认和测试的不同概念，因为它们共同有助于证明模型的正确性和准确性。本章还讨论了一些现有的模型开发流程（主要由建模与仿真领域制定），因为它们是本文后文提出的验证生命周期的基础。所提出的框架主要关注验证（不过，其设计也允许采用确认技术）。验证技术分为四大类，并简要介绍其中一些选定的技术，因为本文后文将会用到这些技术。

第二部分

Part II

技艺

Techne

技艺 在希腊语中意为工艺、技艺或艺术。在科学领域，它指知识实践；*Techne* 与 *Epistēmē* 相似，都蕴含原理知识，但 *techne* 的不同之处在于，其意在制作或实践，而非“深入理解”；它属于应用科学，关注的是主题的“如何”。

第二部分涵盖了本文所讨论研究的技术，将第一部分提供的理论概念应用于实践，并在由方法、技术、算法和进程构成的综合框架下进行技术性讨论，同时以拟议解决方案的形式阐述其实际意义。

“没有知识，实践便毫无用处；没有实践，知识也毫无用处。”

– Ali bin Usman Hajvery（ Kashaf-Almahjoob）

第5章：所提出的方法论与验证框架

本章阐述了本论文提出的解决方案框架的核心内容。本章介绍了一系列方法、技术、算法、子过程、活动和途径，作为解决基于*BOM*的模型组件可组合性验证中各种问题的拟议解决方案。所有这些贡献都整合到一个统一的框架中，我们将其称为：可组合性验证框架。

所提出的验证框架由不同的方法、技术、算法、子过程、活动和方法构成，它们共同涵盖基于组件的建模与仿真（CBM&S）生命周期。

5.1 基于组件的建模与仿真生命周期

CBM&S 生命周期受到 Sargent、Petty 和 Balci 提出的不同建模架构的启发，这些架构已在第4.1.1节中讨论。我们在其不同阶段加入了所提出的贡献，对其进行了扩展。所提出的 CBM&S 生命周期主要分为四个象限：（i）初始阶段，（ii）建模，（iii）执行，以及（iv）分析。每个象限包含不同阶段，每个阶段又包含多项活动（或活动循环）。每项活动由与相应阶段相关的方法和技术组成。在生命周期中，这些阶段会被反复迭代；每次迭代代表一个层级，因此整个 CBM&S 生命周期是一个多层级过程；每个层级都会使待研究问题的解决方案得到细化，因为该过程遵循自顶向下细化原则，相关内容见第4.2节。CBM&S 生命周期的上述所有特征均展示在 **图 25** 中，并划分为四个象限：

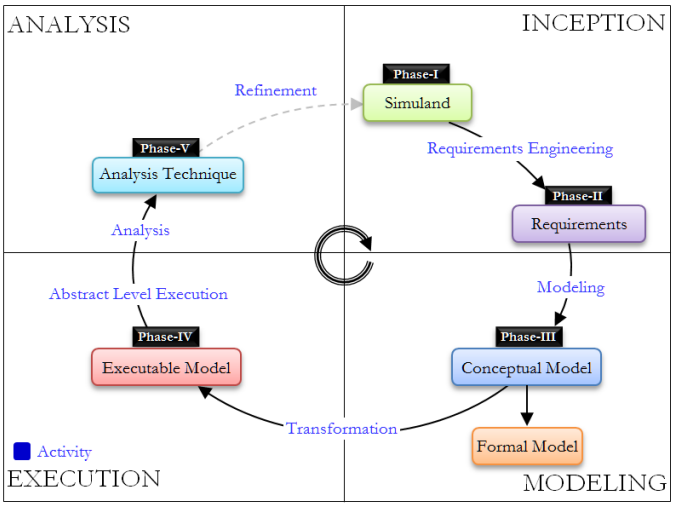


图25：CBM&S 生命周期

以下各小节将逐一详细介绍每个象限及其相关的内部活动、方法和技术。

5.2 初始阶段

CBM&S 生命周期的第一个象限称为“初始阶段”，它开启了整个过程。首先，将真实系统的抽象汇集为仿真对象。仿真对象可以通过 UML 图（**图 26**）输入，也可以采用其他形式化或非形式化表示。

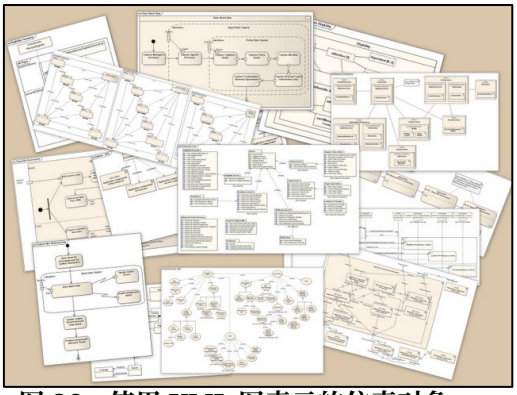


图 26：使用 UML 图表示的仿真对象

基本思路是收集相关知识，以便建模者能够在特定参考框架下构想真实系统，也就是说，明确研究该系统时所处的语境。将仿真对象输入框架后，可用于（i）通过需求工程过程收集需求，以及（ii）从 BOM 仓库中搜索并发现合适的组件，以构建组合模型。如果仓库中不存在所需组件，则从头构建该组件并将其添加到仓库中。需求工程活动的结果是形成需求规格。需求规格形式化方法（如第 1.3.2 节所述）用于表达此框架的形式化需求：

$$RS = \langle O, S \rangle$$

其中

$O = \{o_1, o_2, o_3 \ldots, o_n\}$ 是一组最终必须实现的目标或目的。这些目标通常是在建模领域的场景语境中定义的。因此，以目标或目的形式表达的属性可能特定于某个场景，而非标准的系统性质。例如，在餐馆模型中，目标可能是顾客得到食物供应并收取付款，而不是模型应当不存在死锁（不过这可能是一个必要条件）。

$S = \{s_1, s_2, s_3 \ldots, s_n\}$ 是一组系统约束（系统性质或特定场景的安全性/活性属性）。无死锁性（或其他类似的系统性质）可能是实现上述目标所必需的约束，因此必须得到满足。我们建议在可组合性验证框架的需求规格说明中定义以下强制性（或默认）约束：

S1 = 所有相互作用的组件[25]在语法层面应当可组合S2 = 所有相互作用的组件在静态语义层面应当可组合S 3 a = 相互作用组件的状态机应当彼此匹配，使其能够持续运行，直至到达最终状态或目标状态[26]。S 3 b = 如果将概念模型转换为可执行模型，则后者应当正确表示前者的结构和行为。

表 10：可组合性验证中的强制约束

我们断言，**[S1 ∧ S2 ∧ (S3a ∧ S3b)]** 是整体可组合性验证的必要条件。S1 和 S2 确保组合模型在结构上保持一致。S3a 则确认组合模型的行为在实现给定目标方面是连贯的。满足 S3b 遵循模型验证的定义（见第 4.1 节），因为它确认了该定义的第二部分，即：“将概念模型转换为可执行模型的准确性”。因此，验证过程的整体成功取决于 S3b 约束是否得到满足。这些默认约束的合取提出了需求的三个 C，即：(i) 一致性，(ii) 完备性，以及 (iii) 正确性 [111]。一致性是为了确保组合组件的输入和输出连接保持均衡。完备性是为了确保待组合组件的信息完整，从而检查组合过程不会缺少实现推进所需的输入。正确性则用于确认组合组件按照预期以正确的方式进行交互。

如果所有目标都已实现且所有约束都已满足，那么我们就认为该模型在所有层级上均可组合，并且相对于其规格已得到验证。我们提出的框架的总体目标是提供环境和工具支持，以评估这一主张。

发现的结果是一组候选 BOM；将它们与仿真对象及需求进行匹配后，便可选出适合用于组合的 BOM。将这些 BOM 组合起来即可形成概念模型。

5.3 建模

在建模象限中，以基于 BOM 的组合模型作为输入，并形成概念模型。此外，还会生成形式化模型及其图形表示法（如第2.7.4节所述），以便记录概念模型[27]。鉴于 BOM 本身是一种概念框架，且用于对无法进行任何形式执行的被动组件建模，因此需要对概念模型进行一系列扩展和细化

[25] 在组合模型中，并非每个组件都必须与其他所有组件交互。例如，A、B 和 C 组合后，A 与 B 交互，B 与 C 交互，但 A 不与 C 交互。[26] 如果模型未定义任何最终状态且不会终止，我们就假定模型中存在某些称为目标状态的重要状态；这些状态的可达性可以确认目标已实现。[27] 如果不同团队分别负责开发生命周期的不同阶段，此步骤虽为可选，但会带来益处。本文档有助于理解基本组件及其组合的结构和行为。

通过外部输入和我们提出的模型转换算法，使其能够实现为可执行形式，并发送到“执行”象限（图25）进行抽象层级执行。我们提出的扩展和细化如下：

- BOM状态机到状态图XML（转换）
- 组合BOM到Petri网–PNML（转换）
- 基本 BOM 到扩展 BOM （扩展）
- 扩展 BOM （ E-BOM ）组件到着色Petri网（CPN）组件模型（转换）
- 基本BOM到带时间的扩展BOM（扩展）
- 基本 BOM 到带概率因素的扩展 BOM （扩展）
- BOM 到基于 CSP 的过程模型（扩展与转换）

后续章节将详细说明这些扩展、细化和转换。需要注意的是，每当概念模型得到扩展或细化时，都要在迭代中重新审视建模象限。

5.4 执行

如前所述，此象限主要用于抽象级执行活动。它将建模象限中的概念模型所对应的以下已实现且可执行的形式作为输入：

- 状态图XML（SCXML）
- Petri网 –PNML
- 着色Petri网（CPN）组合组件模型
- 基于通信顺序进程（CSP）的组件进程

后续章节将详细讨论这些可执行形式及其抽象层级执行过程。

5.5 分析

执行过程会产生一些结果。这些结果将在分析象限中进行分析。我们的验证框架支持以下不同的分析技术：

- 状态机匹配分析
- 基于Petri网的代数分析
- 基于着色Petri网的状态空间分析
- 模型检验分析

这些分析技术将在后续章节中讨论。当可组合性验证中的所有必要步骤均已完成，且经过研究的组合模型被认定符合给定的需求规范时，CBM&S 生命周期便会进入实现与仿真的后续步骤，如 图 27 所示。这些步骤的细节不在本文讨论范围之内。

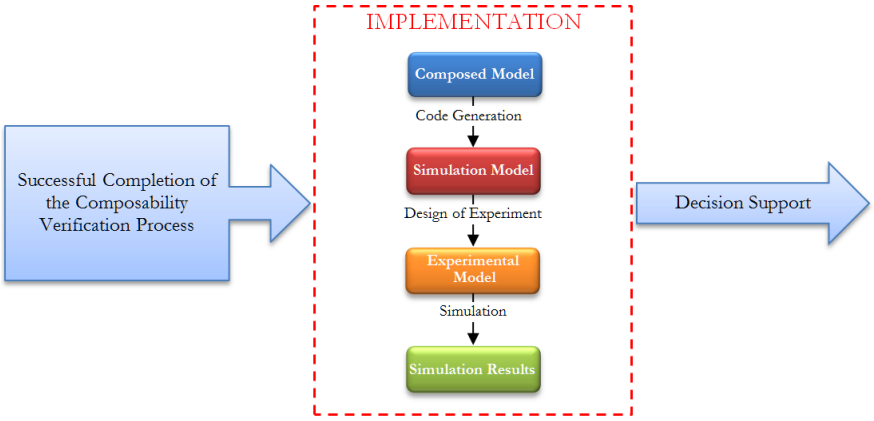


图 27：实现与仿真

5.6 可组合性验证框架

本节将详细讨论我们提出的可组合性验证框架所包含的各种方法、技术、程序、算法和模块，并将其视为 CBM&S 生命周期中的构建模块，与生命周期的不同阶段相连接。了解这些细节对于理解第6章所述的可组合性验证过程至关重要。

5.6.1 发现、匹配与组合（DMC）

在基于组件的开发中，构建可复用组件并将其存储在函数库或存储库中，以便日后按需复用，是一种常见做法。为复用现有组件，会采用发现、匹配与组合（DMC）范式 [19] 。我们假设 BOM组件库维护在一个存储库中。建模者利用仿真对象提供的信息，尝试从存储库中搜索并发现 BOM组件。如果检索到一组候选组件，则会通过匹配过程对其进行筛选。匹配过程将仿真对象中的候选组件与需求规范进行匹配，从而选出适合用于组合的组件。关于BOM组件发现与选择过程中的语法匹配和语义匹配，我们已在 [54] 中提出并进行了详细讨论。本文提出了一组发现规则，在匹配仿真对象中的候选项时必须满足这些规则。我们将这些规则用于候选项与仿真对象之间的语法匹配和语义匹配。此外，我们还建议将候选项与给定需求进行匹配，因为候选项可能与相应的仿真对象匹配，但若不符合其需求，可组合性验证仍会失败。我们在框架中实现了 DMC过程，其示意如 图28 所示。我们还假设，如果存储库中不存在所需组件，则从头构建该组件，并将其添加到存储库中以供复用。DMC过程的结果是一个基于BOM的组合模型。该组合模型作为输入放入建模象限，并被视为系统的概念模型。我们建议建模者也使用我们提出的形式化规范和图形表示法（见第2.7.4节）来构建形式化模型。该形式化模型可用于文档记录，并展示组件的组合方式。不过，这是一个可选步骤，不被视为我们

CBM&S 生命周期。在第7章和第8章中，也描述了示例的形式化模型，以便读者理解。

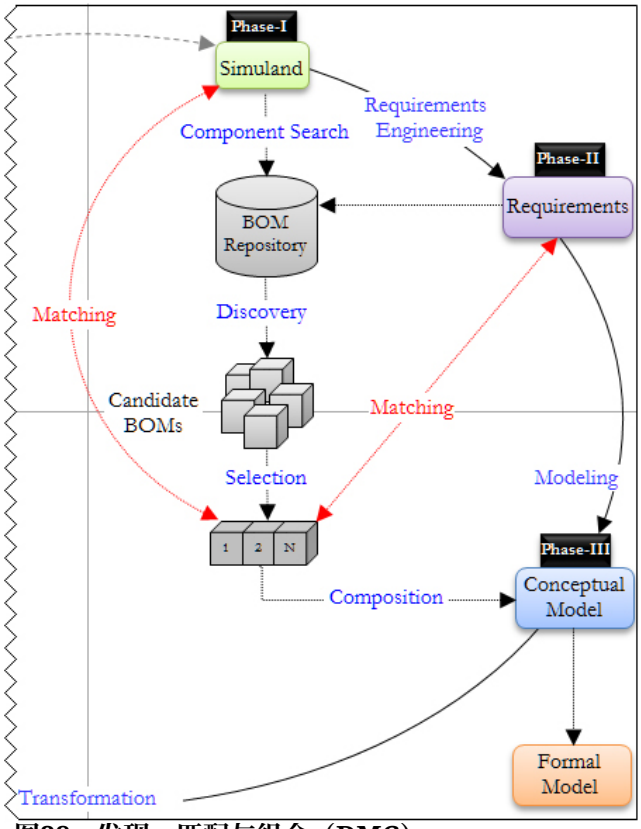


图28：发现、匹配与组合（DMC）

5.6.2 结构与行为评估

建模象限中接收的概念模型需要经过结构与行为评估，以确认模型的一致性、完整性和正确性，并确保其适用于不同可组合性层次的全面验证。在对概念模型进行更深层次的可组合性验证之前，检查其结构和行为是有益的。如果模型在结构和行为方面保持一致，则可信度会提高，据此，后续深入验证过程中可以作出各种有用的假设。

如果模型的结构或行为存在差异，我们就可以跳过后续步骤，节省时间和计算资源，并在重复整个过程之前进行必要的设计细化。这种安排遵循第4.2节所述的自顶向下细化原则。模型的结构通过静态分析技术进行分析（见第4.3.2节），而模型的行为则通过动态分析技术进行评估。

5.6.3 静态分析

我们提出两种静态分析程序：(i) 语法匹配和 (ii) 静态语义匹配。这些程序用于评估结构，并在语法层面和静态语义层面验证可组合性。之所以称为静态分析，是因为评估依据的是预定义规则，无需以任何形式执行，而且应用这些规则所依据的信息是静态的。

语法匹配（SM）

该模块负责根据以下规则评估语法层面的 BOM 可组合性。该模块的结果可验证各组件在语法上能否正确地相互连接。这些规则最初是在 [54] 中介绍的一种 BOM 匹配技术中提出的。

> **SM规则1：**
> 两个组件之间交换的每个事件的名称[28] 都应相同，即发送事件的名称应与接收事件的名称相同。

发送事件定义在 BOM 的事件类型中，其中发送方是 BOM 本身，接收方是其他某个 BOM（在该组合中）；而接收事件则是在 BOM 事件类型中定义的事件，其中发送方是其他某个 BOM（在该组合中），接收方是 BOM 本身。

> **SM-规则 2：**
> 每个发送事件都应至少有一个对应的接收事件，反之亦然；也就是说，发送/接收事件对应该完整。

> **SM 规则 3：**
> 发送事件的参数数量（事件类型的内容特征）应与接收事件的参数数量相同。

满足语法匹配规则1、规则2和规则 3，即可满足默认约束S1（见表 10）；这是整体可组合性验证的必要条件。**图 29**展示了语法匹配活动的不同步骤。

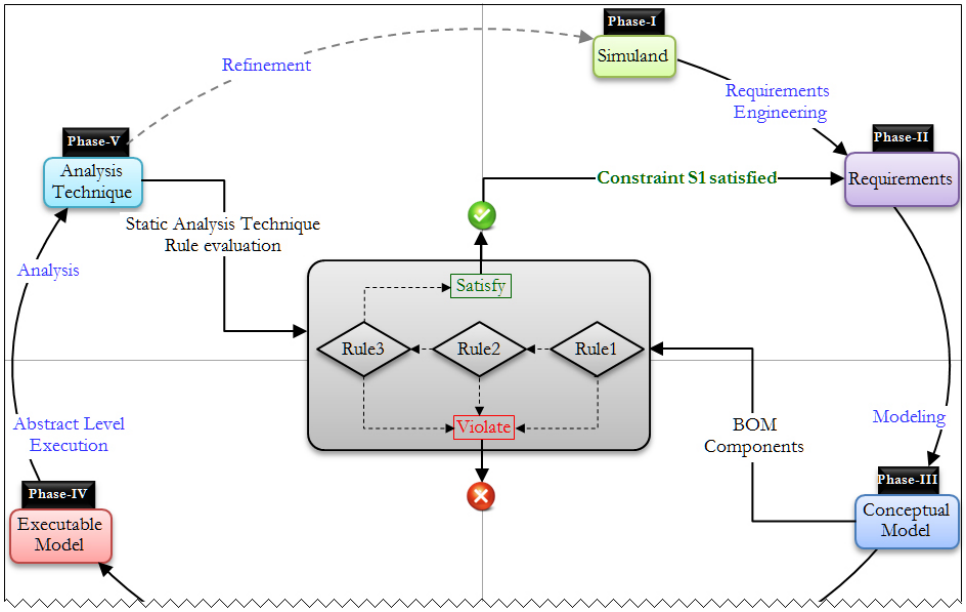


图 29：语法匹配

[28] 假设在 BOM 构建中，事件及其对应的动作具有相同的名称

静态语义匹配（SSM）

该模块负责依据特定规则评估静态语义层面的BOM可组合性。该模块的结果可验证组件组合是否有意义，以及组件之间的通信是否按预期得到理解。为确认这些事实，我们提出在两个层面进行静态语义匹配：(i) 操作级匹配和 (ii) 消息级 [**53**]:

(i) 操作级匹配

在基于BOM的组合模型中，操作由交互模式（POI）描述。POI由参与组合的基本BOM中的一组动作构成。在操作级语义匹配中，需要确保组合组件共享相同的“感兴趣领域”，并且是为相同的目的（或目标）而组合，从而保证该组合在（静态）语义上有意义且不存在任何语用歧义。即使感兴趣领域相同，组件也可能服务于不同目的，例如，在军事领域中，营部（BHQ）组件可能有多种用途，并可参与多种不同的操作。因此，明确所选组件的目的对于获得有意义的结果也很重要。

为确保操作级的语义一致性，我们建议在构建基本BOM时，在动作定义中指定以下语义属性[29]，并在组合基本BOM时，在POI中指定这些属性。在静态语义匹配中，这些属性用于比较并确认BOM组合中涉及的动作是否正确。

o **关注区域：**它描述了使用组件和操作进行建模的系统所关注的区域或领域。我们建议将“关注区域”定义为基本BOM中每个动作以及POI中的语义属性。此属性将确认所有组件共享相同的领域知识。对于可能属于多个领域的某些通用组件（例如队列等），我们建议对该组件进行特化，并使其成为所选关注区域的成员。例如，在餐厅组合模型中，可以将通用队列组件特化为餐厅队列，并使用动作*JoinRestaurantQueue()*和*ServeCustomer()*，而不是Put()和Get()动作。

o **目的：**目的描述整个操作的意图或目标。在 BOM 组合中，POI 表示组合组件正在执行的单个操作。不过，一个或多个组合组件也可能被设计用于多种目的；在特定场景中，多用途组件可能只有一部分参与组合。例如，顾客组件可能是通用的，可用于多种目的；而餐厅服务员组件则专用于餐厅场景。因此，在餐厅场景中选择顾客组件时，其目的应与该场景中的其他组件保持一致。由此，我们将“目的”定义为基本 BOM 中动作的语义属性（多重性为 ≥ 1）。

[29] 在BOM中，概念建模元素（实体、事件、状态和动作）支持语义字段 [**65**]

(ii) 消息级匹配

BOM 表示事件驱动组件，这些组件通过发送或接收事件（消息）来运行。在消息级，组合组件之间的通信必须有意义，并且接收方应能按照发送方的意图理解其语义。在这一层面，我们提出匹配发送事件和接收事件的参数的数据类型和计量单位

[53] 。

[54]

假设这些 BOM组件具有与 [54] 中所提出的相应 OWL附件。BOM-OWL附件用于定义领域本体的语义类、其属性、数据类型和个体，并将其存储在BOM仓库中。为了评估操作级和消息级的静态语义匹配，我们采用以下规则：

SSM规则1

所有动作（涉及某项操作）的“关注区域”属性的交集应与 *POI* 的交集恰好相同，或属于相应本体中的一个等价类[30]：

$$\bigcap_{i=1}^{n} \mathrm{Act_i}.\mathrm{AOI} \cong \mathrm{POI}.\mathrm{AOI}$$

SSM-规则 2

所有动作的“目的”属性的交集应恰好与 *POI* 的相同，或属于相应本体中的等价类：

$$\bigcap_{i=1}^{n} \text{动作}.\text{目的} \cong \mathrm{POI}.\text{目的}$$

SSM-规则 3

发送事件和接收事件的事件参数中每个元素的数据类型应属于同一类、等价类，或具有直接层级关系；也就是说，发送方参数的数据类型应属于接收方参数数据类型的直接子类（反之则不成立）。

例如，发送事件包含一个类型为“秒”的参数，而接收事件需要一个类型为“时间”的参数，根据该规则，两者在语义上匹配。**图30**以基本数据类型为例。在实际情况下，BOM组件会包含更多领域特定的复杂数据类型。

[30]在 OWL 中，如果两个类具有相同的语义含义，并且包含相同的个体（实例），就可以将它们标记为等价。例如，Healthcare 和 Medical 是同义词。我们将其表示为 ≅

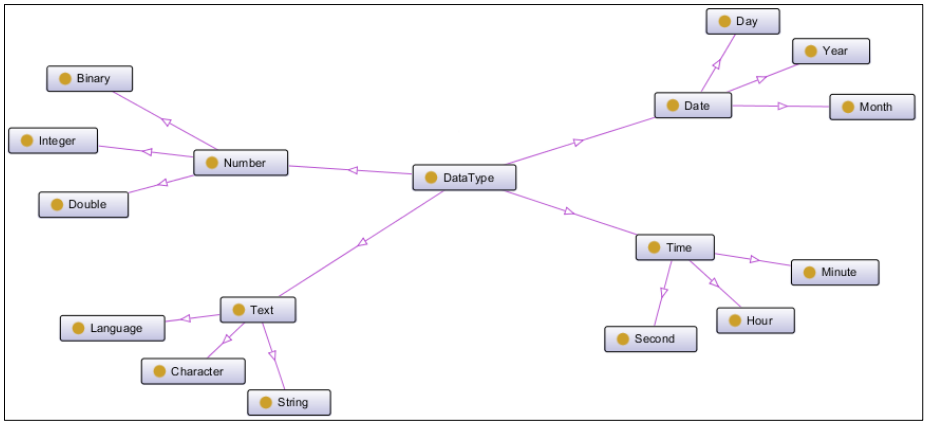


图30：数据类型本体的部分子类

SSM 规则 4

事件参数中表示的测量单位应当相同或等价，或者属于直接类层次结构，从而能够在没有信息损失（或信息损失可接受）的情况下进行转换。

我们假设，如果两个测量单位之间存在直接关系，即父级关系或子级关系，那么它们的转换损失将是可接受的。例如，发送事件有一个单位为 m/s（米每秒）的参数，用于表示速度，而接收事件所需的单位为 Km/hr（千米/小时）。这是有效的语义匹配，因为这些量可以无损转换。

语义匹配技术

为了匹配两个元素，我们提出了一种语义匹配技术，如**图31**所示。该技术使用 OWL-API [112]，语义推理引擎（FaCT++、Pellet 或 HermiT）和 OWL 本体文档，对任意两个元素 A 和 B 发起查询，并将二者的语义关系输出为以下类型之一：

1. 精确（A = B） 2. 等价（A ≅ B，即 A 和 B 属于等价类） 3. 直接父类（A 是 B 的直接父类） 4. 直接子类（A 是 B 的直接子类） 5. 间接（A 和 B 不直接相连，但属于同一层级） 6. 无关系（A 和 B 不相关）

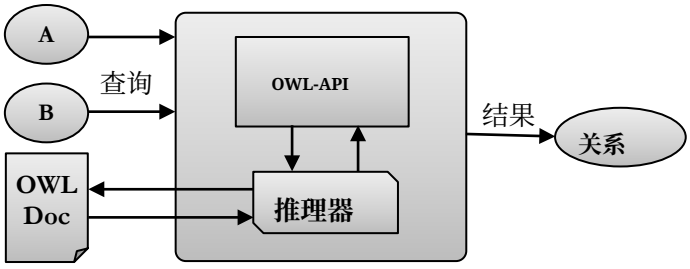


图31：语义匹配技术

此技术用于使用**表11**中给出的算法[31] 评估静态语义匹配规则 1、2、3 和 4。

[31] 本论文中提供的算法，其伪代码约定和格式在大多数情况下均遵循 [132] 所制定的指南。

算法：语义匹配

输入：{Actions}、POI、BOM-OWL

输出：TRUE、FALSE

```
Owl ← 加载本体(BOM-OWL)
 {CommonAOI} ← ⋂_{i=0}^{n} a_i ∈ 动作.AOI ▷ 返回所有动作的共同关注区域集
对于 caoi ∈ {CommonAOI} 执行
    SR1 ← 获取语义关系(caoi, POI.AOI, Owl) ▷ 假设 Get-Semantic-Relation()
    函数使用图图 31 ◁所示的语义匹配技术实现
    如果 SR1 = “精确”或“等价” 则 ▷ 规则1满足……继续
        next
    else
        返回 FALSE
    结束如果
结束循环

共同目的 ← ⋂_{i=0}^{n} a_i ∈ 动作.目的 ▷ 给出所有动作的共同目的集合
对于 通信端口 ∈ { 共同目的 } 执行
    SR2 ← 获取-语义-关系(cp, POI.purpose, Owl)
    如果 SR2 = “精确”或“等价” 那么 ▷ 规则2满足……继续
        next
    else
        返回 假
    结束如果
结束循环

{Events} ← 获取事件(动作) ▷ 获取动作对应的事件
    对于 e ∈ 事件 执行
        如果 e=发送事件 则
            f ← 获取接收事件(e, 事件) ▷ 获取 e 对应的接收事件
            {PE} ← e.Parameters ▷ 发送事件 e 的参数集
            {PF} ← f.Parameters ▷ 接收事件 f 的参数集
            ▷ 由于 SM 规则 3，e 和 f 的参数数量必须相同
            对于 pe∈PE & pf ∈PF 执行
                SR3 ← 获取-语义-关系(pe.Type, pf.Type, Owl) ▷ 比较参数类型
                如果 SR3 = “精确”或“等价”或“直接-子级” 则
                    ▷ 规则 3 满足……继续执行规则 4
                    SR4 ← 获取-语义-关系(pe.Unit, pf.Unit, Owl) ▷ 比较单位
                    如果 SR3 = “精确”或“等价”或“直接-父级”或“直接-子级” 则
                        返回 真 ▷ 静态语义匹配成功
                    else
                        返回 假
                    结束条件判断

                else
                    返回FALSE
                结束条件判断
            结束循环
        else
            next
            ▷ 转到下一个发送事件，无需检查接收事件（因为 SM-规则2）
        结束条件判断
    结束循环
```

表11：语义匹配算法

语义匹配算法以一组动作（从待组合的基本BOM中解析而来）、用于指定各动作之间如何连接的交互模式（POI），以及相应的 OWL 本体文档作为输入。如果静态语义匹配成功，该算法的输出为 TRUE；如果任何规则被违反，则输出为 FALSE。**图 32**展示了静态语义层面上验证BOM可组合性的步骤。

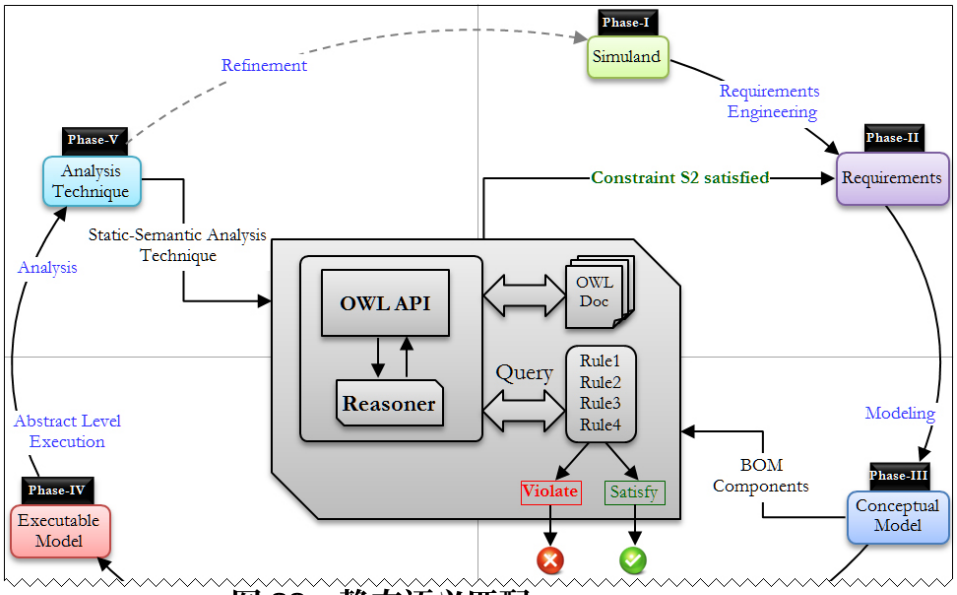


图 32：静态语义匹配

如果语义匹配成功，则满足需求规范的默认约束（S2）（见**表 10**），这是整体可组合性验证的必要条件。

5.6.4 动态分析

我们使用动态分析技术（见第 4.3.3 节）评估概念模型的行为。首先，对组件执行状态机匹配过程，以评估行为一致性。评估成功后，我们将选择所提出的多种动态分析技术之一，在动态语义可组合性层进行深入验证。由于这些分析需要在不同抽象层执行（如第 5.4 节所述），因此称为动态分析。

状态机匹配（SMM）

状态机表示组件的行为，是BOM组件中必不可少的动态部分。在动态语义层面对BOM可组合性进行验证时，组合组件的行为彼此协调一致至关重要；也就是说，它们的交互必须保持一致，才能朝着组合目标推进。为确保这一点，我们断言（作为必要条件），组合组件的状态机应彼此匹配。BOM状态机本质上是事件驱动的，并通过交换事件来推进。为确保组合BOM组件的状态机彼此匹配，需要在抽象层执行这些状态机。因此，我们提出了一种技术 [113]，可将每个BOM状态机转换为SC-XML（状态图XML）[114] 格式。SCXML示例如**图33**所示。

```
<scxml initialstate= STATE1>
    <state id = STATE1 final = True>
        <transition event=EVENT NAME
            target=STATE2/>
    </state>
    <state id = STATE2 final = false>
        <transition event=EVENT NAME
            target=STATE1/>
    </state>
</scxml>
```

图33：SCXML格式

我们使用SCXML API开发了一个用于执行的运行时环境。该环境会解析SCXML文件（转换后的BOM状态机）并创建实例。随后，它将所有状态机初始化至其初始状态，并模拟事件的发送和接收，以观察状态机变迁，直至它们到达最终状态。状态机匹配过程基于以下算法：

算法：状态机匹配

输入：{SM} ∈ BOM状态机，{Actions}
输出：TRUE，FALSE

```
{SCXML}← 将状态机转换为SCXML(句法匹配)
▷ 将所有 BOM 状态机转换为 SCXML 格式 ◁

创建 和 初始化 事件控制器：EC
▷ 事件控制器负责控制事件的发送和接收 ◁

用于 scxml ∈ { SCXML } 执行
    SC ← 解析(scxml) ▷ 解析 scxml 文档
     创建并初始化 SCXMLExecutor(SC)
    ▷为每个状态机实例化一个 SCXMLExecutor 线程 ◁

    完成 ← 假
    当 (完成 =假) 时
        当前状态 ← 获取当前状态() ▷ SCXMLExecutor 返回当前状态
        如果 当前状态.是最终状态 = 真 则
            完成 ← 真
        结束 if
        ▷获取要发送或接收的下一个动作 ◁
        {下一个动作}← 当前状态.获取动作()
        对于下一∈ 下一个动作执行
            如果 下一.类型 = “发送” 那么
                EC.Put(next) ▷ 模拟发送下一个动作
                SCXMLExecutor.Trigger(next) ▷从当前状态转移到下一状态
            else
                EC.Get(next) ▷ 模拟接收下一个动作
                SCXMLExecutor.Trigger(next) ▷从当前状态转移到下一状态
            结束 if
        结束 for
    结束 while ▷由于发送或接收动作，状态-机将
                转移到下一状态，因此当前状态将被更新。
                如果到达最终状态，则状态机匹配将
                成功终止◁
结束 for
```

表12：状态机匹配算法

图34展示了状态机匹配过程。该过程以 BOM 状态机为建模对象，自动将其转换为 SCXML 可执行格式，并在抽象层执行环境中进行状态机匹配。

该程序成功运行意味着所有状态机彼此匹配，从而满足 BOM 可组合性的必要（但非充分）条件，即需求规范中的约束S3a。满足 S3a 可证明组合组件的行为设计具有一致性和完备性。一致性源于组件处于正确的因果顺序；完备性则是因为其输入和输出（发送和接收事件）足以使其达到最终状态。然而，除非组合满足其需求规范，即满足所有指定目标和所需约束，否则我们仍无法保证需求的正确性 3rd C。此外，状态机匹配方法可能会使系统达到最终状态，但不会探索组合组件行为交互的所有可能性。因此，需要采用适当的动态分析方法对模型进行更深入的分析。

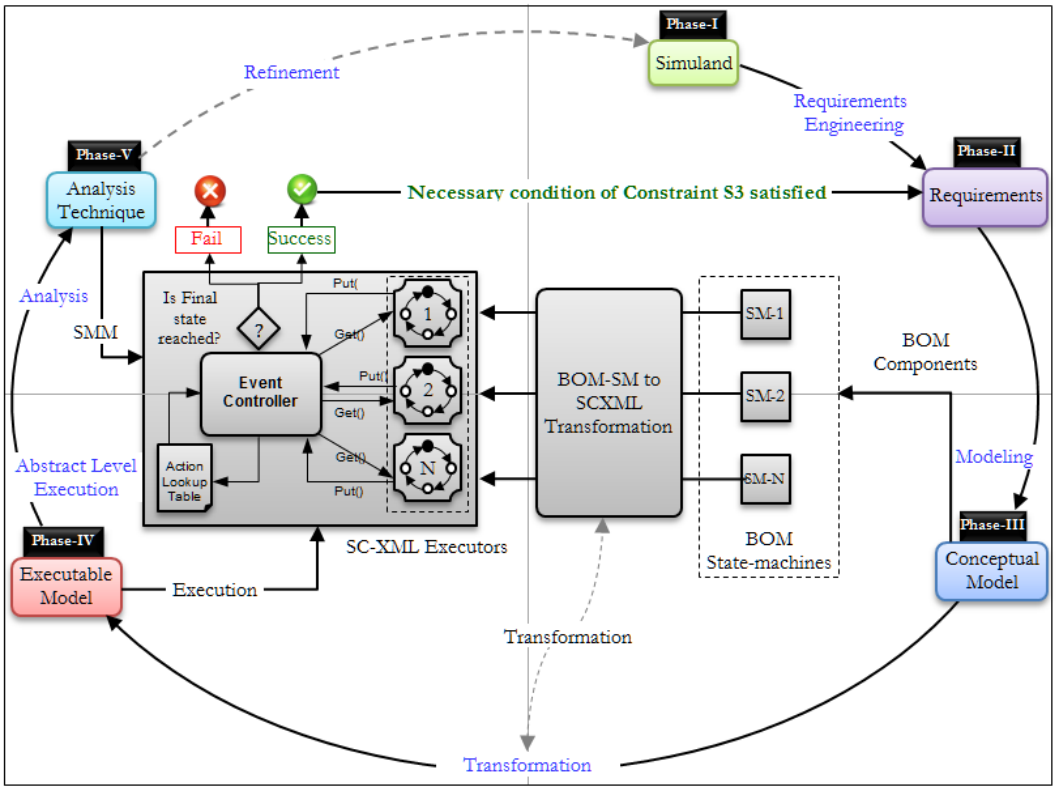


图34：状态机匹配过程

因此，为了进行更深入的评估，我们提议利用 Petri网 和 CSP形式化体系的建模与分析优势，并将第3章介绍和讨论的三种分析方法纳入我们的验证框架。在动态语义层面选择适用于可组合性验证的方法，取决于模型的性质。以下各小节将详细讨论这些方法。

5.7 PN代数技术

该技术的基本思路是将 BOM 转换为 Petri网 格式，并使用代数方法验证需求规范中给出的属性。在验证框架中，我们提出以下步骤来进行代数分析：

5.7.1 BOM到PNML的转换

第一步，将BOM组件转换为Petri网标记语言（PNML）格式[115]，这是一种基于XML、用于描述库所/变迁网的格式。首先，解析所有组件的BOM状态机，并将每个**状态**转换为Petri网模型中的一个**库所**。类似地，每个**事件**（发送或接收事件）都转换为Petri网中的一个**变迁**，且不重复。如果发送方的对应状态S以事件t作为退出条件，且下一状态为S′，则在库所P与变迁t之间连接一条出弧。从变迁t到另一个表示下一状态S′的库所P′之间连接一条入弧。类似地，接收方的状态R转换为库所Q，下一状态R′转换为Q′。入弧和出弧均连接到t。BOM中的发送方和接收方实体以令牌的形式表示在库所中。**图35**展示了发送方和接收方状态机的一部分如何转换为Petri网。库所P和Q中的令牌表示组合模型的当前状态（或标识）。当变迁t开火时（即事件t由P发送并由Q接收），令牌被传送到P′和Q′，表示组合模型的下一标识。

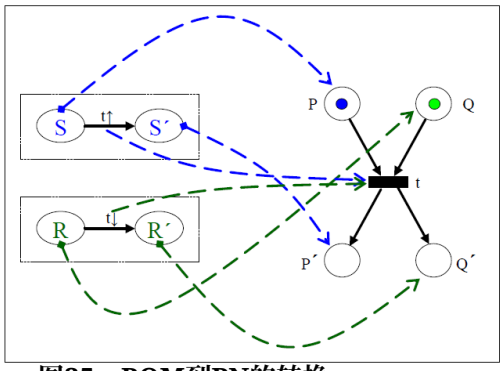


图35：BOM到PN的转换

当 BOM 中每个状态机的所有状态和事件都绘制在 Petri网模型中，且没有重复元素，并且每个库所或变迁都已连接、没有断开的链接时，转换过程即告完成。

5.7.2 PN代数计算

在此步骤中，将计算 Petri网的发生矩阵以及库所/变迁不变量。为执行此步骤，我们使用平台无关 Petri 网编辑器（PIPE）API [116]。PIPE 是一个基于 Java 的开源 API，可执行各种与 Petri网相关的操作。它提供 API 功能，可自动计算 Petri网模型的代数资源，例如关联矩阵和库所/变迁不变量。

关联矩阵

A个 Petri网模型的发生矩阵通过用 A+ 减去 A- 矩阵计算得出：

算法：关联矩阵计算

输入： Petri网模型（P-库所 × 变迁）

输出： m × n 矩阵 A

```
1  初始化一个大小为 m × n 的矩阵 A_minus，其中 m=|P| 且 n=|T|
2  对于 i=0 到 m 执行
3      对于 j=0 到 n 执行
4          如果 pi ∈ P 与 tj ∈ T 那么 ⊳ 即，p 是 t 的输入库所
5              A[i][j] ← 弧权重 ⊳ 弧权重始终为 ≥ 1
6          else
7              A[i][j] ← 0
8          结束如果
9      结束循环
10 结束循环
11
12 初始化一个大小为 m × n 的矩阵 A_plus，使得 m=|P| 且 n=|T|
13 对于 i=0 至 m 执行
14     对于 j=0 至 n 执行
15         如果tj ∈ T 与 pi 相连∈ P 则⊳ 即，p 是 t 的输出库所
16             A[i][j] ← 弧权重 ⊳ 弧权重始终为 ≥ 1
17         Else
18             A[i][j] ← 0
19         结束 if
20     结束 for
21 结束 for
22
23 初始化一个大小为 m × n 的矩阵 A
24 对于 i=0 到 m 执行
25     对于 j=0 到 n 执行
26         A[i][j] ← A_plus[i][j] - A_minus[i][j]
27     结束循环
28 结束循环
29 返回 A
```

表 13：关联矩阵计算

第 10 行计算 A- 矩阵。第 12–21 行计算 A+ 矩阵，第 23–28 行计算最终的关联矩阵。

库所不变量和变迁不变量

计算 Petri网模型的 P-不变量和 T-不变量的方法已得到广泛研究。计算 P-不变量和 T-不变量基本集的基本原理基于 Farkas方法 [117]。寻找 P-不变量的算法如下所示。该过程的输入是关联矩阵 A 和单位矩阵 B，二者的大小均为 m × n。输出是矩阵 C，其各行构成 P-不变量的基本集。

算法：P不变量计算	
输入： 关联矩阵 A，单位矩阵 B	
输出： 矩阵 C（C 的行 = P-不变量）	
1	C ← A | B ▷ 将 A 与 m × n 阶单位矩阵 B 增广
2	对于 i=1 到 n 执行 ▷ n = |T|
3	对于 每一对行 c_1, c_2 在 C[i-1] 其中 $c_1[i]$ 和 $c_2[i]$ 符号相反 执行
4	$c \leftarrow |c_2[i]|. c_1 + |c_1[i]|. c_2$
5	$c' \leftarrow$ 行 c 中每个元素的 c/g.c.d ▷ g.c.d =最大公约数
6	将行 c' 添加到矩阵 C[i-1]
7	结束循环
8	删除 C[i-1] 中 i^{th} 分量非零的所有行，结果为 C
9	结束循环
10	返回 C

表 14：库所不变量

对关联矩阵 A 进行转置后，可使用相同的过程来求取 T-不变量。[118] 中介绍了该算法改进的细节和相关讨论。这些算法已在 PIPE API 中实现，可通过函数调用的形式使用。

5.7.3 性质验证方法

代数分析技术的结果是判断某个性质相对于 Petri网模型是否满足或违反。执行属性验证有多种方法，不过，关于推理某个性质成立的必要和充分条件，通常都有相应的定理。在 Petri网文献中，已有许多用于证明属性证明定理的解法（证明），可在需要时用于证明不同的性质。利用这些定理和现有的代数资源，可以开发出一种性质验证方法（算法），在 Petri网模型上评估定理给出的条件，并得出所需属性得到满足或遭到违反的结果。**图36**展示了验证框架中代数验证技术的机制：

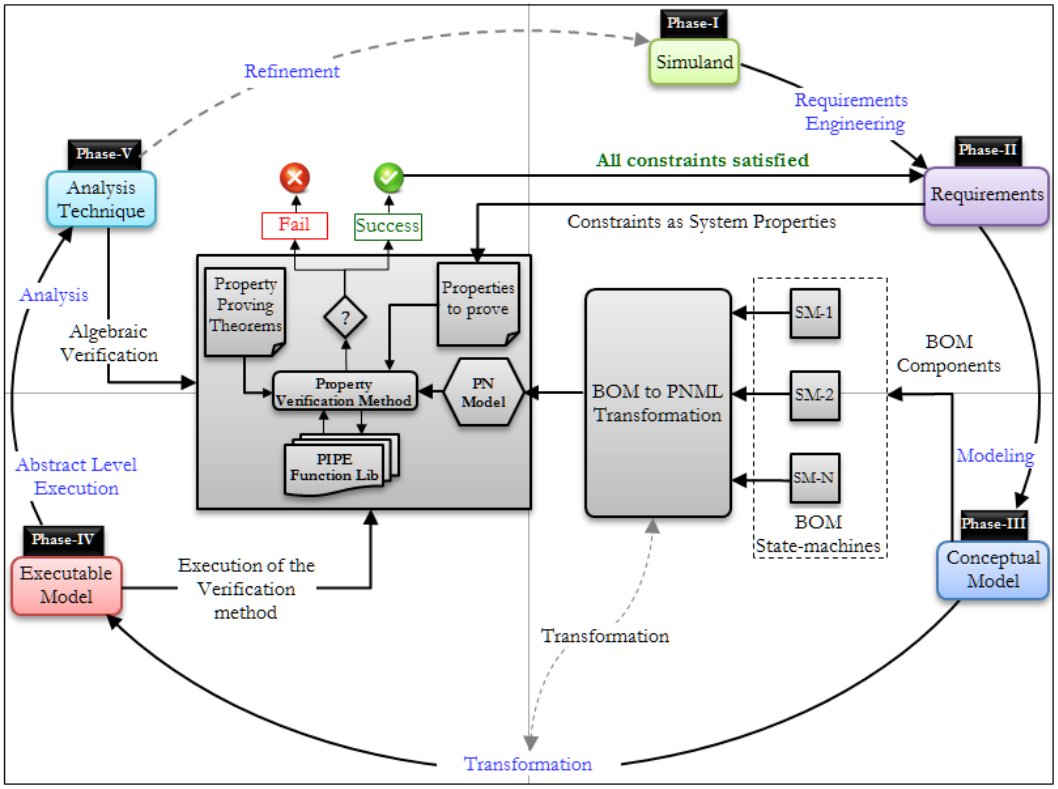


图36：PN代数技术

为了解释我们的方法，我们将在第7章介绍定理，并给出一种示例性质验证方法，用于分析 Petri网模型中的公平性。

PNML 执行与状态空间图

需要注意的是，PIPE库还提供了一个执行环境，可用于运行转换后的PNML模型。如果令牌（每个令牌代表一个BOM实体）最终到达其最终状态（库所），则执行成功。这表明模型已正确转换，并且能够正确表示其源（即概念模型）的行为。PIPE库还提供了用于生成和可视化PNML模型状态空间图的函数。通过图可达性分析，这有助于发现死锁并验证其他系统性质。

5.8 基于CPN的状态空间分析技术

为验证动态语义可组合性而提出的第二种方法，基于有色Petri网和状态空间分析技术。该方法有效利用了有色Petri网形式化、CPN建模与编程语言、其执行环境及支持工具的潜力，以针对需求规范在动态语义层面对组合模型进行验证。该方法的独特之处在于其以数据为中心的特性。如第3.1.5节所述，CPN支持三级PN建模，其中令牌具有结构化形式，并可表示数据对象。此外，变迁能够涵盖系统行为的更多细节。因此，可以更详细地建模系统的结构和行为。为了发挥该方法以数据为中心的特性，我们提出以下阶段：

5.8.1 BOM扩展

当前的BOM标准缺少某些结构和行为语义，而这些语义对于复杂系统行为建模至关重要，因此我们需要规范其他模态，以便更详细地捕捉系统的结构和行为[119]。因此，我们提出通过应用扩展有限状态机（EFSM）的概念来扩展BOM概念模型规范，该概念在[120]中作了详细介绍和讨论。扩展有限状态机（EFSM）由以下元组定义：

$M = (Q, I, \Sigma_1, \Sigma_2, V, \Lambda)$ where:

Q (≠∅) 是一个有限状态集。

$I \subset Q$ 是初始状态集

Σ_1 是一个有限的（发送或接收）事件集。

Σ_2 是一个有限的动作集（动作是要执行的指令，不应 d
与 BOM 动作混淆，后者用于交互模式中）。

V 是状态变量集。

Λ 是一个变迁集；每个变迁 $\lambda \in \Lambda$

$$\lambda = q \xrightarrow[\{v_{in}\} \mid \{v_{out}\}]{e\,[g]\,/\,a} q'$$

其中

q 和 $q' \in Q$ $e \in \Sigma_1$ 是一个事件，g 是一个条件（或守卫），$a \in \Sigma_2$ 是一个动作。

这意味着，如果系统处于状态 q，发生事件 e，且守卫 g 满足，则将执行动作 a，系统将转移到下一状态 q′。在变迁 $\lambda \in \Lambda$ 射击过程中，变量 $\{v_{in}\}$ 用作输入，变量 $\{v_{out}\}$ 用作输出。

示例：

本例是对 [120] 中讨论的队列扩展有限状态机的修改版本，旨在说明扩展有限状态机的概念。队列组件的状态为空或非空，插入操作在队列后端进行，删除操作在队列前端进行。此外，队列具有最大容量。使用放入和获取两个事件来更新队列的状态。

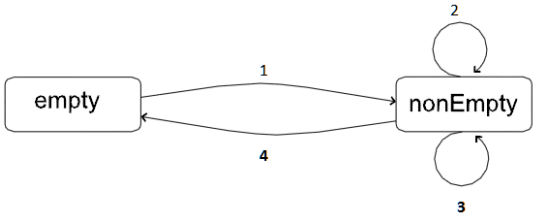


图 37：缓冲区扩展有限状态机 [120]

缓冲区的扩展有限状态机模型为：M = **(Q, I, Σ_1, Σ_2, V, Λ)**，其中

Q= {空, 非空}

Σ = {放入(字符串 obj), 获取}

q_0: 空

V = {前端, 后端, M, 数据}

Λ：转换规范：

1：空 —— Put(obj) [ ] / 动作{ 后端++; 数据[后端]=obj;} / {后端} | {后端, 数据} ——→ 非空

当接收到放入事件时，转换1允许队列从空状态转移到非空状态。在此变迁期间，变量后端会递增。此外，Put事件的参数“Obj”存储在数据的后端位置。

2: 非空 —— 放入(obj) [后端<M]/ 动作{ 后端++; 数据[后端]=obj;} / {M, 后端} | {后端, 数据} ——→ 非空

如果后端小于最大容量，接收到放入事件时，转换2会使队列重新进入非空状态。在此变迁期间，变量后端会递增。此外，Put事件的参数“Obj”存储在数据中的后端位置。

3：非空 —— 获取(数据[前端]) [后端≥前端+1∧后端 <M]/ 动作{ 前端++} / {M, 前端, 后端, 数据 } | { 前端 } ——→ 非空

变迁3使队列返回非空状态。当后部变量大于或等于前端+1 且小于最大容量时，该变迁触发。它将发送Get事件，并将前端位置的数据作为参数发送。在此变迁过程中，前端变量递增。

4：非空 —— 获取(数据[前端]) [前端+1=M]/ 动作{后端=0; 前端=0} / {M, 前端, 数据} | { 后端, 前端 } ——→ 空

变迁 4 允许队列在 front+1 达到最大容量时返回空状态。它会发送一个 Get 事件，并携带位于前端位置的数据。在此变迁过程中，前端和后端变量都会重置为零。

我们将扩展有限状态机的概念应用于 BOM 概念模型，从而引入状态变量以及变迁的扩展表示（事件、守卫条件、动作），并将其命名为：扩展 BOM 或 E-BOM。BOM 扩展有若干优势：

在 BOM 状态机中使用变量（或状态变量），可以对组件的属性（结构）及其因状态变化和变迁发生而产生的影响（行为）进行建模。这些属性的值还可以

用于算术或逻辑求值，以评估变迁的触发条件，并将变量值从一个模型传递到另一个模型。因此，在组合中，一个组件输出的值可以由组合中的另一个组件使用。同样，通过引入动作，我们可以在变迁发生期间操作组件的状态变量，从而对仅通过简单的带标签迁移（例如标准 BOM 中的事件/动作）无法实现的系统真实且复杂的行为进行建模。

在将BOM扩展为E-BOM的过程中，有些特征从现有BOM导入，其他特征则由建模者提供。**图38**比较了哪些部分可以从BOM导入、这些部分在E-BOM中如何使用，以及哪些部分由建模者从外部提供。我们为建模者设计了一个输入工具，建模者可以通过该工具选择初始状态、从现有BOM实体特征中选择状态变量，或添加新变量，并为这些变量指定数据类型。建模者还可以使用现有BOM中的事件来设计变迁，并添加守卫条件（条件语句）和动作。目前，我们允许建模者使用SML语言编写守卫条件和动作脚本。这是因为扩展BOM之后将用于CPN环境，而该环境使用SML编写代码脚本。将每个BOM组件扩展为相应的E-BOM后，我们便进入下一阶段。

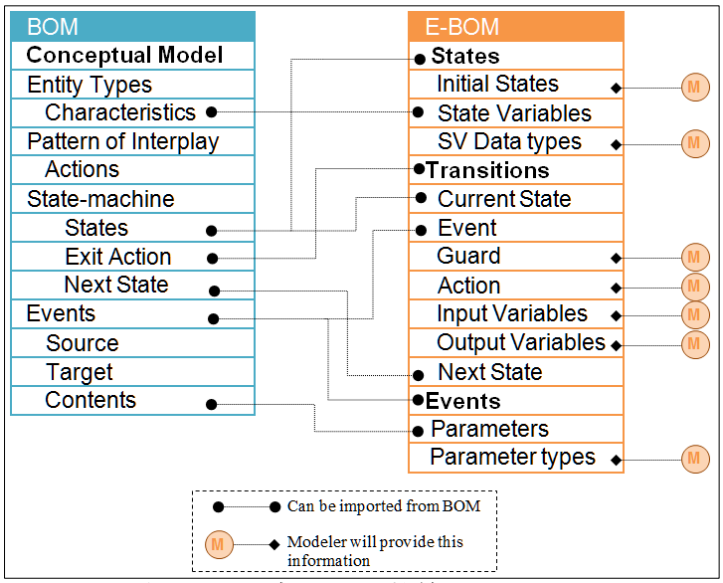


图38：BOM与E-BOM比较

5.8.2 E-BOM到CPN组件的转换

我们提出了一种基于 CPN构造[32]的三层组件模型，并使用元模型转换规则（将在本节后文定义）将先前开发的 E-BOM 转换为该组件模型。我们提出的 CPN组件模型定义如下：

CPN-CM = ⟨SL, BL, CL⟩ where:

SL = 结构层 BL = 行为

层 CL = 通信层

[32]要理解这一机制，需要具备 CPN 形式化体系和 CPN工具环境的背景知识。我们建议查阅在线文档 [82]

结构层

结构层表示组件的物理属性或特性。E-BOM 中定义的状态变量会转换为结构层。在 CPN-CM 中，这些状态变量以 CPN库所的形式定义，并由颜色表示不同的数据类型。在 CPN模型执行过程中，通过触发相连的变迁对其进行操作。如果状态变量被定义在变迁的输入或输出集合中（如 EFSM 中所述），它们就会与这些变迁相连。状态变量的值由 CPN令牌指定。令牌的数据类型与各自状态变量库所的数据类型相同。结构层的形式化定义如下：

SL = 〈SV, Σ_{SV}, C_{SV}, V_{SV}, I_{SV}〉 其中

- SV 是一个库所集（用于表示概念模型的状态变量）
- Σ_{SV} 表示一个颜色集集，用于表示状态变量数据类型
- C_{SV}：SV→Σ_{SV} 是一个为每个状态变量指定颜色集的函数
- V_{SV} 是一个有限的类型化变量集[33]，分配给 SV 中的每个成员。
- I_{SV}：SV → EXPR 是一个初始化函数，它为每个状态变量指定一个类型为 Type [SV] 的初始值表达式。

行为层

行为层以 CP 网的形式表示 E-BOM 组件的状态机，其中 E-BOM 组件的每个状态都变为一个整数类型的 CPN 库所，每个 E-BOM 变迁都变为一个 CPN 变迁，而流动的整数类型令牌表示组件状态的变化。如果结构层的状态变量在 E-BOM 变迁中被指定为输入或输出，则它们会与本层的变迁相连。
E-BOM 中的守卫条件和动作会转换为 CPN 变迁标注。它们应符合 CPN-ML 脚本语法，否则 CPN-ML 编译器将报错。形式上，行为层定义如下：

行为层 = 〈S, T, A_T, A_{SV}, A_C, G, Act〉 其中

- S 是由库所组成的集合，称为状态。每个库所都指定为 INT 类型。该集合表示 E-BOM 状态机的状态
- T 是由变迁组成的集合（表示 E-BOM 变迁）
- A_T 是由弧组成的集合，称为传递弧。每条传递弧连接一个状态 s ∈ S 和一个变迁 t ∈ T，并连接 t 与另一个状态 s′ ∈ S，若 s → t → s′。A_T 中的每条弧都指定一个 INT 类型的变量 v。变迁 t 开火时，会将一个令牌从状态 s 转移到状态 s′，以反映状态机中的状态变化。
- A_{SV} 是另一组弧，称为“*sv* 弧”，它将状态变量（结构层中的）连接到变迁。如果在 E-BOM 变迁中将状态变量 SV 指定为输入变量，则通过一条入弧将其连接到该变迁。如果在 E-BOM 变迁中将状态变量指定为输出变量，则通过一条出弧将该变迁连接到状态变量。每条*sv* 弧 上都指定了一个变量 v，其类型与所连接的状态变量相同。

[33] 请注意，这些变量是内置的 CPN 变量，用于定义传递令牌的弧绑定，与我们所说的状态变量不同

• **A_C** 是另一组弧，称为“通信弧”，它将通信层的通信端口（CP）连接到变迁。如果某个 cp∈CP 对应于一个接收事件（如 E-BOM 中所指定），则通过一条通信弧 将其连接到变迁 t。相反，如果某个变迁 t 对应于一个发送事件，则通过一条弧将其连接到 cp∈CP。每条通信-弧上都指定了一个变量 v，其类型与所连接的 CP 相同。

• **G: T→表达式** 是一个守卫函数，用于为变迁 t（如 E-BOM 中所规定）指定守卫，使得 Type[G(t)] = **布尔值**

• **Act: T→表达式** 是一个动作，并被指定给变迁 t（如 E-BOM 中所规定）。该动作可以是单条可执行语句，也可以是完整的过程。通常，动作用于从关联的输入状态变量或通信端口读取数据；处理数据，并将结果写入关联的状态变量或通信端口。动作也可用于对数据进行类型转换。

通信层

通信层负责与其他组件进行通信。它通过“端口库所”提供用于连接组件输入和输出的接口，同时还提供有关数据交换类型的信息，即承载消息参数内容的复杂数据类型令牌，如 E-BOM 事件类型中所述。行为层的变迁与该层的端口库所相连。弧方向取决于变迁的类型（发送或接收）。当一条消息（E-BOM 事件）在通信层的输入端口处被接收时，会触发行为层中的变迁，从而使状态机（位于行为层中）向前推进。当某个变迁在输出端口处产生一个预期由其他组件接收的发出事件时，也可能发生这种情况。形式化地，通信层 CL 定义如下：

〈通信端口, Σ_{CP}, C_{CP}, V_{CP}, PT〉

• 通信端口是称为“通信端口”的端口库所集合。通信端口将组件的输入/输出连接到其他组件（见第 3.1.5 节）。每个通信端口对应一个事件（发送或接收）。

• $\Sigma_{通信端口}$ 表示一个颜色集集合，该集合表示通信端口的数据类型（由 E-BOM 的相应事件参数指定）。每个颜色集由所有参数的数据类型组合而成。如果没有参数，则分配空类型，该类型携带一个空白令牌。

• $C_{通信端口}$: 通信端口→$\Sigma_{通信端口}$ 是一个为每个端口库所分配一个颜色集的函数。

• $V_{通信端口}$ 是一个有限的类型化变量集合，分配给每个通信端口，且 Type[v]∈ Type[通信端口]。

• PT：CP→ {输入、输出、输入/输出} 是端口类型函数，它为每个端口库所分配一个端口类型（如第 3.1.5 节所述）。

E-BOM 到 CPN-CM 转换规则

从 E-BOM 到 CPN-ML 的转换规则总结如下：

E-BOM		→	CPN组件模型		
			结构层	行为层	通信层
1	状态	→		类型[整数]的状态库所	
2	初始状态	→		添加令牌类型[整数], 值 = {0, 1...n} 对于每个初始状态	
3	状态变量（SV）	→	（类型为颜色集的）状态变量库所定义于 4 中，如果状态变量（SV）需要初始化		
4	状态变量数据类型	→	定义状态变量库所的颜色集		
5	事件	→			添加通信库所对于每个颜色集类型的事件在 6 中定义
6	事件参数及其类型	→			添加一个与事件参数类型相同的颜色集如果有事件参数，多个参数，则添加产品类型的颜色集，其中每个事件参数类型都会添加到产品中。（如果没有参数，则使用空类型）
7	BOM-动作	→		变迁	
8	退出条件（动作，下一状态）	→		连接一条过渡弧从状态库所到变迁，并从变迁到下一状态库所-为库所声明变量 v 类型为[int]，添加到每条弧上。	
9	变迁守卫	→		为每个变迁添加变迁	
10	变迁动作	→		为每个变迁	
11	{Input-Variables} 的变迁	→	从每个 SV 到变迁连接 SV 弧如果它被添加为该变迁中的输入变量。在……上声明 SV 类型的 sv arc.		
12	……的{Output-Variables}变迁	→	将 SV 弧连接到每个来自某个变迁的 SV，前提是它被添加为该变迁中的输出变量。在……上声明类型为[SV] 的 sv arc.		
13	接收事件 ∈ 事件	→			从通信端口连接一条弧到该变迁（如果它有一个接收事件。声明 CP 在[CP] 上声明一个类型为的变量弧。将 CP 标记为输入端口。
14	发送事件 ∈ 事件	→			从以下位置连接一条弧变迁到 CP（如果它有一个发送事件。声明 cp 类型为[CP] 的变量弧，将 CP 标记为输出端口

表15：转换规则

示例

让我们来看看上一节讨论的队列示例的 E-BOM n:

E-BOM：队列组件				
状态	{空, 非空}			
初始状态	{空}			
状态变量	{前端 : 整数, 后端 : 整数, 最大值 : 整数, 数据: 字符串[ ]}			
& **它们的 D 数据类型**				
事件	{放入 (obj:字符串), 获取(obj:字符串)}			
变迁				
	Put	**Put**	**Get**	**Get**
当前状态	空	非空	非空	空
事件	放入(obj)	放入(obj)	获取(空值)	获取(空值)
守卫	[ ]	[后端<M]	[后端≥前端+1∧后端 <M]	[前端+1=M]
动作	后端++	后端++	前端++	后端=0；前端=0
输入变量 后端		后端，M	后端，前端，M，数据[前端]	后端，前端，数据[前端]
输出变量 后端，数据(obj)		后端，数据(obj)	前端	后端，前端
下一状态	非空	非空	非空	空

基于上述 E-BOM，转换将生成**图39**所示的 CPN-CM 模型。该模型可在 CPN环境中完全执行，并可通过通信层与其他组件组合。

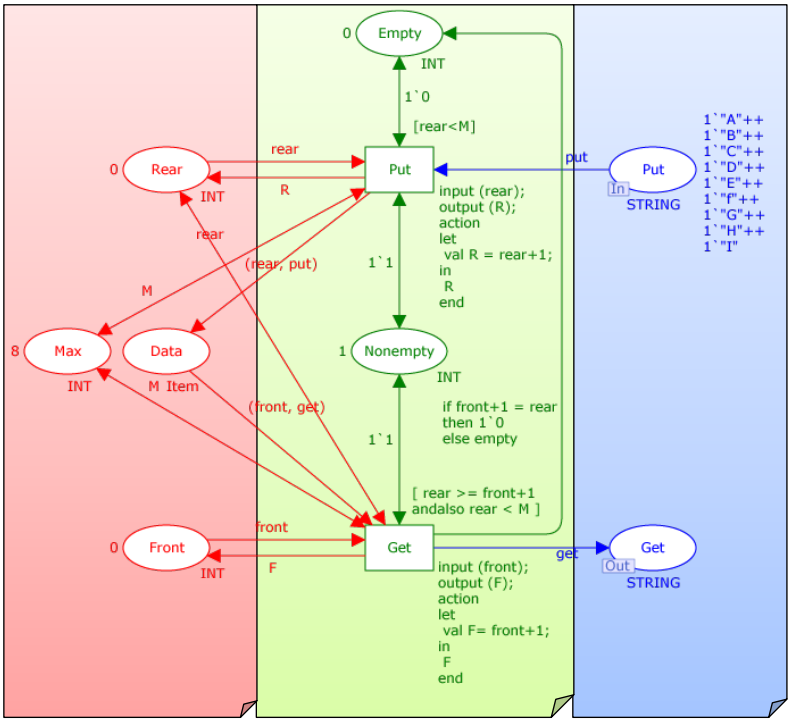


图39：队列组件的 CPN-CM 表示

红色区域表示结构层，由状态变量组成。绿色区域表示行为层，展示状态机。蓝色区域表示通信层，展示通信端口（CP）。当 CP-Put 接收到一些数据时，后端和 Max 变量的值将作为输入读取。守卫 rear<M 得到满足。从空状态中取出一个令牌。执行动作，

将后部变量递增，然后最终触发变迁 Put。之后，非空状态中会产生一个令牌，以显示状态转换。此外，后部变量和数据变量也会更新。Max 变量保留该令牌（由于双向弧）。如果再次触发 Put，则会重复相同的过程。如果触发 Get 且守卫得到满足，则将前端、Max 和数据变量作为输入读取。数据（从队列前端取出）将被发送到输出CP。当数据被清空时，令牌将被发送到空状态。

自动化转换工具

为了自动化 E-BOM 到 CPN 的转换过程，我们开发了一种转换工具，它以 E-BOM 组件为输入，自动生成 CPN 组件模型三个层的 CPN 代码。代码遵循 CPN-XML 规范。对于每个 E-BOM 组件，都会以编程方式生成一个单独的 CPN 子页，并在一个 CPN 输出文件中生成必要的 CPN 元素（库所、变迁、弧、颜色集、变量声明、初始标识多重集、守卫条件、动作、代码段、CPN ML 函数、端口、端口标签），该文件可加载到 CPN 工具中。生成 BOM 组合的所有 CPN 模型后，建模者会创建一个主模型，并“手动”组合生成的 CPN-CM 模块（使用 CPN 层次化特性）。此步骤的输出是一个组合 CPN 模型。建模者还需要为每个组件初始化数据（以令牌赋值的形式，即状态变量的令牌初始值和状态机的初始状态）。

S3b 评估

需求规范中的 S3b 约束要求：“如果概念模型被转换为可执行模型，那么后者应正确表示前者的结构和行为”（参见**表 10**）。因此，我们必须将每个 CPN 组件（可执行模型）与其对应的 BOM（概念模型）进行比较，以检查其结构和行为在转换后是否得以保留。为了证明转换后 S3b 仍然成立，我们依赖以下断言：

1. 由于 BOM 扩展为 E-BOM，因此 BOM ⊂ E-BOM。建模者在 E-BOM 中添加的任何信息都不会导致 BOM 的结构信息丢失。因此，E-BOM 在结构上保留了 BOM。 2. 为检查生成的 CPN 组件是否包含所有事件及其参数、状态及其退出条件、动作及其发送方/接收方，我们至少需要一条负责转换这些元素的转换规则：a. 规则6和规则7（参见**表15**）负责将事件及其参数转换为 CPN 组件。b. 规则1和规则8负责转换状态及其退出条件。c. 规则7负责将 BOM-动作指定为 CPN 模型中的变迁。此外，规则5定义了用于连接发送方或接收方的端口库所。规则1、6、7和8的存在证实了相应 BOM 的结构在转换过程中得以保留。

3. CPN 工具提供了用于编译 CPN 模型的内置编译器，并会报告模型中是否存在语法错误。没有错误表明转换后的模型在结构上保持一致，且在行为上能够正常运行。

对于行为双模拟，我们提出了一种检查技术。首先，我们评估所有生成的组件是否具有概念模型中定义的相同行为。因此，我们通过提供所需的输入来测试每个CPN模型的功能输出。如果提供正确的输入后，模型能够产生预期输出，那么其功能行为就是正确的。为执行功能测试，建模者用所需参数的令牌初始化所有输入型通信端口（CP）。（示例见**图39**，其中输入型通信端口“放入”用字符串类型的令牌初始化。）随后执行模型。如果模型在相应的输出通信端口上产生预期输出（在**图39**中，预期输出应是在“获取”输出CP处取出的字符串类型令牌），则功能测试成功。建模者会对所有生成的CPN组件执行功能测试。

第二步，在组合所有CPN组件之后（即主模型的套接字库所与CPN组件的通信端口库所相连之后），建模者需要检查CPN组件是否严格按照BOM组合的交互模式连接。此外，执行组合模型时，一个组件向另一个组件发送和接收事件的序列（可通过观察主模型中令牌的移动来查看）是否遵循交互模式，也需要进行检查。如果执行符合交互模式，且组件持续推进直至到达最终状态，那么我们就认为转换后的模型与概念模型行为双模拟。这证实了S3b 约束得到满足。

执行可以采用自动模式或交互模式。在自动模式下，系统会随机选择多个可继续路径中的一条；而在交互模式下，建模者可以自行选择路径。建模者可以利用此选项探索可能导向成功执行场景的路径。执行过程中，CPN工具还提供数据收集监视器，用于记录数据值，这对于收集执行过程的统计数据和结果非常有价值。

5.8.3 组合CPN模型的验证

下一步将进行状态空间分析。首先，使用CPN状态空间计算工具生成组合CPN模型的状态空间。如第3.1.5节所述，状态空间是由节点（系统状态或标识）和弧（变迁）组成的图。生成状态空间后，可以使用不同的查询函数来探索状态空间图，以解答各种验证问题。查询函数类似于一种用于探索状态空间图的算法。这些算法基于Petri网状态空间分析的理论概念，用于验证Petri网属性。因此，我们将需求规范中给出的系统属性转化为适当的Petri网属性。PN文献中已有大量关于如何规范Petri网属性，以及如何推理其可满足性或违反情况的研究贡献。在CPN状态空间分析中，可以利用现有方法为相应的Petri网属性开发查询函数。

CPN 工具为常见查询任务提供了一些内置函数。我们还提出了一个附加函数库，用于执行针对我们

的可组合性验证框架和需求规范的特定查询。图40展示了使用查询函数对组合CPN模型进行状态空间分析的过程。我们将这些查询函数分为两类:

(i) 一般系统性质

此类别包含常见的系统性质，例如无死锁性、活锁、饥饿，或有界性、互斥、公平性、顺序性、时间同步等。如果需求规范将这些或类似的系统性质中的任何一项纳入约束，则会将其转换为 CPN 术语，并从函数库中选取合适的查询函数，利用组合 CPN模型的状态空间进行验证。

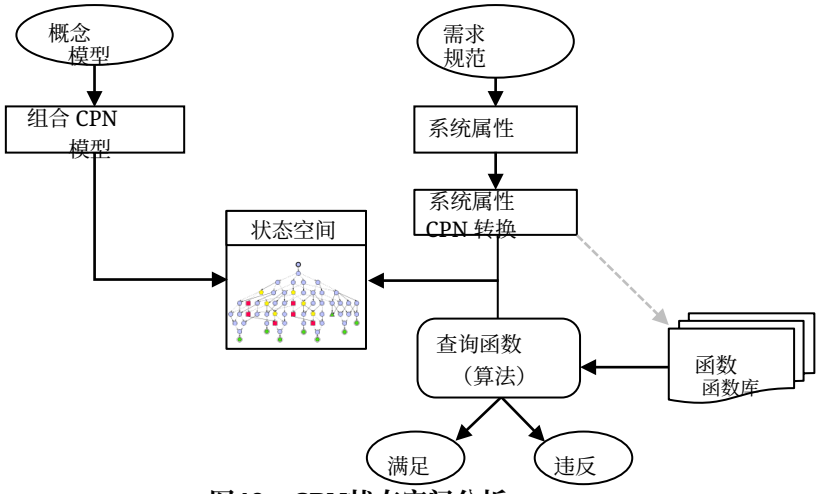


图40：CPN状态空间分析

例如，无死锁性可以转换为 CPN 术语，表示为:

“整个状态空间图中不存在没有出弧的标识”

因此，本质上我们需要在状态空间图中找到一个违反上述条件的节点。如果没有找到这样的节点，则称该模型无死锁。库函数 **`ListDeadMarking()`** 会返回一个由所有没有出弧的标识组成的集合（如果存在）。如果该查询的结果为空列表，则我们断言该模型无死锁。类似地，还有其他库函数用于评估其他系统性质。

（ii）场景特定属性

这些属性特定于构建模型所依据的真实系统场景。需求规范中的目标或目的通常会被转换为场景特定属性。在 CPN 术语中，典型的目标或目的可以转换为某种期望的标识：结构层中状态变量的值符合特定标准，或行为层中达到特定状态；也可以是期望通信层的输出端口上出现某些数据。目标或目的也可以由上述各种可能性的组合来表达。

场景特定属性还可能包含某些安全性或活性假设，用以表示为了满足（或违反）需求而必须（或不得）出现的某些期望（或不期望）情形。这些属性大多是对需求规范中定义的约束进行 CPN 转换的结果。

与一般系统性质不同，验证场景特定属性并非标准操作，具体取决于这些属性的定义方式。最常见的做法是使用我们提出的库函数：**`IsEqual()`**、**`IsNotequal()`**、**`IsBetween()`**、**`IsUpperBound()`** 或 **`IsLowerBound()`** 来构造**“谓词”**，作为条件评估准则。随后，我们使用 **`SearchNode(`**谓词**`)`**函数查找满足该谓词的节点。如果找到一个或多个节点，则验证目标可达。在需要了解变迁发生的顺序如何引向某个特定情形（即某项属性得到满足的情形，例如如何达到某个目标或目的）的情况下，我们会将该谓词与 **`SearchArc()`**函数配合使用。它会告知我们图中通向属性满足状态的路径。我们还开发了一个导出函数，用于生成整个状态空间的 .DOT 文件；该文件可在 GraphViz 或 Gephi 等图形工具中查看，以便进行可视化，并在图上开展进一步测试，例如查找两个特定节点之间的某些路径、最短路径或最长路径。当 CPN 组合模型满足需求规范中的所有属性时，我们称其在动态语义可组合性级别上通过了验证。第8章将讨论一个野战炮兵场景，作为 CPN 状态空间分析的示例，以阐释我们的方法。

用 CPN 术语表达特定场景性质的一个例子是餐馆模型。在该模型中，我们假设顾客可能会因为等待服务员送来账单的时间过长，而不付账就离开餐馆。顾客的这种行为称为“弃绝”，是不受欢迎的。在 CPN 中可将其表述如下：

> “图中不应存在一条名称为“*balk*”的弧，使其通向任何标识”

弧由变迁的射击生成。存在balk弧意味着模型中的某处发生了客户拒绝（通过射击balk变迁）时的发生。因此，我们实质上需要查找状态空间图中不存在该弧的情况。这可以使用**`SearchArc()`**函数完成。请注意，这是一个简单示例。在某些情况下，可能需要搜索变迁序列（称为轨迹）或环，以验证某个性质。**错误！未找到引用源**。描述了状态空间分析技术的整体过程。

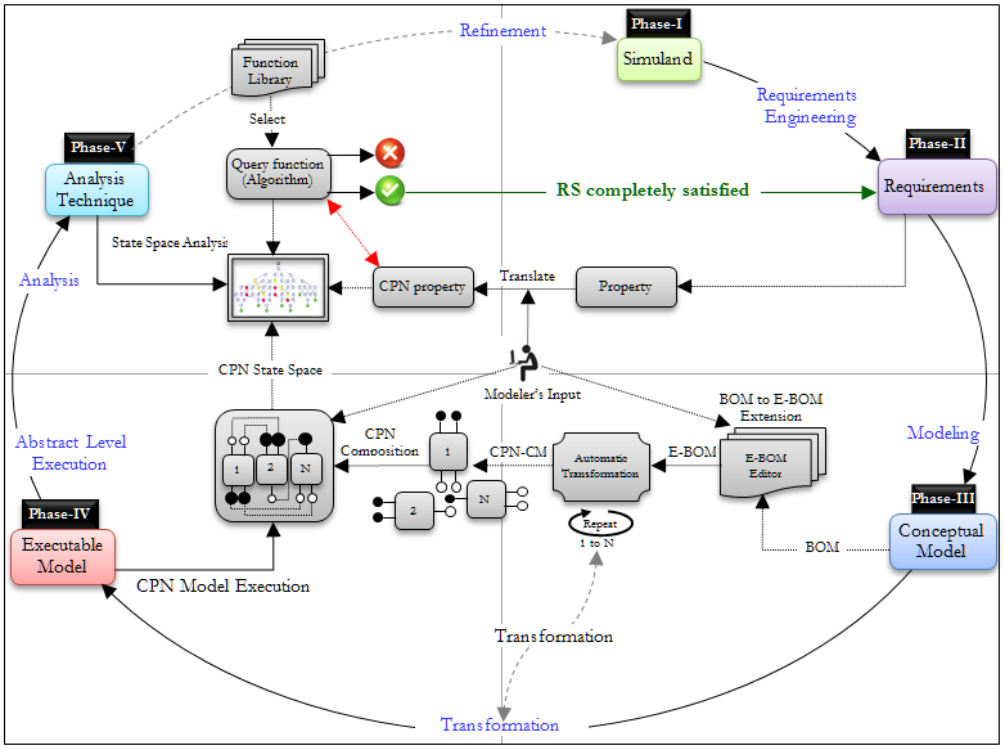


图41：状态空间分析技术

状态空间缩减技术

为了缓解众所周知的状态空间爆炸问题，我们提出了一种称为“组合状态空间”的约简技术。该技术的主要思想是，在CPN模型的层次化组合中，我们只考虑主模型中的库所，并将所有组合组件视为黑盒。每个组件的输入和输出都可以通过令牌的流动及其携带的数据进行观察。因此，在状态空间图中，我们仅保留主模型中存在任意令牌的标识（即主模型中的任意库所至少包含一个令牌），并使用**表16**中给出的算法删除状态空间图中的所有其他节点。所得图是实际图的约简形式，仅考虑反映组合状态空间。的那些标识。之所以称为组合状态空间，是因为它只表示实际状态空间的一部分，而这一部分是组件组合所产生的交互的结果。根据我们的经验，整个状态空间的这一子集足以评估目标、目标和约束是否得到满足。

算法：组合状态空间生成	
输入： 原始状态-空间图 G	输出： 约简状态空间图 G

```
1  {顶点} ← 获取-顶点(G) ⊳ 将图中的所有节点提取到一个集合中
2  对于 v∈到 {Vertices} 执行
3      如果 假 ← 是否-已筛选(v) 则
4          G ← 移除顶点(G, v)
5      Else
6          next
7      结束如果
10 结束循环
11 返回 G ⊳ 约简状态空间
12
13 过程 移除顶点(图 G，顶点 v)
14 {前驱顶点} ← 获取-前驱顶点(G，v) ⊳ 获取 G 中 v 的所有前驱顶点
15 {后继顶点} ← 获取- 后继顶点(G，v) ⊳ 获取所有后继顶点 v 在 G 中的顶点
16 对于 p∈到 { 前驱顶点 } 执行
17         对于 s∈到 { 后继顶点 } 执行
18             G ← 添加边(p, s, "有向") ⊳ 从每个前驱节点添加一条有向弧
19                                          至每个后继节点 ⊲
20             G ← 删除顶点(v)
21         结束循环
22 结束循环
23 返回 G
24
25 过程 是否-已筛选（顶点 v）
26         如果 {places}← 获取数据(v) 那么 ⊳ 每个顶点都是一个标识，其中包含 所有
27         ⊳ 模型中的库所及其名称和令牌值 ⊲
29  对于 p∈到 { places } 执行
30           如果 p 是一个主库所且不为空 那么
31                 返回 TRUE ⊳ 找到一个有效的 非空标识，其中 主库所
32           else
33                 next
34           结束条件判断
35       结束循环
36       ⊳ 如果循环已完成，则不存在非空的主库所
37 返回 FALSE
```

表16：组合状态空间生成算法

利用组合状态空间生成算法，我们可以过滤不必要的节点并缩小图的规模。这种方法目前的局限在于，我们首先需要构建实际的状态空间；如果模型过大，这就会成为瓶颈。不过，这一局限源于无法从外部修改 CPN 状态空间图的生成过程。否则，如果将我们的约简技术原理应用于状态空间生成算法，就能直接生成约简后的图。在第8章中，我们将介绍把约简技术应用于野战炮兵示例模型后得到的结果。

5.9 基于 CSP 的模型检验技术

针对动态语义可组合性验证提出的第三种方法基于模型检验。模型检验被广泛认可为一种用于软件验证的形式化技术。在这种方法中，我们提出使用通信顺序进程形式作为模型描述语言，并使用流程分析工具包（PAT）作为其执行和验证环境，以便根据需求规范在动态语义层面验证组合模型。这种方法的优势在于能够回答各种各样的验证问题，因为验证标准可以使用线性时序逻辑、计算树逻辑或任何时序逻辑扩展来描述。模型检验技术正变得越来越有前景，也越来越受到许多软件验证用户的认可，因为模型检验社区不断贡献改进的算法、高效的数据结构和更快速的技术，以便处理具有复杂建模需求的大型模型。我们提出将基于 CSP 的模型检验验证方法集成到可组合性验证框架中。为使用模型检验方法执行可组合性验证，我们提出以下几个阶段。

5.9.1 BOM扩展

用于基于 CSP 的模型检验方法的 E-BOM 扩展同样受到第5.8.1节所述扩展有限状态机概念的启发。CSP 的扩展 BOM 也可以包含状态变量，但由于 CSP#规范不允许声明字符串或更高阶数据类型，因此状态变量的定义仅限于整数和布尔值[34], ；根据我们的经验，这些类型足以使用 CSP 对 BOM组件的行为进行建模（否则，就需要将组件简化为细节较少的版本，只规定必要的行为细节）。E-BOM 的变迁包含当前状态、事件（及其参数）、守卫、动作和下一状态。不过，在这种情况下，动作脚本使用 CSP#规范语言而非 CPN-ML 语言编写。此外，我们不使用输入和输出变量，而是使用仅供组件访问的局部变量，以及组合模型中所有组件均可访问的全局变量。拟议的 BOM 扩展还将纳入时间约束和概率因子等其他信息，以便对实时系统和概率系统等复杂系统的行为进行建模和验证。

由于定时CSP支持多种定时行为模式来表达定量时序需求，例如延迟、超时和截止期限，因此我们建议在 BOM 扩展中将这些模式用作时间函数，这有助于将 E-BOM 自动转换为定时CSP组件。如下面的表格所述，这些时间函数本质上是分配给 E-BOM 变迁的:

[34] 通常，由于需要控制状态规模并避免状态空间爆炸的风险，大多数模型检验描述语言都不允许使用高级或用户定义的数据类型。不过，如果不可避免地需要使用此类类型，PAT工具确实提供了从外部库导入类的机制。在这种情况下，建模者需要在 PAT 中手动编写组件，而不能依赖我们的自动转换工具。

时间函数	用法与说明
等待[持续时间]	Wait 用于对活动中的延迟进行建模。一个使能的 变迁会等待给定的持续时间，然后触发。
超时[持续时间，下一] 为	变迁指定超时函数后，它会等待一个 事件发生。如果事件在超时之前发生 ，它将转移到 变迁定义中描述的下一状态；否则，它将转移 到超时参数中描述的下一状态。
截止期限[持续时间]	变迁受约束，必须在截止期限到达时开火。 两者的区别在于，**等待[]** 和**截止期限[]** 函数的不同之处在于 前者会使系统处于非活动状态，也就是说，系统除了 等待之外什么也做不了；而使用后者时，系统处于活动状态，可以 响应事件等，直到其截止期限到达。

表17：E-BOM中的时间函数

在 BOM 扩展过程中使用这些时间函数中的任意一种，都有助于描述实时系统的行为。为了进一步描述复杂反应式系统的行为，我们还建议在 BOM 扩展中引入概率因素。这些概率因素既可以用于将系统行为建模为马尔可夫决策过程（MDP），如第3.2.5节所述；也可以使用特定的概率分布函数，对随机时间延迟、超时或截止期限进行建模。 在对 MDP 行为进行建模时，可以按照 PAT工具提供的以下记法，将概率因素分配给组件状态的多个变迁：

概率选择
{
[P1]: 转换1
[P2]: 转换2
...
[Pn]: 转换 n
};
其中 $\sum_{i=1}^{n} Pi = 1$

为了对时间函数进行随机化，我们建议将常用的概率分布函数作为参数分配给 E-BOM：

概率分布 函数	用法和说明
正态分布[均值、方差]	返回一个符合给定参数的正态分布随机值 均值和方差。（由于 PAT 不支持高阶类型 因此我们将这些函数限定为使用整数）。
离散[a， b]	返回一个介于 a 和 b 之间（包括 a 和 b）的离散均匀分布随机值 使得 a < b
指数分布 [1/ l a m b d a]	返回一个来自参数为 1/lambda 的指数分布的随机值

表18：概率分布函数

为了实现这些赋值，我们用 C# 开发了一个外部函数库，可导入 PAT 并在其中使用。调用这些函数会根据指定的概率分布生成一个随机数。除了时间函数之外，这些函数还可用于为全局变量或局部变量生成随机值，从而有助于对不同的概率系统行为进行建模。将每个 BOM 组件扩展为相应的 E-BOM 后，我们便进入下一阶段。

5.9.2 E-BOM 到 CSP# 转换

在此阶段，每个 E-BOM 组件都会转换为一个 CSP# 过程组件，并组合成一个可执行系统。此转换的主要思路基于 [121], , 该文献讨论了将 UML 状态机转换为 CSP。为了能够将其用于 E-BOM 转换，我们对这一转换进行了扩展，加入了通信通道、时间函数和概率因子。**表 19** 展示了转换过程中所用的规则：

E-BOM		CSP# 语句与描述
状态	→	状态名称() E-BOM 中的每个状态都定义为一个 CSP 过程。 最终-状态() = Skip; 此语句定义了 CSP 中的一个最终状态，其中 Skip 是一个保留字，表示该过程终止成功。如果组合模型的任何组件中都不存在此类语句，则该模型被称为非终止模型。
组件	→	组件名称 = 初始-状态() 或 组件名称(i) = 初始-状态(i) 定义一个初始状态并将其赋予组件。如果组件有多个实例，则传入参数'i'，用于表示实例编号。
变迁	→	**简单转换：** 状态() = [守卫] 事件 !/? 参数 {动作} → 下一状态(); 变迁采用上述格式定义，其中 State() 是组件的当前状态。 [守卫] 是一个条件语句。只有当其为真时，变迁才会使能。事件通过以下方式发送通过事件通道使用"!"符号发送，或使用"?"符号接收。对于 E-BOM 中的每个事件组件，我们按如下方式定义通道： 通道 事件名称 0; 在 CSP# 中，"0"表示通信通道的缓冲区大小为零，这进一步意味着它是一个同步通道。参数是在事件交换过程中传递的值，并且使用点号"."分隔 动作是变迁开火时应执行的脚本。通常，这些动作会用于更新局部变量或全局变量。NextState() 是变迁开火时将到达的新状态。它必须在 CSP 组件体内定义。
		带时间函数的变迁： 以下语句表示一个带定时函数的变迁： 状态() = [守卫] 等待[d]; 事件 !/? 参数 {动作} → 下一状态(); 状态()=[守卫] 事件!/?参数 {动作} → 下一状态() 截止期限[d]; 状态()=[守卫]事件?参数{动作}→下一状态() 超时[d] 下一状态2 (); 请注意，在发生超时时，变迁只能接收事件。

		马尔可夫决策过程风格的变迁：
		状态() = 概率选择{
		[概率1]: [守卫] 事件 !/? 参数 {动作A} → 下一状态A()
		[概率2]: [守卫] 事件!/? 参数 {动作B} → 下一状态B()
		};
		请注意，变迁中动作或下一状态的后缀 A 和 B。采用这种 CSP# 代码风格，可以对多个变迁进行建模，并为其赋予不同的概率，以创建动作的一个变体，该变体当仿真运行中选中这些变迁之一，或选中下一状态（或两者）时触发。

		具有概率分布函数的变迁： 要使用概率函数，首先需要导入我们的外部概率函数库 使用： #import "PAT.Lib.ProbabilityDistributionFunctions"; 以下是一些函数调用方式示例： 变量x = 调用(正态分布, 10, 4); 定义了一个整数变量，该变量将通过正态分布随机赋值，其均值为=10，方差为=4

		等待[调用(指数分布, 1/4)]; 采用指数分布的延迟函数，其中到达间隔率为 1/4。
状态-变量	→	变量变量-名称=初始-值; or #定义 常量 初始-值; 在 CSP# 中使用弱类型变量，这意味着声明变量时不会指定类型。所有组件都可以访问全局变量，而局部变量只能由其所属的组件访问。
组件	→	组件名称 = 初始-状态() 或 组件名称(i) = 初始-状态(i) 定义一个初始状态并将其分配给组件。如果组件有多个实例，则会传入一个参数'i'，表示实例编号。
组合模型	→	组合模型 = 组件1 ||| 组件2 ||| ... 组件N; 组合模型（名称）定义为由 CSP 流程组件组合而成，并使用交错算子'|||'将各组件连接起来。但是，如果使用广播事件（即将一个事件发送给所有组件）、一对多或多对一同步事件，则会使用并行运算符将 CSP 流程组件组合起来; 组件组合时使用运算符'||'。

表 19：E-BOM 到 CSP# 转换规则 s

我们开发了一种转换工具，它以所有 E-BOM 为输入，并使用 CSP#描述语言输出单个组合模型。生成的 CSP# 组合模型可以在 PAT工具中打开并编译，以检查错误。如果未发现错误，则认为转换后的模型在结构上是一致的，在行为上是可运行的，并且已就绪，可用于仿真和验证。也可以通过命令行操作直接编译、执行和验证该模型。

S3b 评估

需求规范中的 S3b 约束要求："如果将概念模型转换为可执行模型，后者应正确表示前者的结构和行为"（参见 **表 10**）。为了评估 S3b，即检查生成的可执行模型（CSP 组合模型）的结构和行为是否正确表示其概念模型（ BOM 组合），我们提出以下步骤：

1. 对于每个 CSP 组件，手动检查其是否包含对应 BOM 组件中的所有状态。
2. 检查 BOM 中每个状态的退出条件是否对应 CSP 中的变迁和下一状态。
3. 在 PAT 模拟器中执行生成的 CSP 模型，并观察所有组件是否都能到达其最终状态（若模型为非终止模型，则检查每个组件是否迭代地重新访问其初始状态）。

步骤 1 和 2 通过检查确认生成模型的结构正确表示其概念模型，而步骤 3 则确认其行为与概念模型双模拟，因此满足 S3b 约束。

5.9.3 组合CPN模型的验证

在此阶段，使用 PAT模型检查器对 CSP组合模型进行可组合性验证。首先，将需求规范翻译为 CSP# 性质（或断言）描述语言。该语言基于经典线性时序逻辑 (LTL) 及其实时 LTL、概率 LTL 等不同扩展的混合形式，用于构造各种类型的断言（验证问题），例如可达性属性、安全性属性、活性属性、无死锁性等。我们使用 PAT 断言规范语言的句法，将给定需求规范中的目标和约束翻译为

。以下是如何描述 PAT 断言的一些通用示例：

1	#assert 系统 无死锁;	This 断言 检查 死锁 “系统”中的自由度
2	#assert System reaches goal?0;	此断言用于检查 “系统”是否能够达到其目标（通过 接收一个 goal 事件 取值为‘0’ 参数）
3	#assert 系统 |= <>目标?0;	这是一个等价的 LTL断言 它检查目标是否最终 可达。
4	#assert 系统 |= []<>目标?0;	此 LTL断言检查 目标是否始终最终可达 是否由 系统实现。请注意，它是 与断言2不同。
5	#assert 系统 |= <>目标?0 截止期限[50];	This 断言 验证 goal 带时间约束的可达性，也就是说， 它会检查目标是否可达 是否在 50 个时间单位内?
6	#assert 系统 |= <>目标?0 概率;	此断言用于检查最小值和 max 概率 of the goal 可达性。
7	#define 目标 (Some-Variable == True); #assert 系统达到目标;	这是验证目标的另一种方式 可达性，其中目标 定义基于某个变量的值 。

表 20：PAT 断言的一些示例

定义断言且其句法正确后，我们可以运行 PAT模型检查器，并从列表中选择所需的断言来进行验证。模型检查器会显示验证结果：若验证成功，则表明断言已通过验证；若断言不满足，则会提供一个反例。第9章将通过一个野战炮兵示例，展示如何使用定义为 PAT 断言的需求规范来验证 CSP组合模型。**图42**描述了状态空间分析技术的整体流程。

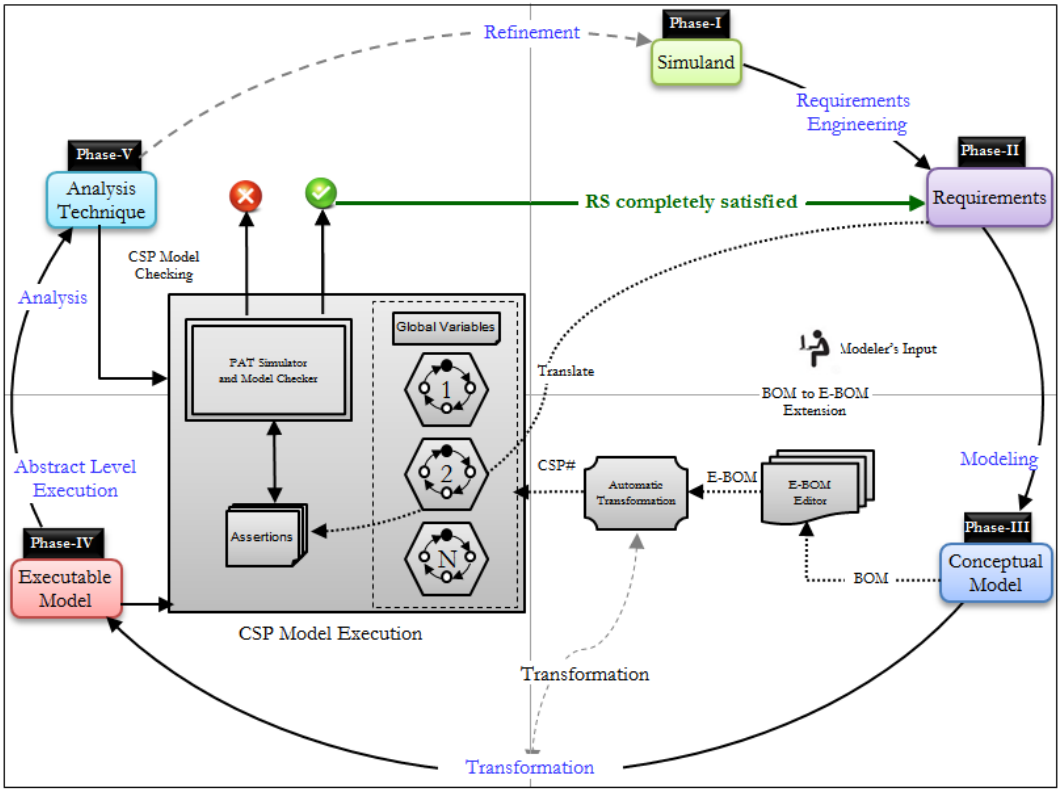


图42：基于CSP的模型检验技术

5.10 总结

本章将详细讨论所提出的可组合性验证框架及其结构和功能规范。我们将从基于组件的建模与仿真生命周期的角度，解释每项活动、算法、技术和过程。可组合性验证分为三个层次：静态可组合性、语义可组合性和动态语义可组合性。所提出框架的主要目标是针对需求规范，在这些层次上验证可组合性。建议采用静态分析技术评估前两个层次，而第三个层次则建议采用动态分析技术进行验证。首先，使用状态机匹配技术评估组合组件的行为。若通过这一步，它们将接受以下三种方法之一的检验，以进行动态语义可组合性验证：(i) 代数分析技术；(ii) 状态空间分析技术；或 (iii) 模型检验。选择哪种方法取决于模型的性质。第10章将提供一些指南，说明如何选择合适的方法。

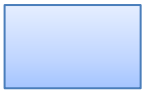
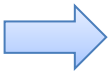
当整个可组合性验证过程成功完成时，这意味着基于 BOM 的组合模型在结构和行为上是一致的，在句法、语义和动态语义层面上具有可组合性，并且符合给定的需求规范。

第6章 可组合性验证过程

Chapter 6
Composability Verification Process

第5章主要介绍了我们提出的可组合性验证框架的规范，包括对各个模块、其运行机制以及所执行程序的详细说明。本章将介绍如何使用该框架，也可将其用作可组合性验证框架的操作手册。本章最后还会根据给定的输入，就如何选择合适的方法提供必要的建议。

以下流程图中所用形状的说明如下：

对象，数据，模型，组件等。		任何这种颜色的形状表示一个 3rd 方工具	
以下对象的列表、集合或集对象、数据、模型		Stop 意味着过程失败。	
过程或操作		继续表示成功，因此，过程与实施阶段	
迭代过程。		比较两个对象。	
扩展或转换对象的		比较多个对象	
扩展或转换许多对象的			
注释			
数据、过程、信息 flow			
页连接符			
重复过程 (Go to 上一步)			

6.1 可组合性验证过程

图 43 至 **图 53**以流程图的形式展示了可组合性验证过程。（图中所示步骤将在本节后文中说明）：

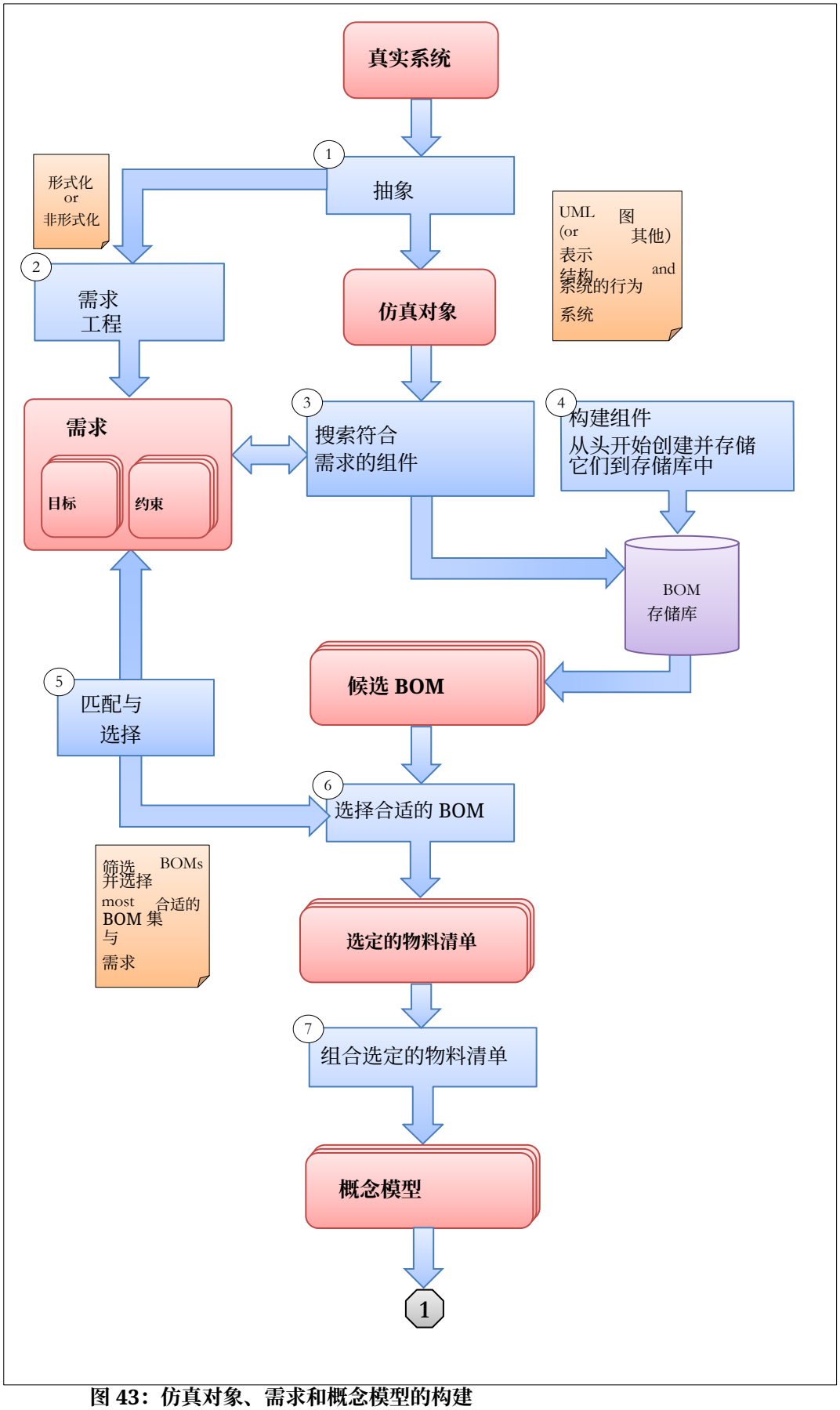


图 43：仿真对象、需求和概念模型的构建

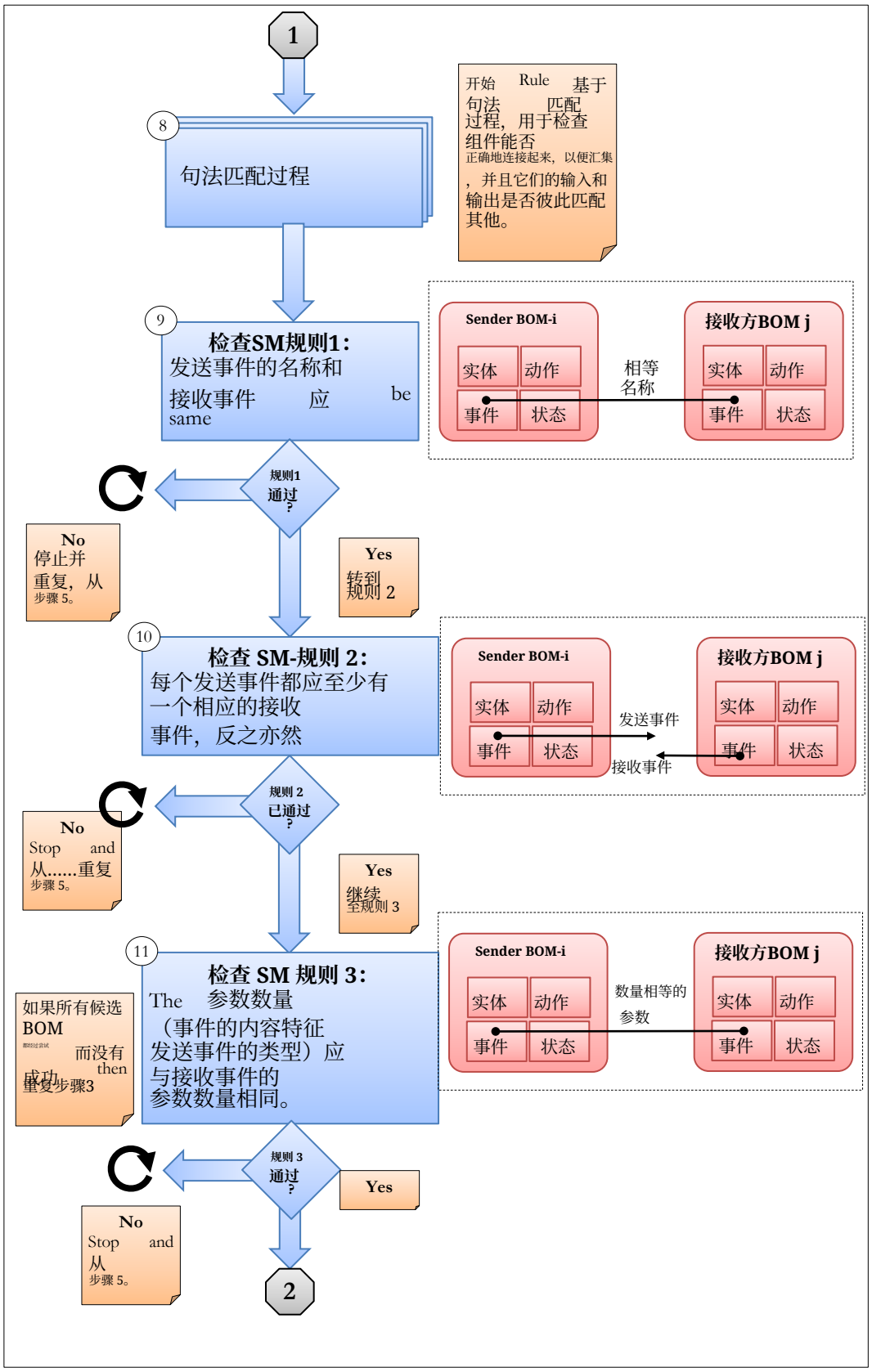

1
8
句法匹配过程
开始 Rule 基于句法 匹配过程，用于检查组件能否正确地连接起来，以便汇集，并且它们的输入和输出是否彼此匹配其他。
9
检查SM规则1：
发送事件的名称和
接收事件 应 be same
Sender BOM-i
实体
动作
事件
状态
相等名称
接收方BOM j
实体
动作
事件
状态
规则1通过？
No
停止并重复，从步骤 5。
Yes
转到规则 2
10
检查 SM-规则 2：
每个发送事件都应至少有一个相应的接收事件，反之亦然
Sender BOM-i
实体
动作
事件
状态
发送事件
接收事件
接收方BOM j
实体
动作
事件
状态
规则 2已通过？
No
Stop and 从……重复步骤 5。
Yes
继续至规则 3
11
检查 SM 规则 3：
The 参数数量（事件的内容特征发送事件的类型）应与接收事件的参数数量相同。
如果所有候选 BOM 而没有成功，then 重复步骤3
Sender BOM-i
实体
动作
事件
状态
数量相等的参数
接收方BOM j
实体
动作
事件
状态
规则 3通过？
Yes
No
Stop and 从步骤 5。
2

图 44：句法匹配过程

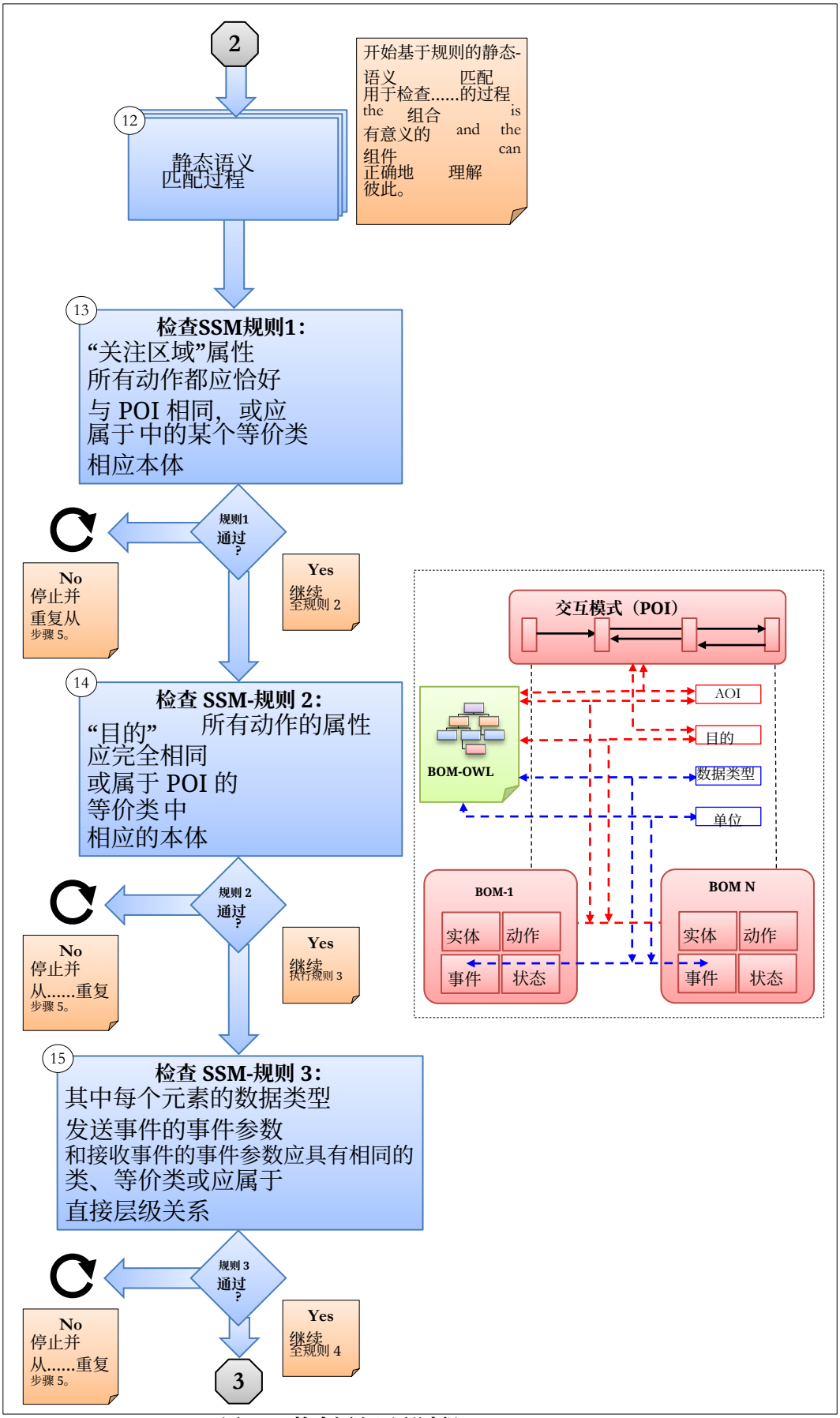


图 45：静态语义匹配过程

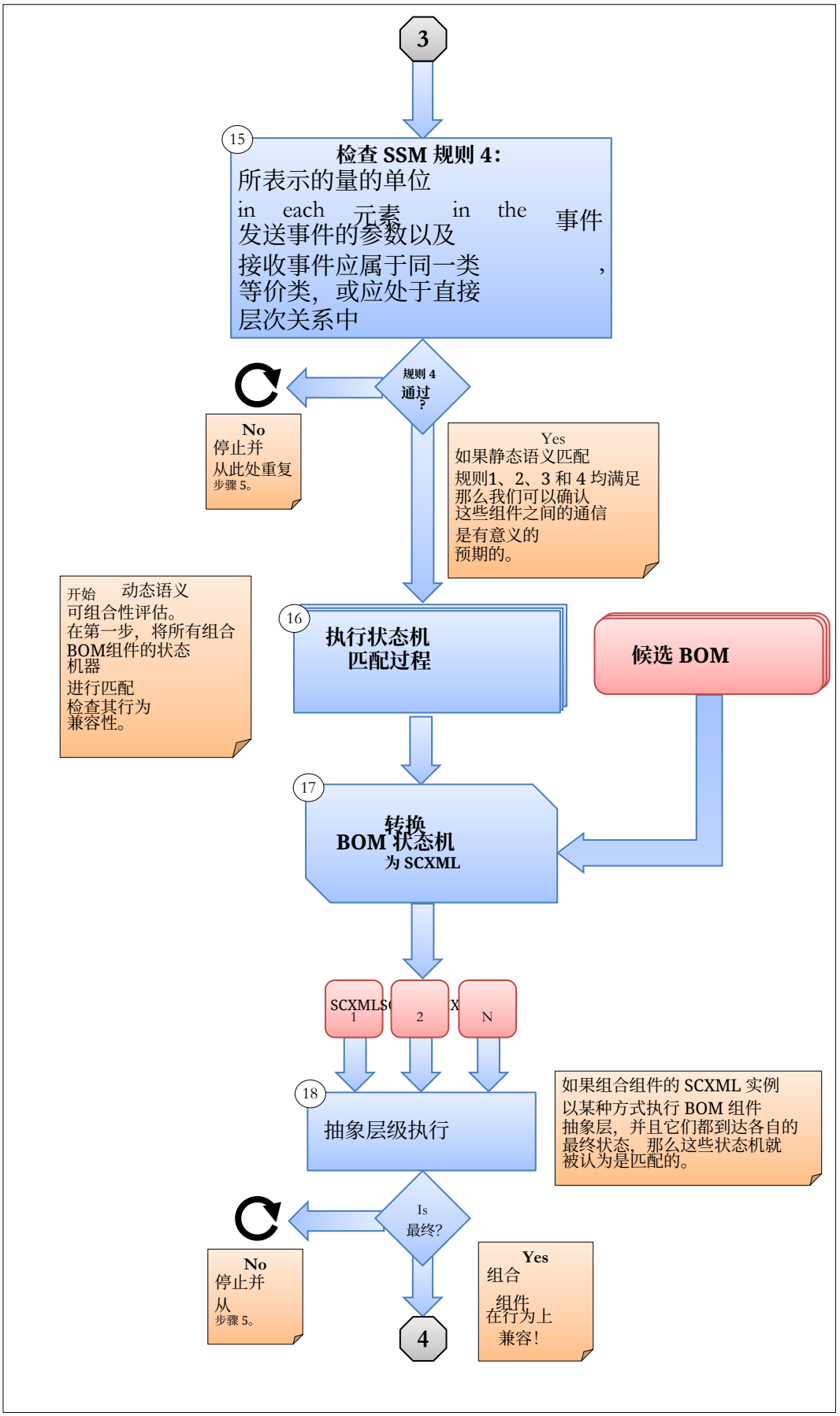

3
15
检查 SSM 规则 4：
所表示的量的单位
in each 元素 in the 事件
发送事件的参数以及
接收事件应属于同一类
，
等价类，或应处于直接
层次关系中
规则 4
通过?
No
停止并
从此处重复
步骤 5。
Yes
如果静态语义匹配
规则1、2、3 和 4 均满足
那么我们可以确认
这些组件之间的通信
是有意义的
预期的。
开始 动态语义
可组合性评估。
在第一步，将所有组合
BOM组件的状态
机器
进行匹配
检查其行为
兼容性。
16
执行状态机
匹配过程
候选 BOM
17
转换
BOM 状态机
为 SCXML
SCXML
1
2
X
N
18
抽象层级执行
如果组合组件的 SCXML 实例
以某种方式执行 BOM 组件
抽象层，并且它们都到达各自的
最终状态，那么这些状态机就
被认为是匹配的。
Is
最终?
No
停止并
从
步骤 5。
Yes
组合
组件
在行为上
兼容!
4

图 46：状态机匹配过程

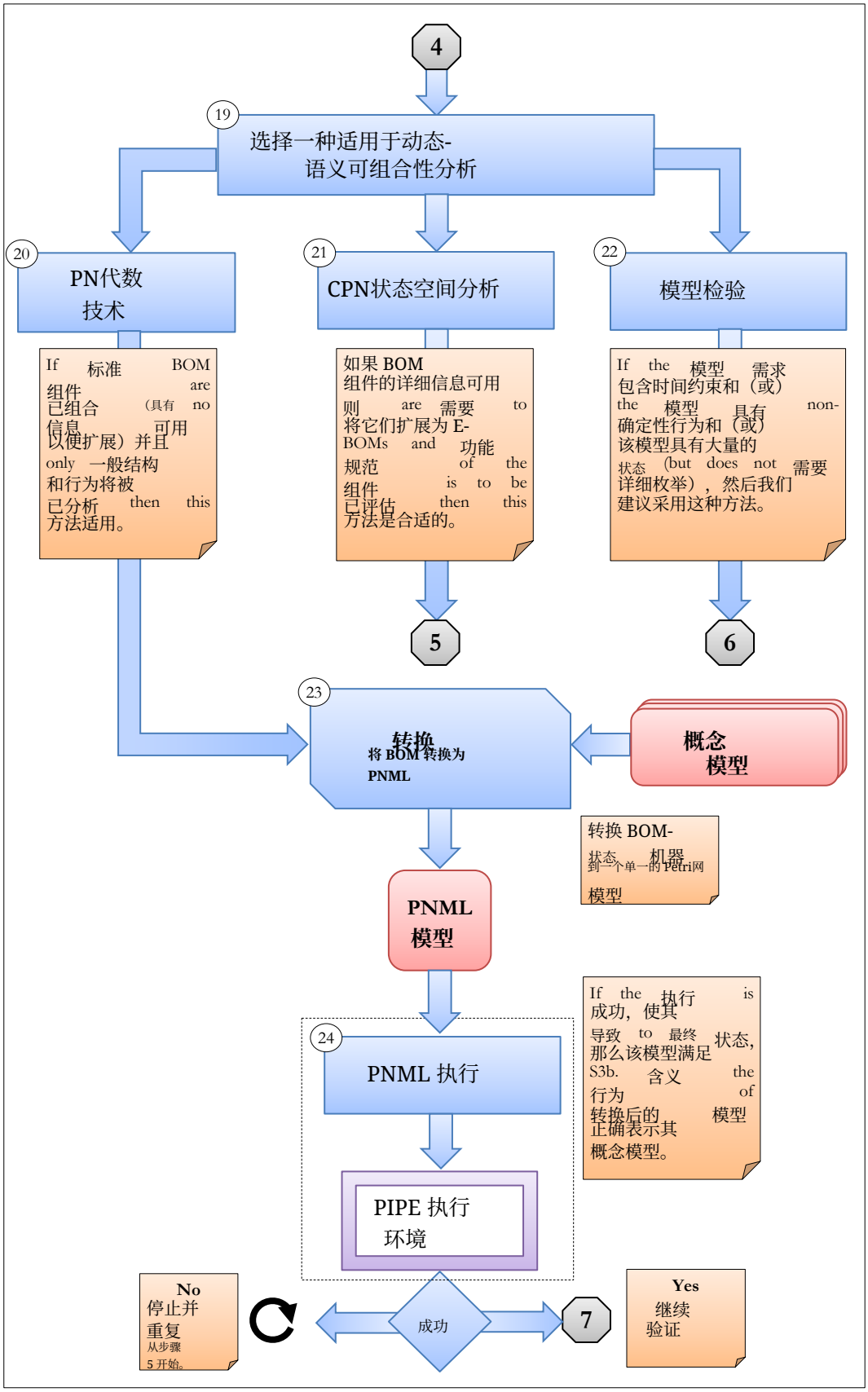


图47：方法选择 | PN代数技术

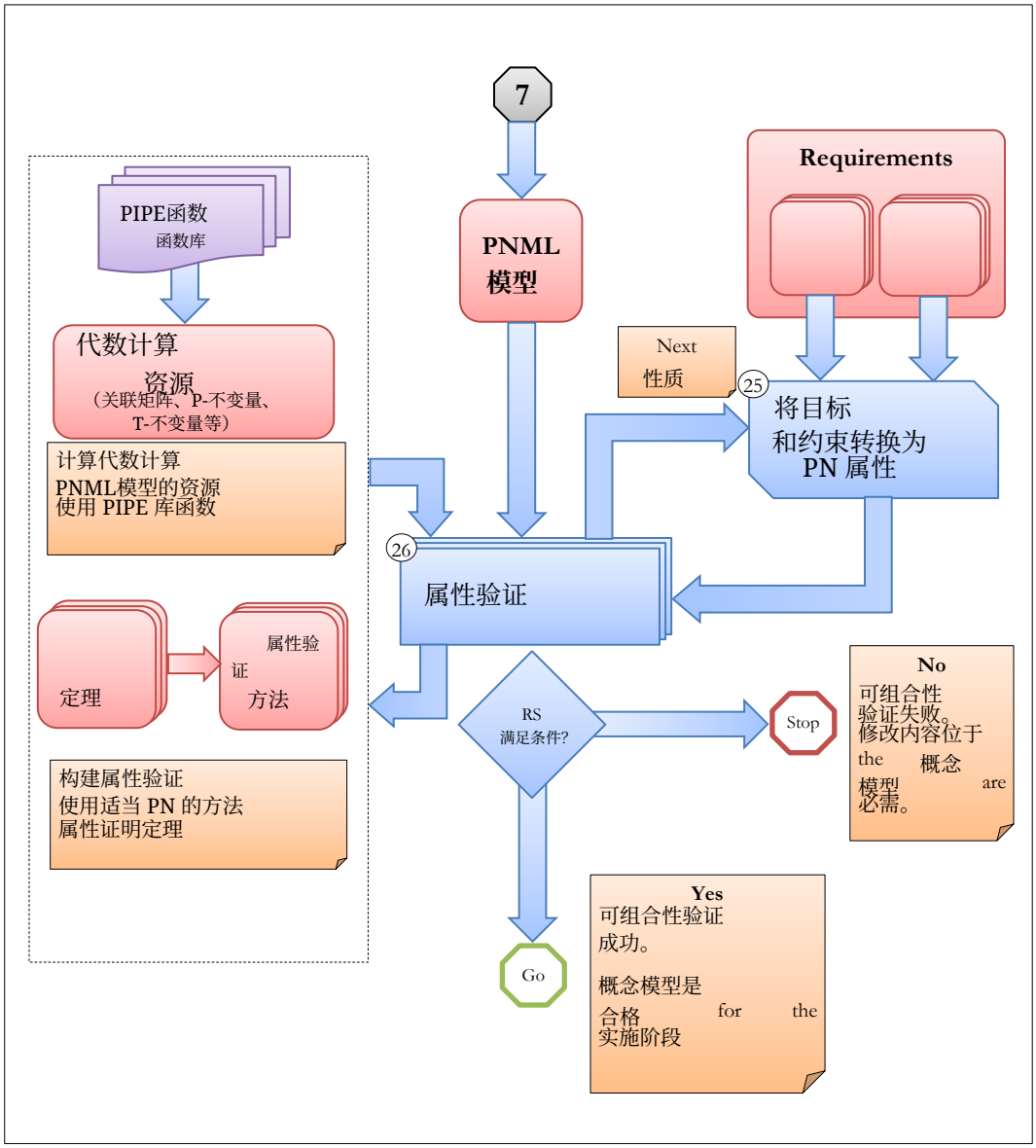


图 48：PN代数技术（续）

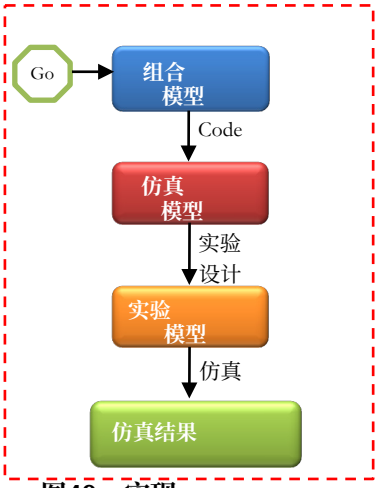


图49：实现

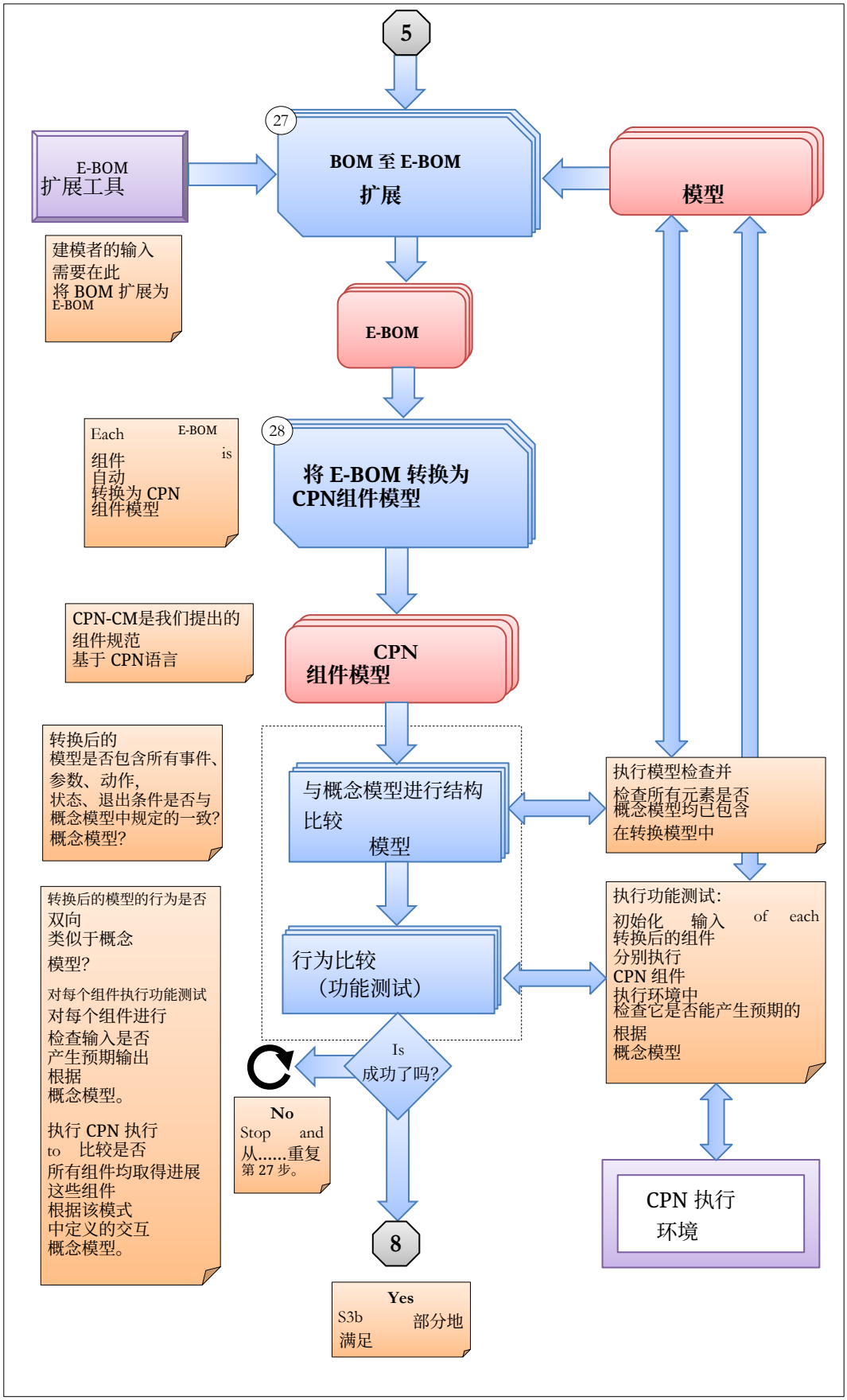


图50：状态空间分析技术

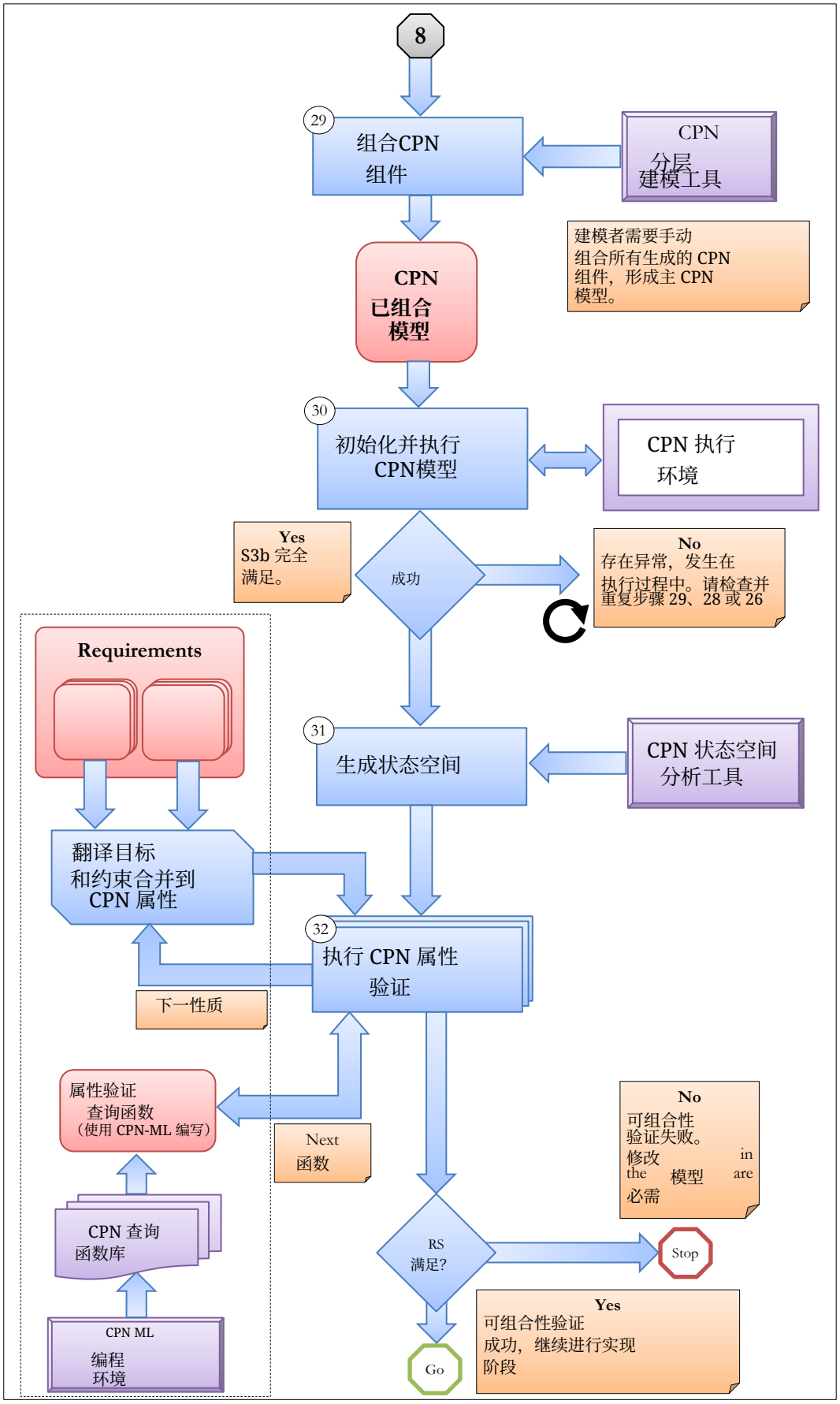


图51：状态空间分析技术（续）

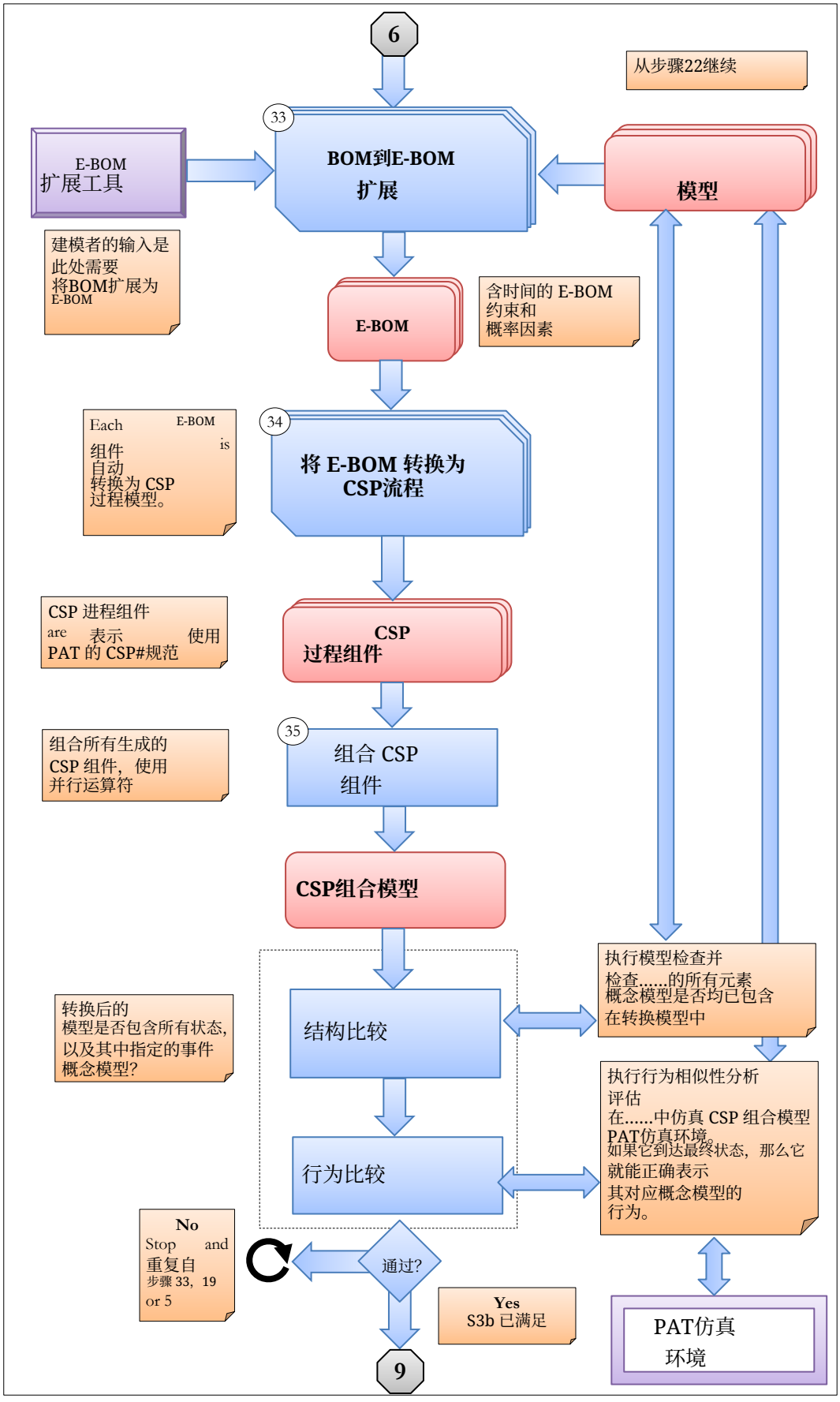

6
从步骤22继续
33
BOM到E-BOM
扩展
E-BOM
扩展工具
模型
建模者的输入是
此处需要
将BOM扩展为
E-BOM
E-BOM
含时间的 E-BOM
约束和
概率因素
34
将 E-BOM 转换为
CSP流程
Each
E-BOM
组件
is
自动
转换为 CSP
过程模型。
CSP 进程组件
are
表示
使用
PAT 的 CSP#规范
CSP
过程组件
35
组合所有生成的
CSP 组件，使用
并行运算符
组合 CSP
组件
CSP组合模型
执行模型检查并
检查.....的所有元素
概念模型是否均已包含
在转换模型中
转换后的
模型是否包含所有状态，
以及其中指定的事件
概念模型?
结构比较
执行行为相似性分析
评估
在.....中仿真 CSP 组合模型
PAT仿真环境。
如果它到达最终状态，那么它
就能正确表示
其对应概念模型的
行为。
行为比较
No
Stop
and
重复自
步骤 33，19
or 5
通过?
Yes
S3b 已满足
PAT仿真
环境
9

图52：模型检验

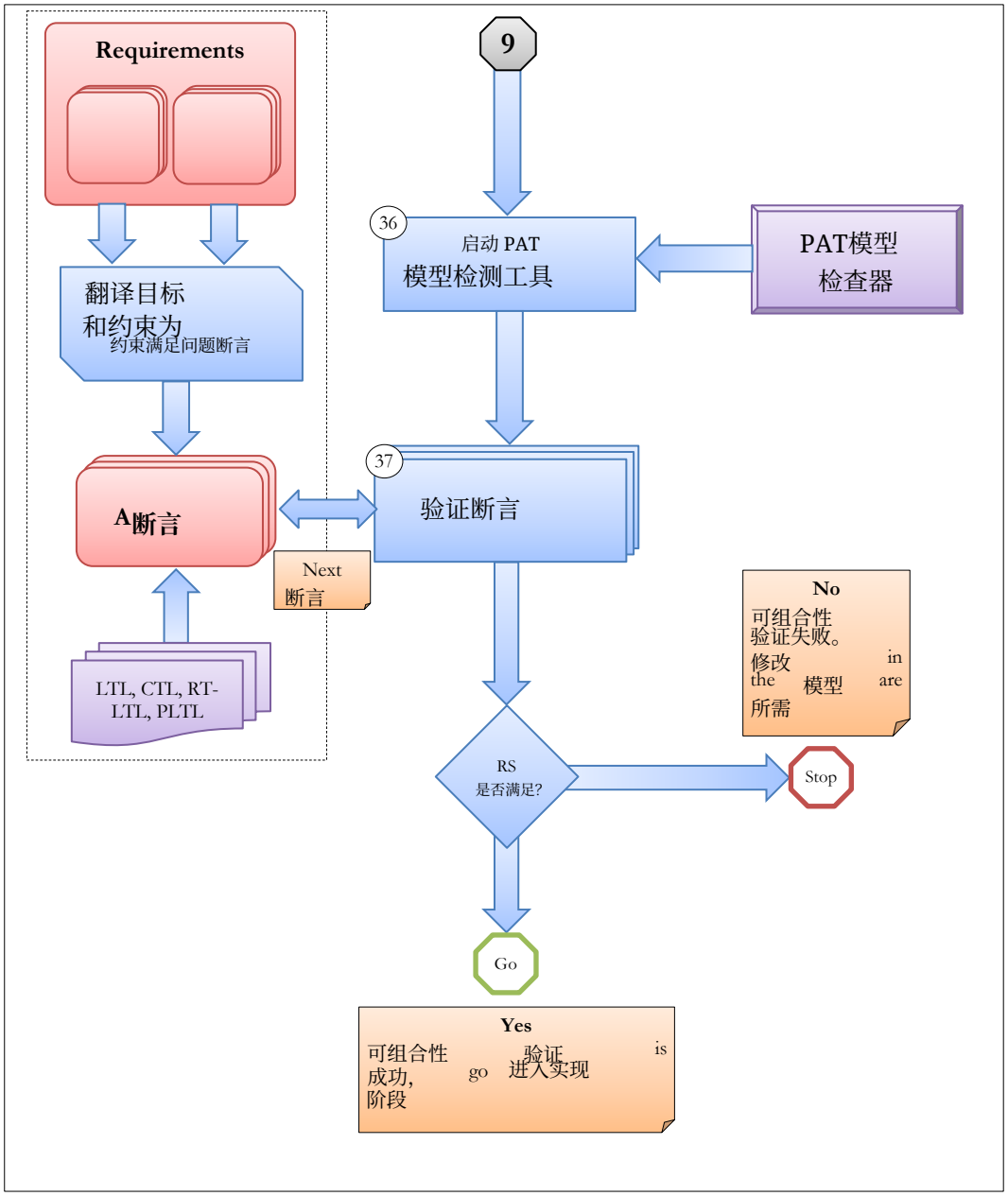


图 53：模型检验（续）

可组合性验证过程说明如下：

6.1.1 仿真对象、需求和概念模型的构建

在步骤1中，对真实系统进行研究，并制定合适的仿真对象。它可以用形式化或非形式化的方式描述。我们假设使用 UML图来描述仿真对象。同时还要研究该系统，以收集需求，并使用我们提出的形式化需求规范方法来制定需求（步骤2）。掌握这些信息后，在 BOM仓库中搜索合适的组件，并假设这些组件的组合将构成一个代表仿真对象的概念模型（步骤3）。如果找不到所需的组件，

则从头构建该组件，将其添加到存储库中，然后在当前语境中使用（步骤4）。

发现的组件称为候选组件。在这些候选项中，会选出最合适的组件，即与仿真对象和给定需求最匹配的组件（步骤 5、6）。

这些 BOM组件经过组合后，会构建出一个概念模型（步骤 7）。我们建议建模者还使用我们提出的 BOM形式化体系和图形表示法，为该概念模型创建形式化模型。这将有助于记录和理解概念模型及其组合的细节。

6.1.2 句法匹配过程

验证过程启动时，会将组合BOM模型（概念模型）输入基于规则的静态分析器，以在语法层面验证其可组合性（步骤 8-11）。如果通过此层面的验证，则满足约束S1（定义见**表 10**），只有此时模型才会进入下一步（否则，验证过程将停止，并重新选择另一个候选项进行组合，再重新执行此步骤）。

6.1.3 静态语义匹配过程

在下一步中，使用语义分析器在静态语义层面对组件进行分析（步骤 12-15）。通过此步骤后，即满足约束S2（定义见**表 10**），BOM组合便可进入动态语义层进行验证。

6.1.4 状态机匹配过程

在此层面，首先使用状态机检查器对BOM组件执行状态机匹配（步骤 15 – 18）。成功完成状态机匹配即满足约束S3a（定义见**表 10**）。

6.1.5 动态语义可组合性验证的方法选择

在下一阶段，验证框架为动态语义可组合性层的分析提供三种验证技术选择。如果没有可用于将BOM组件扩展为E-BOM的信息，建模者可以选择代数技术。因此，概念模型将被转换为PNML，而无需进行任何扩展。如果建模者拥有将BOM转换为E-BOM所需的详细信息和数据，且模型不表示实时系统，则强烈建议选择第二种提出的方法（CPN状态空间分析）。如果模型表示实时系统，并且其本质具有随机性，则建模者应选择第三种方法（模型检验）。这些只是一般指南，并非具体规则。方法的最终选择取决于模型的性质、需求规格属性的性质以及可用信息。

6.1.6 PN代数技术

选择代数技术后，首先将概念模型转换为PNML模型（步骤23）。该PNML模型在PIPE执行环境中执行，以评估S3b（步骤24）。如果评估成功，则逐一获取需求规格属性，并将其转换为PN属性（步骤25）。随后选择一个能够证明该PN属性的属性证明定理。基于该定理，以算法的形式构造属性验证方法。运行该算法可以证明或证伪需求规格属性（步骤26）。如果需求规格中的所有属性都得到满足，则模型验证成功；否则，停止该过程并进行模型细化。

6.1.7 状态空间分析技术

选择 CPN 状态空间分析技术时，首先将每个 BOM组件扩展为 E-BOM（步骤27）。此步骤需要建模者提供输入，可通过 BOM到E-BOM扩展工具完成。扩展完成后，使用自动转换工具将每个 E-BOM 转换为我们提出的 CPN组件模型（步骤28）。此步骤的输出是一组 CPN组件。在此步骤中，需要通过检查和功能测试方法，对生成的组件与各自的 BOM 进行结构与行为比较。如果比较成功，则需求规范的 S3b 约束得到部分满足。

随后，建模者需要使用 CPN层次化工具，在主模型中组合这些生成的组件（步骤29）。（在主模型中，使用插口绑定每个组件的输入端口和输出端口。）模型组合完成后，使用 CPN 执行环境执行模型（步骤30），以测试所有组件是否正确地相互交互并取得必要进展，从而到达最终状态。如果执行成功，则约束S3b 得到完全满足，这表明可执行模型的结构和行为正确地表示了其相应的概念模型，因此，对可执行模型执行的任何验证操作都意味着其概念模型是正确的。
下一步，对 CPN模型进行状态空间分析（步骤 31）。首先生成模型的状态空间图。然后，针对需求规范中的每个目标和约束，从函数库中创建或选取一个验证查询函数。使用 CPN-ML程序执行环境执行该函数，其结果表明该性质是否得到满足或遭到违反（步骤 32）。如果所有性质均得到满足，则称可组合性验证过程成功。

6.1.8 模型检验

选择模型检验技术后，首先将每个BOM组件扩展为E-BOM（步骤 33）。此步骤需要建模者输入，可通过BOM到E-BOM扩展工具完成。可以为状态或变迁指定时间约束和概率因素。扩展完成后，每个E-BOM都会自动转换为CSP#进程规范（步骤 34），然后进行组合（步骤 35）。每个CSP

表示BOM组件的过程，其组合可根据组合组件的性质，使用顺序运算符‘;’、并行运算符‘| |’、交错算子‘| | |’或（非确定性或用户指定的）选择运算符‘[ ]’来完成。我们建议并行组合每个CSP，使每个过程并行执行，并通过各自的通信通道发送或接收事件以彼此同步。随后可使用PAT模拟器对组合模型进行仿真。若一次成功的仿真运行中至少存在一条路径通向最终状态，则表明组合模型的行为正确地表示了其概念模型，因而满足约束S3b。

在下一步骤中，使用PAT模型检查器对以CSP格式定义的断言（使用LTL、实时 LTL或PLTL）进行验证，并得出其满足或违反的结果（步骤3、6、7）。如果所有断言均得到满足，则称可组合性验证过程成功。

6.2 总结

本章展示了可组合性验证过程的流程图。该流程图指出了过程中各个步骤以及每个步骤的输入和输出形式。它可作为使用三种不同方法进行可组合性验证的指南。本章还提出了一些建议，以帮助做出合适的选择。验证过程成功完成后，组合模型便可进入实施阶段，在此阶段使用合适的仿真平台对其进行编程和仿真。此外，还可以构建实验模型，对已实现的模型进行不同的实验，并生成仿真结果以供研究和决策。本论文不涉及实施阶段。

第7章 使用PN代数技术进行公平性验证

Chapter 7
Fairness verification using PN Algebraic Techniques

本章通过一个制造系统的示例，说明代数技术如何帮助验证组合模型的系统性质。在该示例中，公平性被选为所需的系统特征。

7.1 公平性

公平性已在第3.1.3节中根据Petri网的一种性质作了定义。本章将详细介绍公平性这一概念。直观地说，公平性是一种活性性质，意味着系统中任何足够频繁地成为可能（或使能）的组件，都不应被无限期延迟。[122]。
根据充分程度，文献中通常将公平性分为以下三种类型：

无条件公平性
也称为无偏，意指系统中的每个组件都无条件地无限次推进。“推进”是指取得进展（例如，发射一个变迁）。无条件公平性也称为非确定性选择，通常存在于彼此独立的组件之间 [122]。

弱公平性
也称为公正，意指系统中从某个时刻起持续处于使能状态的每个组件最终都会推进。

强公平性
系统中每个无限多次处于使能状态的组件都会无限次推进。弱公平性与强公平性的一项显著区别在于，弱公平性要求想要推进的组件持续处于使能状态，而强公平性不要求组件持续处于使能状态。

文献中存在一些重要的公平性泛化形式 [122]:
均等公平性：意味着给予每个组件相等的继续执行机会。它可以被视为正义。由于优先级政策或其他原因，这种类型的公平性并不始终适用于现实世界的场景。

有界公平性：意味着给予每个组件相等数量的机会，使得任何组件在允许其他组件轮流执行之前，连续执行的次数都不会超过“k 次”。

例如，机场有一项值机服务，同时服务两类队列：(i) 商务舱和 (ii) 经济舱。如果该服务主要服务商务舱乘客，但在不服务经济舱队列中的乘客时，服务商务舱乘客的次数不超过（比如说）10 次，则称其是公平的。

在Petri网中，可以从两个角度看待公平性，即：变迁公平性和标识公平性。前者对应于变迁选择的公平性，后者则涉及状态的公平可达性。

7.2 公平性验证

验证模型公平性的方法有多种。本章重点讨论如何使用PN代数分析验证公平性，并给出PN模型具有公平性的充要条件。在PN模型中评估这些条件涉及定理和线性代数计算，因此将其归类为代数技术。基于以下定理，我们提出了一种自动验证公平性的算法。

在Petri网中，公平性主要根据变迁的发生（或发射）来理解。如果存在一个正整数t_1和t_2，使得对于任意可达标识k M**以及任意发射序列**σ**：**

$$\#(t_1/\sigma) = 0 \Rightarrow \#(t_2/\sigma) \leq k \ \wedge \#(t_2/\sigma) = 0 \Rightarrow \#(t_1/\sigma) \leq k \tag{7.1}$$

（符号#(t/σ)表示变迁t在发射序列σ中发生的次数）

换言之，两个变迁都不应在未让另一个至少发生一次的情况下，发生超过有限次（k）。这称为具有上界 = k 的有界公平性（或 B-公平性）。如果每对变迁之间都存在有界公平关系，则称整个网是公平的 [**123**]。

对于 Petri网模型中公平性性质的代数验证，应用以下定理。这些定理的详细内容及证明见 [**123**]。

定理 I

给定一个关联矩阵为 *A* 的 *Petri*网，如果存在一个发射次数向量 *X*，使得：

$$A.X \geq 0 \text{ and } X \neq 0$$

则 *Petri*网公平的一个必要条件是，*X* 的每个分量均为正。

定理 II

如果 *Petri*网 N 对于任意初始标识 M 均有界，则定理 *I* 中的条件是 *N* 公平的充要条件。

推论：如果存在一个由正整数组成的 *P*-不变量 *Y*，使得：A.Y=0 那么可以保证该 *Petri*网是结构有界的。

定理 III

公平的 *Petri*网 *PN* 至多只有一个再生向量（即一个最小*T*-不变量）。

基于上述有界公平性的定义以及定理 I、II 和 III，若一个 PN 满足以下两个条件，则称其为公平：(i) 给定的 PN 模型中存在一个**单一** T-不变量 X，其每个分量均非零，且乘积 AX = 0；(ii) 至少存在一个 P-不变量，这意味着该网是结构有界的。

7.3 制造系统

本节介绍一个制造系统的基于组件的组合模型。借助该组合模型，本文展示了如何使用所提出的验证框架来验证所需的规范；在本例中，该规范包括作为一项重要质量约束的公平性。本文讨论了本例的两个不同场景。在第一个场景中，验证框架验证出该模型不公平。在第二个场景中，模型经过修改，随后通过验证，并满足公平性约束。

7.3.1 场景 I

假设该制造系统模型由两台机器 M1 和 M2 以及一台共享机器人 R 组成，如图 54所示。机器人将原材料装载到机器上，并在机器上进行加工以生产产品。机器人在任一时刻被分配（“装载”）给其中一台机器。机器人装载后，会将原材料放到机器上并进行加工。产品生产完成后，机器人卸载，并可供另一台机器使用。

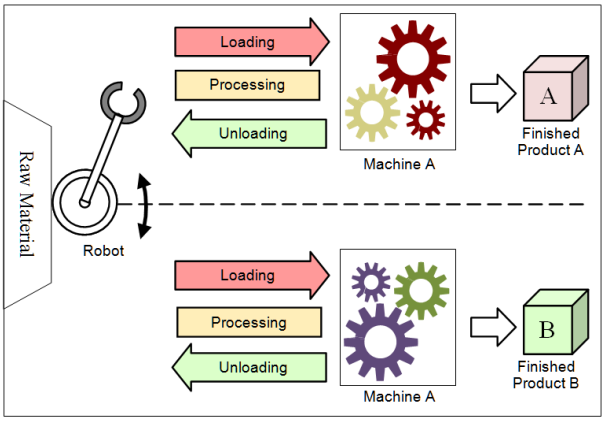


图 54：制造系统（摘自[124]）

可组合性验证过程如下启动：

仿真对象与需求规范

第一步，根据图 54识别仿真对象的实体、事件和状态。根据第6章所述的可组合性验证过程中给出的步骤，利用仿真对象和需求规范构建适当的概念模型。

我们将制造系统的需求规范定义为：

$$\mathbf{RS}_0 = \langle \mathbf{O}, \mathbf{S} \rangle \text{ where:}$$

目标 $O = \{o_1, o_2, o_3\}$

o_1：机器1 应持续生产产品1，不应出现无限延迟o_2：机器2 应持续生产产品2，不应出现无限延迟o_3：两台机器生产产品的比例应为 1:1

系统约束 $S = \{s_1, s_2, s_3, s_4\}$

s_1：机器1、机器2 和机器人组件应能在语法层面进行组合

s_2： 机器1、机器2和机器人组件应能在静态语义层面组合

s_{3a}： 组合模型的状态机匹配应成功。由于这些模型不会终止，因此不存在最终状态；取而代之的是，将考虑目标状态："机器*1*完成生产"和"机器*2*完成生产"。

s_{3b}： 转换后的可执行模型正确表示了概念模型的结构和行为。

s_4：共享机器人应公平地对待两台机器（即 k-公平性；k=1）。

概念模型

参与制造组合模型的每个 BOM 模型的形式化规范和图形表示如下。为便于阅读，形式化定义中使用以下颜色代码来区分不同的 BOM元素：

BB_0 = 〈 **EnT, EvT, S, AcT** 〉， 其中:

EnT = 机器1 {C_0(标识符:整数)}**EvT** = {E_0(LoadingM1, 机器人, 机器1, 空值), E_1(UnloadingM1, 机器人, 机器1, 空值), E_2(ResetM1, 机器人, 机器1, C_0)}**Act** = { A_0(LoadingM1, 机器人, 机器1, E_0), A_1(UnloadingM1, 机器人, 机器1, E_1), A_2(ResetM1, 机器人, 机器1, E_2)}S = {S_0(M1等待, A_0, S_1), S_1(M1处理中, A_1, S_2), S_2(M1已完成, A_2, S_0)}

表 21：Machine1 基础 BOM 的形式化定义

BB_1 = 〈 **EnT, EvT, AcT, S** 〉， 其中:

EnT = 机器2 {C_1(标识符:整数)}**EvT** = {E_3(加载M2, 机器人, 机器2, 空值), E_4(卸载M2, 机器人, 机器2, 空值), E_5(重置机器2, 机器人, 机器2, C_1)}**Act** = { A_3(加载M2, 机器人, 机器2, E_3), A_4(卸载M2, 机器人, 机器2, E_4), A_5(重置机器2, 机器人, 机器2, E_5)}S = {S_3(机器2等待中, A_0, S_4), S_4(机器2处理中, A_1, S_5), S_5(机器2已完成, A_2, S_3)}**T**

able 22: Formal definition of Machine2 Base-BOM

BB_2 = 〈 EnT, EvT, AcT, S 〉 其中:

EnT = 机器人 {}

EvT = {E_6(加载M1, 机器人, 机器1, 空值), E_7(卸载M1, 机器人, 机器1, 空值), E_8(加载M2, 机器人, 机器2, 空值), E_9(卸载M2, 机器人, 机器2, 空值)}

行为 = {A_6(加载M1, 机器人, 机器1, E_6), A_7(卸载M1, 机器人, 机器1, E_7), A_8(加载M2, 机器人, 机器2, E_8), A_9(卸载M2, BB_2, 机器2, E_9)}

S = {S_6(空闲, {A_6, S_7},{A_8, S_7}), S_7(忙碌, {A_7, S_6}, {A_9, S_6})}

表23：机器人基础 BOM 的形式化定义

组合式基础对象模型$_0$ = 〈 AcT $_{IN}$, AcT $_{OUT}$, POI 〉 其中:

AcT_{IN} = AcT $_{OUT}$ = ∅

POI = {POI_0(!A_6 , ?A_0), POI_1(!A_7 , ?A_1), POI_2(!A_2, ?A_2), POI_3(!A_8 , ?A_3), POI_4(!A_9 , ?A_4), POI_5(!A_5, ?A_5) }

表24：制造系统组合BOM的形式化定义

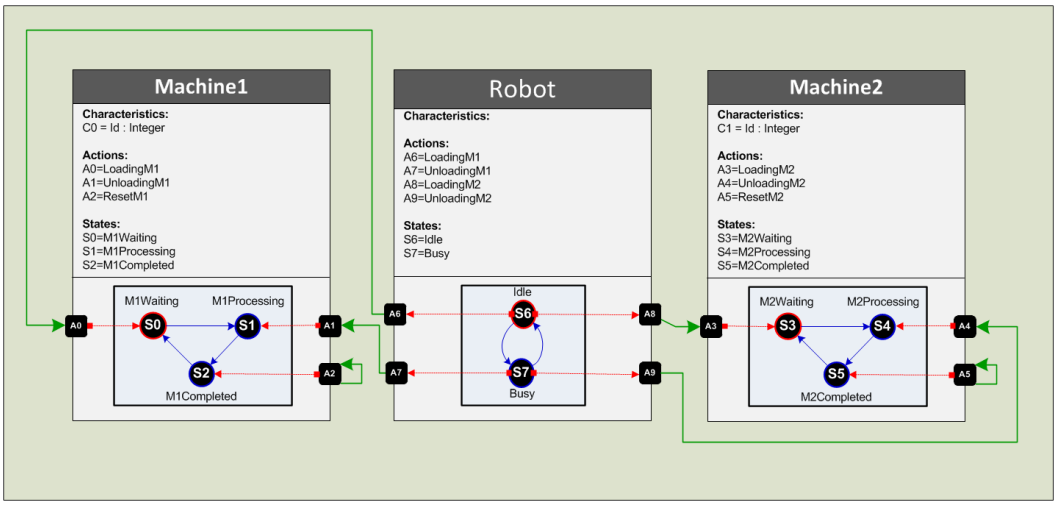


图55：基于制造系统BOM的组合模型

图 55 展示了该制造系统基于 BOM 的概念模型，其中包含三个 BOM，并使用我们提出的图形表示法进行了形式化定义。该图展示了特征、事件、动作和状态之间的映射关系（使用红色虚线表示）。在机器1中，特征 C_0 映射到事件 E_2 ，这意味着事件使用特征 C_0 作为参数。类似地，事件 E_0 分别映射到 A_0、、E_1 映射到 A_1 ，以及 E_2 映射到 A_2 ，这意味着动作使用各自映射的事件。图中动作到状态的映射展示了哪个动作会导致哪个状态转移到新状态（以蓝色箭头表示）。该

基本BOM组件通过表24所示的形式化定义彼此连接，该定义描述了从一个组件到另一个组件的动作的源（！）和目的地（？）。在图55中，这通过带有输入/输出（I/O）标签的黑色箭头线表示。这称为交互模式（在BOM规范中）。

静态分析

规则	机器1	机器2	机器人
● 发送事件的名称和接收事件的名称应相同 ● 每个发送事件都应至少有一个对应的接收事件，反之亦然 • 发送事件的 数量 of 参数（事件类型的内容特征）应与……数量相同 参数 of the 接收-事件。	?LoadingM1(空值)		!LoadingM1(空值)
	?UnloadingM1(空值)		!UnloadingM1(null)
		?LoadingM2(null)	!LoadingM2(null)
		?卸载M2(空值)	!卸载M2(空值)

表25：语法匹配

从表25可以看出，发送事件和接收事件的名称相同（!=Send，?=Receive）。二者之间是一对一关系。此外，每个事件的参数数量均等于1。基于这些事实，可以说这些组件在语法上是可组合的（S1 已满足）。

我们假设机器1、机器2和机器人组件具有表26中所示的语义属性，并满足所有静态语义匹配规则。
以红色突出显示的属性在语义上等价（完全匹配），因此满足 S2。

机器1	机器2	机器人
AOI = {生产、制造，生产线，车削}	AOI = {生产，制造，生产线，抛光}	AOI = {生产，制造，输送机，自动化}
目的 = {制造产品1，制造产品3}	目的 = {制造产品2，制造产品3}	目的 = {制造产品3}
参数的数据类型= {空值}	参数的数据类型 = {空值}	参数的数据类型 = {null}
计量单位 = {}	计量单位 = {}	计量单位 = {}

表26：静态语义匹配

动态分析

状态机匹配过程成功完成，因为机器1和机器2均到达其目标状态，即：*Mcompleted*和*M2-completed*，并满足 S3a。

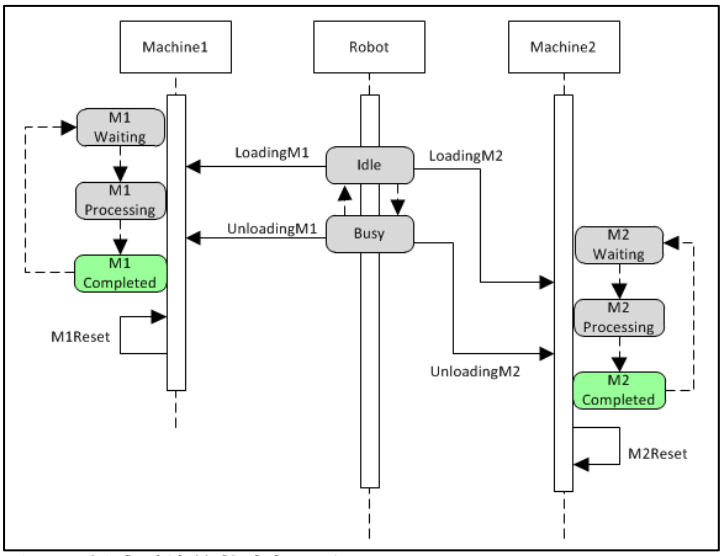


图56：制造系统的状态机匹配

BOM到PNML的转换

下一步，将对组件执行 PNML 转换过程。转换过程的输出是一个 Petri网模型，如**图57**所示。通过检查可以看到，转换后的模型中包含状态及其退出条件、事件和动作（与原始概念模型中的定义一致）。此外，该 Petri网模型还在 PIPE运行环境中执行。此次执行成功，因为库所 P3 和 P6 获得了令牌（表明在执行过程中到达了这些目标状态）。这满足 S3b。

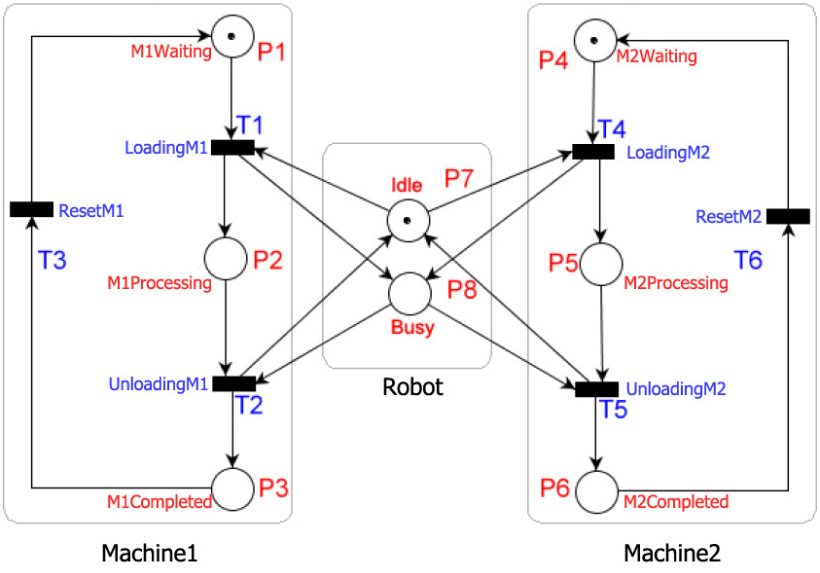


图57：制造系统的Petri网模型

代数资源计算

在此步骤中，使用 PIPE库 的库函数计算图5图57所示 PN组合模型的初始标识 M_0 和关联矩阵A，如下所示：

M_0	P1	P2	P3	P4	P5	P6	P7	P8
	1	0	0	1	0	0	1	0

A	P1	P2	P3	P4	P5	P6	P7	P8
T1	-1	1	0	0	0	0	-1	1
T2	0	-1	1	0	0	0	1	-1
T3	1	0	-1	0	0	0	0	0
T4	0	0	0	-1	1	0	-1	1
T5	0	0	0	0	-1	1	1	-1
T6	0	0	0	1	0	-1	0	0

表27：初始标识和关联矩阵（场景I）

请注意，A 中的行和列标签以及M_0中的元素分别对应图57中的库所和变迁。矩阵A 作为输入传递给不变量计算模块，该模块计算出制造系统Petri网模型中的以下P-不变量和T-不变量：

P1	P2	P3	P4	P5	P6	P7	P8
1	1	1	0	0	0	0	

T1	1
T2	1
T3	1
T4	0
T5	0
T6	0

T1	0
T2	0
T3	0
T4	1
T5	1
T6	1

表28：P-不变量和T-不变量（场景 I）

性质验证函数

为了继续进行验证，我们必须将需求规范中的目标和约束转换为 Petri 网属性：

o_1： 机器1应持续生产产品1，且不存在无限延迟o_2： 机器2应持续生产产品2，且不存在无限延迟o_3： 两台机器应按 1:1 的比例生产产品s_4： 共享机器人应公平地服务两台机器（即 k-公平性；k=1）。

从 {o_1, o_2, o_3 and s_4} 可以清楚看出，只有当机器人以公平的方式（S4）为两台机器提供服务时，它们才能持续生产各自的产品，而不会出现无限期延迟（O1 & O2）。并且，如果公平性受到 k=1 的限制，那么两台机器生产产品的比例将相等，为 1:1。

因此，需求规范的 Petri网形式翻译如下：

PN 属性 P1 = “模型应为有界公平（*K=1*），使机器人交替为两台机器提供服务”。

为了验证有界公平性，我们考虑第7.2节中定义的性质证明定理 I、II 和 III。基于这些定理，我们使用以下算法构造一个性质验证函数：

算法：B-公平性验证
输入： {P-Invariants}, {T-Invariants}, A; 输出： TRUE
1 如果 |{T-Invariants}| = 1 则 ⊳ T-不变量列表-不变量恰好有 1 个，这意味着它是一个 2 再生向量，定理 III⊲ 3 $X_T \leftarrow$ T-不变量[0] ⊳ 从列表中获取唯一的 T-不变量 4 如果$A.X_T \geq 0$ 且 X 中的每个元素$_T$ >0 则⊳ 将 X 相乘$_T$ 与关联矩阵相乘并 5 检查 T-不变量的每个元素是否为 6 正数，定理 I⊲ 7 如果 |{P-不变量}|>0 则⊳ 检查是否存在任何 P-不变量，即 PN 8 模型有界 , 定理 II⊲ 9 返回TRUE 10 else 11 返回FALSE ⊳ 违反定理 II 12 结束 if 13 else 14 返回FALSE ⊳ 违反定理 I 15 结束 if 15 否则 17 返回 假 ⊳ 违反定理 III 18 结束 if

表 29：B-公平性验证

基于此算法，我们对给定的 Petri网模型进行性质验证。显然，T-不变量（见**表28**）中含有零项，违反了定理 I。此外，违反定理 III 的 T-不变量不止一个，因此该网被认为是不公平的。由于该 Petri网不公平，无法保证目标o_1, o_2, o_3能够得到满足，因为任一机器都可能多次获取机器人，而不给另一台机器至少一次获取机器人的机会，从而过度运行（o_1 & o_2未满足）。因此，任一机器都可能面临无法生产足够数量产品以实现所需目标的情况，即无法满足生产产品的 1:1 比例（o_3未满足）；因此，组合模型可能无法满足给定的规格。

7.3.2 场景 II

为了理解公平性验证过程，这里给出一个反例。在此示例中，组合中增加了另一个名为控制器的组件，用于监督机器人分配。控制器的任务是在系统中确保公平性。 控制器的 BOM 模型定义如下：

BB_3 = 〈 **EnT、EvT、AcT、S** 〉，其中：

EnT = 控制器 {}

EvT = {E_{10}(加载M1, 控制器, 机器人 , 机器1, 空值), E_{11}(加载M2, 控制器, 机器人, 机器2, 空值)}

执行 = {A_{10}(加载M1, 控制器, 机器人, 机器1, E_0), A_{11}(加载M2, 控制器, 机器人, Machine2, E_1)}

年代 = {年代$_8$(机器1分配, {A_{10}, 年代$_9$}), 年代$_9$(机器2分配, {A_{11}, 年代$_8$})}

表3：Controller 基础 BOM 的形式化定义

组合式基础对象模型$_6$ = 〈 $\mathbf{AcT}_{IN}$, $\mathbf{AcT}_{OUT}$, **POI** 〉 其中：

$\mathbf{AcT}_{IN} = \mathbf{AcT}_{OUT} = \varnothing$**POI** = {$POI_0$(**!$A_{10}$** , {?$A_0$, ?$A_6$}), POI_1(**!A_7**, ?A_1), POI_2(**!A_2**, ?A_2), POI_3(**!A_{11}** , {?A_3, ?A_8}), POI_4(**!A_9** , ?A_4), POI_5(**!A_5**, ?A_5) }

表31：修改后的制造系统组合BOM的形式化定义

其他组件的定义相同，唯一的区别是，事件 E_0 和 E_1 的发送方为 BB_3 （控制器），而事件 E_0 的接收方为 BB_0 （机器1）和 BB_2 （机器人）；事件 E_1 的接收方则为 BB_1 （机器2）和 BB_2 （机器人）。**图58**展示了修改后的制造系统的组合BOM。

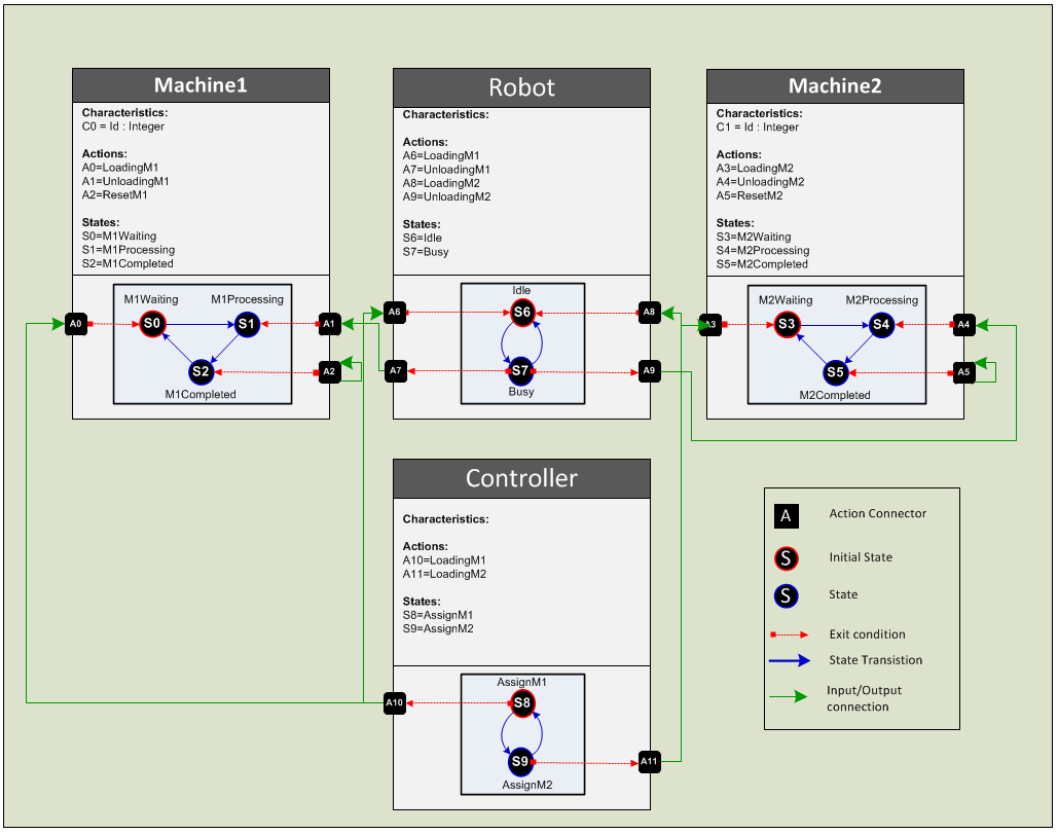


图58：修改后的制造系统组合BOM

启动验证过程后，BOM组件将转换为Petri网模型，如图59所示，其中控制器组件与两台机器和机器人相连，并控制机器分配以确保 K公平性。从图中可以明显看出，机器人一旦被分配给机器1，就不能再重新分配（因为 P9 中缺少令牌），反之亦然。如果将令牌数量增加到“n”，同一模型便可实现 k=n 公平性。

在初始化阶段，初始标识 $\mathbf{M_0}$ 和关联矩阵 A计算如下：

M_0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
	1	0	0	1	0	0	1	0	1	0

A	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
T1	-1	1	0	0	0	0	-1	1	-1	1
T2	0	-1	1	0	0	0	1	-1	0	0
T3	1	0	-1	0	0	0	0	0	0	0
T4	0	0	0	-1	1	0	-1	1	1	-1
T5	0	0	0	0	-1	1	1	-1	0	0
T6	0	0	0	1	0	-1	0	0	0	0

表32：初始标识与关联矩阵（场景 II）

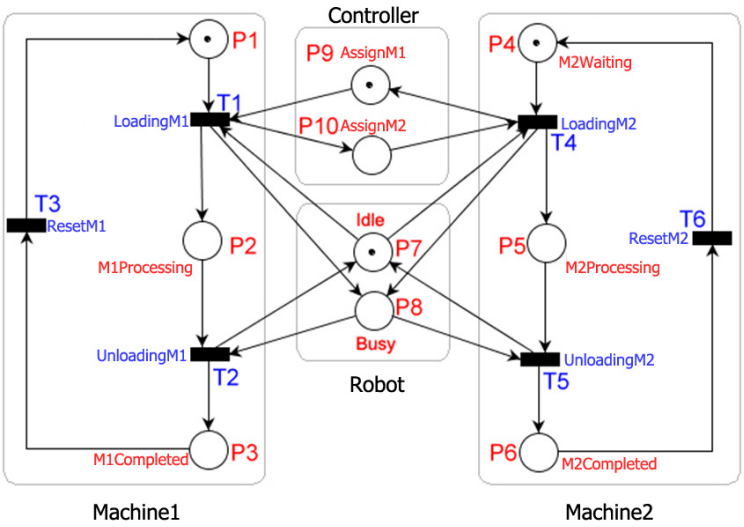


图59：制造系统的修改后的Petri网模型

运行不变量计算模块时，在 图59 所示的模型中发现了以下 T-不变量和 P-不变量：

T1	1
T2	1
T3	1
T4	1
T5	1
T6	1

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	1	1	0	0	0	0	0	0	0

表33：P-不变量和 T-不变量（场景 II）

只有一个（也是唯一一个）具有非零项的 T-不变量，并且存在一个（含有若干非零项的）P-不变量，满足模型实现有界公平所需的条件（定理 I、II 和 III 的条件）。

基于 PN代数分析技术得出的这些结果，我们可以确认组合模型满足给定的需求规范。由于受控控制器，机器人必定会在两台机器之间公平运行，从而实现目标 O_1, O_2 和 O_3 ，并满足所需约束 S_1。

7.4 总结

公平性在组合模型的可组合性验证中至关重要，因为它可以防止某个组件占据主导地位并过度执行，而其他组件却一次也不执行。正如制造系统的示例所示，机器人分配的公平性可以确保两台机器都能运行，以生产所需数量的产品。如果缺乏公平性，我们就无法保证实现这一目标。

我们以制造系统中的公平性验证为例，说明我们的代数验证技术如何发挥作用。值得注意的是，这种技术不会面临状态空间爆炸，因为它不涉及构建可达图，只需计算关联矩阵和P/T不变量即可。许多PN 属性都可以利用这些PN代数计算资源进行验证。另一方面，这种方法只能用于验证有限范围的PN 属性（即存在适用定理的属性）。

第8章 使用状态空间分析技术进行模型验证

Chapter 8
Model Verification using State-space Analysis techniques

有色*Petri*网及其分析技术对于实现准确、高效的验证非常有用，因为它是并发系统规范领域中颇具竞争力的形式化方法之一。将其应用于可组合性验证，尤其关注动态语义可组合性级别时，效果十分显著。*CPN*社区数十年来提出的分析技术，显著提升了针对模型正确性进行高效、准确推理的能力。本章以一个野战炮兵模型为例，介绍如何将基于*BOM*的野战炮兵模型转换为我们提出的着色*Petri*网组件，并通过状态空间分析对其进行验证。

作战建模涉及描述或表示武器系统和作战情境的模型。作战模型有许多类型，这些类型根据其建模目标加以区分。作战建模的一些基本目标包括训练、兵棋推演、武器测试等。

8.1 作战建模

作战建模有意对作战实体及其行为、活动和相互关系进行抽象和简化，以回答国防相关研究问题。不过，不存在能够回答所有问题的通用模型；但有一个通用情境环境的概念，以及在每个战场上都能找到的四项核心活动 [125]。

8.1.1 情境环境

作战建模始于分析对情境环境进行建模所面临的挑战。所有建模的作战实体都处于环境中，也就是虚拟战场。它们感知环境（包括其他实体），并根据自身的知识和目标将感知映射为内部表示。它们在环境中与其他作战实体通信并采取行动。环境包含所有对象，既有障碍物等被动对象，也有敌人或友方单位等主动对象 [125]。

8.1.2 移动

移动是作战建模的核心活动，涉及单个实体的移动。这些实体可以是武器、人员等，也可以是用于建模实体群体移动的聚合模型。模型使用区块和网格；它们为武器系统使用物理模型，并为单位移动使用参考模式 [125]。

热传感器和光学传感器能够帮助以尽可能贴近现实的方式感知环境和其他实体。情报、监视与侦察行动也有助于满足类似需求。为了感知实体的特殊属性，需要对这些特殊属性逐一进行显式建模。如果某项属性被显式建模，就必须使其在侦察过程中产生影响。此外，如果某个细节对军事决策过程很重要，就必须将其纳入感知范围，因此需要由传感器进行观察；这要求将相应事物建模为实体的属性 [125]。

8.1.4 射击

对武器系统之间的决斗以及单位之间的战斗结果进行建模，仍是一个备受关注的主题。在武器系统层面，会分析直射火力和间接射击。直射火力是指目标位于射手的视线范围内。对于火炮等间接火力系统及其他弹道武器，它们无需看见目标，也不必直接瞄准目标射击。其武器沿弹道曲线飞行，这种方式称为间接射击。为了跟踪战况，人们开发了许多模型。例如，基于游戏的计分系统会统计目标被命中的次数和位置，并使用“命中即摧毁”概率（基于真实世界的数据）来模拟命中或未命中目标的情况 [125]。

8.1.5 通信：

这一核心活动涉及通信、指挥与控制的建模。它将前面所有活动联系起来，因为指挥与控制处于环境中，并命令实体开火、移动、观察和通信。文中讨论了军事指挥部中的若干指挥与控制模型；随着越来越多的仿真模型必须依据可用信息作出决策，而过去这通常需要人类决策者参与，这些模型也愈发重要。指挥与控制建模得越好，就越不需要军事专家来提供逼真的训练环境 [125]。基于这些作战建模原理，本文给出了一个野战炮兵示例模型，以说明使用状态空间分析进行可组合性验证的方法。**图60**突出展示了作战建模的各项活动。

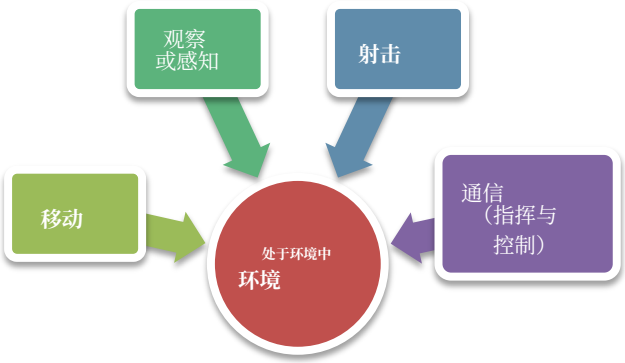


图60：作战建模的活动

8.2 野战炮兵

野战炮兵（FA）是一种间接火力系统[35]，能够在射击系统与目标之间无需视线的情况下打击对手。步兵使用轻型、中型或重型榴弹炮（火炮），为作战部队提供火力支援。同样，海军炮兵也能提供火力，利用导弹打击陆上或海上的直射或间接目标。野战炮兵的总体任务是使用加农炮、火箭和导弹火力摧毁、压制或遏制敌人，并帮助将所有火力支援力量整合到诸兵种联合作战中 [126]。野战炮兵系统根据需要为友军机动部队提供近距离支援、实施反火力和阻断。此类火力可遏制、分割或摧毁敌方攻击队形或防御工事；遮蔽敌人的视线，或以其他方式妨碍其发现并攻击友方目标；还可使用远程火箭或导弹火力摧毁敌方后端纵深目标 [127]。

野战炮兵武器通常部署在隐蔽阵地，以免被敌人侦察发现。野战炮兵射击的这一特点使其成为一个间接射击问题。观察射击（解决间接野战炮兵射击问题的技术）由前沿观察员、司令部（HQ）、火力指挥中心（FDC）以及射击单位（炮兵连）的射击分队协同完成 [126]。**图61**展示了野战炮兵的基本要素，以及间接射击的情形：前沿观察员发现敌方单位，并请求附近友军单位提供火力支援。需要指出的是，这种场景只是为举例而假设并简化的；如今，野战炮兵系统已更加现代化，技术也更加先进。

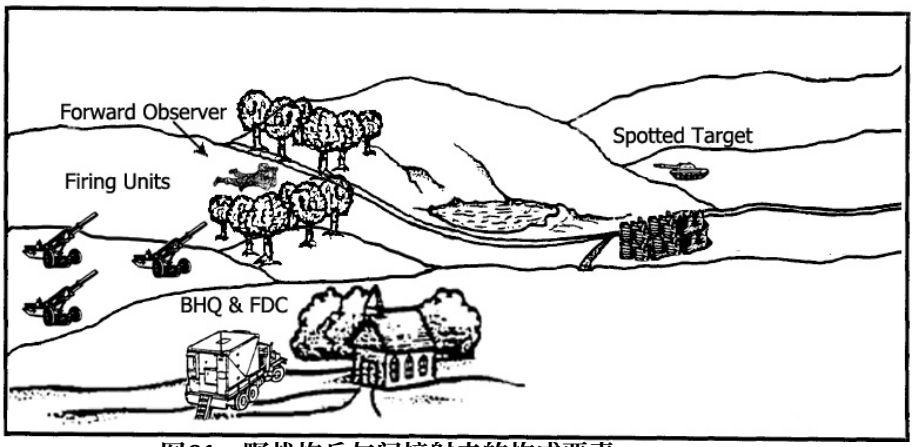


图61：野战炮兵与间接射击的构成要素

8.2.1 仿真对象

基于**图61**，考虑一个间接火力支援场景。在此场景中，敌方单位不在射击单位的视线范围内。观察哨的一名士兵（前沿观察员）观察敌方区域并发现潜在目标。发现目标后，他会呼叫 BHQ 请求火力支援，并提供目标详情。BHQ 请 FDC 从战术和技术角度处理目标。从战术角度看，目标应具有较高的重要性，以取得战术优势。从技术角度看，目标应处于支援炮兵的射程内。如果

[35] 尽管存在一些异常情况，野战炮兵会采用直射火力模式进行交战

目标有效，FDC 就会批准请求，否则将拒绝请求。如果请求获批，BHQ 会将目标分配给射击单位（炮兵连）。我们假设目标分为三种类型：轻型目标（例如营地、部队和卡车）、中型目标（例如坦克、轻型火炮）或重型目标（例如炮兵部队、导弹发射装置）。目标将分别分配给一个、两个或三个炮兵连。这是因为要彻底摧毁中型和重型目标，需要多个炮兵连的火力。根据这一假设，BHQ 将目标分配给炮兵连。炮兵连组件通过计算目标距离和方位角（角度），调整自身方向和仰角，装填适当的弹药并发射炮弹。战场组件受到开火攻击后，如果引爆发生在杀伤半径内，则目标被视为已摧毁，否则视为未命中；观察员会观察到结果，并将这一信息提供给 BHQ。随后针对其他潜在目标重复这一过程，直到所有敌方单位都被压制，这就是最终目标。

8.2.2 野战炮兵模型

基于上述对仿真对象的非形式化描述，构建了一个野战炮兵模型。野战炮兵建模可能有多个目标，包括训练、演习、武器测试或作战优化。针对该仿真对象，发现、选取并组合了以下基于 BOM 的模型。

野战炮兵概念模型

野战炮兵的基于 BOM 的概念模型形式化定义如下：

观察员 = 〈 EnT, EvT, S, AcT 〉 其中 ：

EnT = 观察员 {C_0(标识符), C_1(位置), C_2(当前目标), C_3(结果)}

EvT = {E_0(观察场地, 观察员, 战场, 空值), E_1(发现目标, 战场, 观察员, 目标), E_2(请求火力支援, 观察员, BHQ, currtgt), E_3(请求已批准, 火力指挥中心, BHQ, 观察员, 空值), E_4(请求被拒绝, 火力指挥中心, BHQ, 观察员, 空值), E_5(引爆, 战场, 观察员, detonation), E_6(目标被摧毁, 观察员, BHQ, 空值), E_7(目标未命中, 观察员, BHQ, 空值)}

Act = {A_0(观察场地, 观察员, 战场, E_0), A_1(发现目标, 战场, 观察员, E_1), A_2(请求火力支援, 观察员, BHQ, E_2), A_3(请求已批准, FDC, 观察员, E_3), A_4(请求被拒绝, 火力指挥中心, 观察员, E_4), A_5(引爆, 战场, 观察员, E_5), A_6(目标被摧毁, 观察员, BHQ, E_6), A_7(目标未命中, 观察员, BHQ, E_7)}

S = {S_0(观察员就绪, A_0, S_1), S_1(观察战场, A_1, S_2), S_2(请求火力支援, A_2, S_3), S_3(等待回应, {A_3, S_4}, {A_4, S_0}) , S_4(等待撞击, A_5, S_5) , S_5(评估损伤, {A_6, S_0}, {A_7, S_0}}

表3 4：观察员基础物料清单

战场 = 〈 EnT, EvT, S, AcT 〉 其中:

EnT = 战场 {C_4(标识符), C_5(野战数据), C_6(影响)}

EvT = {E_8(观察场地, 观察员, 战场, 空值), E_9(发现目标, 战场, 观察员, 目标), E_{10}(开火, 炮兵连1, 战场, 开火), E_{11}(开火, 炮兵连2, 战场, 开火), E_{12}(开火, 炮兵连3, 战场, 开火), E_{13}(引爆, 战场, 观察员, 影响), E_{14}(更新字段, BHQ, 战场, 更新) }

Act = {A_8(观察场地, 观察员, 战场, E_8), A_9(发现目标, 战场, 观察员, E_9), A_{10}(开火, 炮兵连1, 战场, E_{10}), A_{11}(开火, 炮兵连 2, 战场 , E_{11}), A_{12}(开火 , 炮兵连 3, 战场 , E_{12}), A_{13}(引爆 , 战场 , 观察员 , E_{13}), A_{14}(更新字段 , BHQ, 战场, E_{14}) }

S = {S_6(场地就绪, {A_8, S_7}, {A_{10}, S_8}, {A_{11}, S_8}, {A_{12}, S_8}), S_7(正在被观察, A_9, S_6), S_8(正在开火, A_{13}, S_9) , S_9(等待更新, A_{14}, S_6)}

表3：野战基础物料清单

BHQ = 〈 EnT, EvT, S, AcT 〉 其中:

EnT =BHQ {C_7(标识符), C_8(位置), C_9(当前目标), C_{10}(目标状态) }

EvT = {E_{15}(请求火力支援, 观察员, 战场, 目标), E_{16}(处理请求, BHQ, FDC, 目标), E_{17}(请求已批准, FDC, BHQ, 观察员, 空值), E_{18}(请求被拒绝, FDC, BHQ, 观察员, 空值), E_{19}(分配目标, BHQ, 炮兵连1, 炮兵连2, 炮兵连3, 分配目标_目标), E_{20}(射击完成, 炮兵连1, BHQ, 空值), E_{21}(射击完成, 炮兵连2, BHQ, 空值), E_{22}(射击完成, 炮兵连3, BHQ, 空值), E_{23}(目标被摧毁, 观察员, BHQ, 空值), E_{24}(目标未命中, 观察员, BHQ, 空值), E_{25}(更新字段, BHQ, 战场, 更新) }

动作 = {A_{15}(请求火力支援, 观察员, 战场, E_{15}), A_{16}(处理请求, BHQ, 火力指挥中心, E_{16}), A_{17}(请求已批准, 火力指挥中心, BHQ, 观察员, E_{17}), A_{18}(请求被拒绝, 火力指挥中心, BHQ, 观察员, E_{18}), A_{19}(分配目标, BHQ, 炮兵连1, 炮兵连2, 炮兵连3, E_{19}), A_{20}(射击完成, 炮兵连1, BHQ, E_{20}), A_{21}(射击完成, 炮兵连 2, BHQ, E_{21}), A_{22}(射击完成 , 炮兵连 3, BHQ, E_{22}), A_{23}(目标被摧毁 , 观察员 , BHQ, E_{23}), A_{24}(目标未命中, 观察员, BHQ, E_{24}), A_{25}(更新字段, BHQ, 战场, E_{25}) }

S={S_{10}(BHQ就绪, A_{15}, S_{11}), S_{11}(呼叫FDC, A_{16}, S_{12}), S_{12}(等待批准, {A_{17}, S_{13}}, {A_{18}, S_{10}}), S_{13}(分配目标, A_{19}, S_{14}), S_{14}(等待开火, {A_{20}, S_{15}}, {A_{21}, S_{15}}, {A_{22}, S_{15}}), S_{15}(等待损伤报告, {A_{23}, S_{16}}, {A_{24}, S_{16}}), S_{16}(正在更新字段, A_{25}, S_{10})}

表36：BHQ 基础物料清单

火力指挥中心 = 〈 EnT, EvT, S, AcT 〉 其中:

EnT =火力指挥中心{C_{11}(标识符), C_{12}(野战数据) , C_{13}(当前目标), C_{14}(结果)}

EvT = {E_{26}(处理请求, BHQ, 火力指挥中心, 目标), E_{27}(请求已批准, 火力指挥中心, BHQ, 观察员, 空值), E_{28}(请求被拒绝, 火力指挥中心, BHQ, 观察员, 空值)}

Act = {A_{26}(处理请求, BHQ, 火力指挥中心, E_{26}), A_{27}(请求已批准, 火力指挥中心, BHQ, 观察员, E_{27}), A_{28}(请求被拒绝, 火力指挥中心, BHQ, 观察员, E_{28})}

S = {S_{17}(FDC就绪, A_{26}, S_{18}), S_{18}(处理中, {A_{27}, S_{17}}, {A_{28}, S_{17}})}

表37：FDC 基础物料清单

炮兵连1,2,3[36] **=** 〈 **EnT, EvT, S, AcT** 〉 其中:

EnT = 炮兵连1,2,3 {C_{15}(标识符), C_{16}(当前目标) }

EvT = {E_{29}(分配目标, BHQ, 炮兵连1, 炮兵连2, 炮兵连3, 分配目标), E_{30}(开火, Battery123, 战场, 开火), E_{31}(射击完成, Battery123, BHQ, 空值)}

行为层 = {A_{29}(分配目标, BHQ, 炮兵连1, 炮兵连2, 炮兵连3, E_{29}), A_{30}(开火, 炮兵连1/2/3, 战场, E_{30}), A_{31}(射击完成, 炮兵连1/2/3, BHQ, E_{31})}

S = {S_{19}(准备开火, A_{29}, S_{20}), S_{20}(准备火炮, A_{30}, S_{21}), S_{21}(射击, A_{31}, S_{19})}

表38：炮兵连（1、2、3）基础物料清单

野战炮兵 = 〈 $\mathbf{AcT}_{IN}$, $\mathbf{AcT}_{OUT}$, **POI** 〉 , 其中:

$\mathbf{AcT}_{IN} = \mathbf{AcT}_{OUT} = \varnothing$

POI = { POI-0(!A0, ?A8), POI-1(!A9, ?A1), POI-2(!A2, ?A15), POI-3(!A16, ?A26), POI-4(!A27, {?A17, ?A3}), POI-5(!A28, {?A18, ?A4}), POI-6(!A19, ?A29), POI-7(!A30, {?A10, ?A11, ?A12}), POI-8(!A31, {?A20, ?A21, ?A22}), POI-9(!A13, ?A5), POI-10(!A6, ?A23), POI-11(!A7, ?A24), POI-12(!A25, ?A14)}

表39：野战炮兵组合物料清单

图62 展示了野战炮兵组合模型的形式化表示。

[36] 此组件有三个实例。

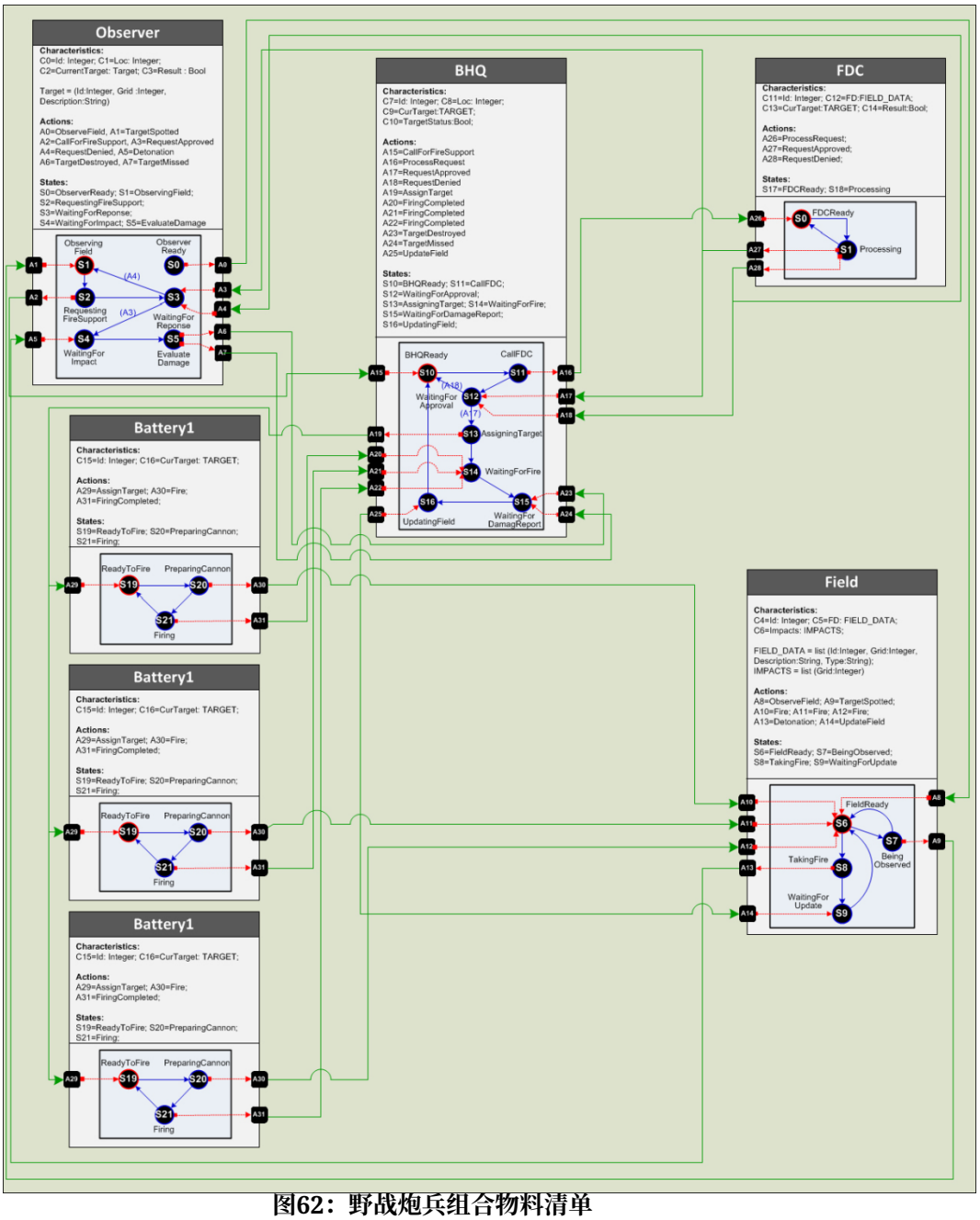


图62：野战炮兵组合物料清单

8.2.3 需求规范

我们将野战炮兵模型的需求规范定义如下：

$RS_0 = \langle O, S \rangle$ where:

目标 O = {o_1} 和系统约束 S = {s_1, s_2 s_3, s_4}o_1：所有敌方单位都必须被摧毁$s_{1,2,3}$：该模型应能在句法和静态语义层面进行组合。状态机应相互匹配，可执行模式应能正确表示概念模型。s_4：不得发生误伤。

8.3 使用CPN状态空间分析验证野战炮兵模型

按照图62发现、选择并组合 BOM 后，概念模型即可进行验证。我们选择CPN状态空间分析技术对其进行验证。

8.3.1 静态与动态分析

我们假设该模型通过了语法分析和静态语义分析。此外，在经历状态机匹配过程时，它能够不断推进，直到达到目标状态。图63展示了组件如何通过事件交换（水平箭头）相互交互，从而使其状态机不断推进（垂直虚线箭头）。基于约束S1、S2和S3a均得到满足这一事实，我们继续进行BOM到E-BOM扩展，这是将概念模型转换为可执行模型的前提步骤。

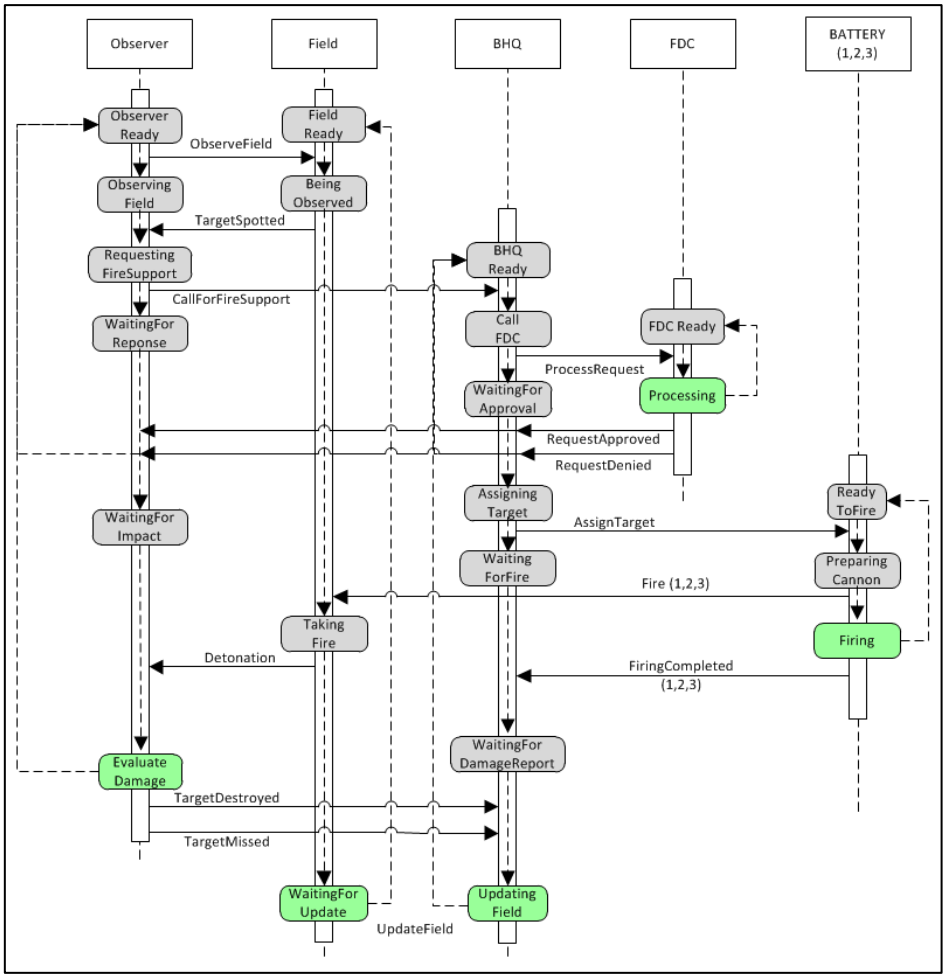


图63：野战炮兵模型的状态机匹配

8.3.2 BOM到E-BOM扩展

在此阶段，借助建模者的输入，所有 BOM组件均扩展为我们提出的E-BOM扩展。下表展示了 FA模型中 BOM的 E-BOM扩展。

观察员工程物料清单						
状态变量和类型	{C_0(标识符:整数), C_1(位置:整数), C_2(当前目标:TARGET), C_3(结果:布尔值)}					
	其中 TARGET = (标识符:整数, 网格[37]:整数, 描述:字符串)					
初始状态	{S_0:ObserverReady}					
变迁						
状态	**事件**	**守卫**	**{状态变量$_{IN}$}**	**{状态变量$_{OUT}$}**	**动作**	**下一状态**
观察员就绪	观察战场					正在观察战场
正在观察战场	发现目标			C2		请求火力支援
请求火力支援	请求火力支援		C2			等待响应
等待响应	请求已批准			C2		等待撞击
等待响应	请求被拒绝					观察员就绪
等待撞击	引爆		C2	C3	动作1	评估伤害
评估伤害	目标已摧毁	[结果=true]	C3			观察员就绪
评估伤害	目标未命中	[结果=false]	C3			观察员就绪

```
动作1 {
    输入 (目标, 引爆);
    输出 (结果);
    动作
    let
     val 网格= #2 目标;
    in
     (已摧毁(网格, 引爆))
    End
}
```

```
函数 已摧毁 (x, []) = false
 | 已摧毁 (x, h::t) = IsDestroyed(x, h) orelse 已摧毁 (x, t);
函数 IsDestroyed(网格, 撞击) =
  let
    val gridst = Int.toString(网格);
    val impactst= Int.toString(撞击);
    val 网格X = valOf(整数.fromString(substring (网格st, 0, 3)));
    val impactX = valOf(Int.fromString(substring (impactst, 0, 3)));
    val 网格Y = valOf(整数.fromString(substring (网格st, 3, 3)));
    val impactY = valOf(整数.fromString(substring (影响t, 3, 3)));
    val X = abs(网格X - 撞击X);
    val Y = abs(网格Y - 撞击Y);
  in
    if (abs(X)<4 且同时 abs(Y)<4)
      则为真
     else false
end;
```

表40：观察员工程物料清单

动作1 是一个 CPN-ML 脚本，用于评估目标是否已摧毁。我们假设火炮发射的炮弹的杀伤半径为 4x4 个网格，即如果炮弹在此半径内命中目标，目标将被摧毁，否则将未命中。请注意，动作1 调用了其他函数，这些函数也在 **表40** 中进行了说明。

[37] 在军事地图中，使用格网坐标系统来确定对象的位置。我们假设此场景中的网格为六位数，前三个整数表示东距，后三个整数表示北距。详情请参见 [**131**]

现场电子物料清单	
SV：类型	{C_4(标识符:整数), C_5(野战数据:战场_数据), C_6(撞击:影响)} 战场_数据 = 列表 (标识符:整数, 网格:整数, 描述:字符串, 类型：字符串）；影响 = 列表（网格：整数）
初始状态	{S_6: 场地就绪}

变迁

状态	事件	守卫	{状态变量$_{IN}$}	{状态变量$_{OUT}$}	动作	下一状态
场地就绪	观察场地					正在被观察
正在被观察	发现目标	[长度 野战数据	C5	C5	动作2	场地就绪
场地就绪	Fire	>0]	C6			正在开火
正在开火	引爆			C6		等待 更新
等待 更新	更新字段		C5	C5	Action3	场地就绪

```
动作2 {
  输入（野战数据）；
  输出（目标）；
  动作
  let
    val indx = 离散（0,(length 野战数据));
    val F = List.nth(野战数据, indx);
    in
    ((#1 F, #2 F, #3 F))
  结束 }
```

```
Action3 {
  输入(更新,野战数据);
  输出 (ufd);
  动作
  let
    val status = #2 更新;
    val 目标 = #1 更新;
    val U = (#1 目标, #2 目标, #3 目标, "敌人");
  in
    if (status=true) then
      rm U 野战数据
    else
      fd
  end }
```

表41：现场电子物料清单

动作2从目标列表（战场数据）中随机选择一个目标，并将其作为参数发送给观察员，模拟观察员在敌方区域发现了一个目标。当从 BHQ 接收到 更新-战场 事件时，执行 Action3。如果目标已被摧毁，则此动作会移除一个对象。

BHQ电子物料清单	
SV：类型	{C7(标识符:整数),C8(位置:整数),C9(当前目标:目标), C10(目标状态:布尔值)}
初始状态	{S_{10}: BHQ就绪 }

变迁

状态	事件	守卫	{SV_{IN}}	{SV_{OUT}}	动作	下一状态
BHQ就绪	请求火力支援		C9			呼叫FDC
呼叫FDC	处理请求			C9		等待 批准
等待 批准	请求已批准		C9			分配 目标
等待 响应	请求被拒绝					BHQ就绪
分配 目标	分配目标			C9	Action4	等待开火
等待开火	射击完成					等待 损伤报告
等待	目标被摧毁		C10			正在更新字段

损伤报告						
等待 损伤报告	目标未命中		C10			正在更新字段
正在更新字段	更新字段			C10		BHQ就绪

```
Action4 {
    输入 (目标);
    输出（赋值_目标）;
    动作
    let
    in
      如果（（#3 目标）= “火炮”）则
        （（#1 目标，#2 目标，#3 目标，[真，真，真]）
      否则如果（（#3 目标）= “坦克”）则
        （（#1 目标，#2 目标，#3 目标，[真，真，假]）
      else
        ((#1 目标, #2 目标, #3 目标, [真, 假, 假])
end}
```

表 42：BHQ电子物料清单

动作 4 用于分配轻型、中型或重型目标。如果目标是重型目标，则为命中该目标分配三个炮兵连。如果目标是中型目标，则分配炮兵连 1 和 2；否则仅分配炮兵连 1。

FDC E-BOM	
SV：类型	{C_{11}(标识符:整数), C_{12}(FD:战场_数据) , C_{13}(当前目标:目标), C_{14}(结果:布尔值)}
初始状态	{S_{17}: FDC就绪 }

变迁

状态	事件	守卫	{SV_{IN}}	{SV_{OUT}}	动作	下一状态
FDC就绪	处理请求		C12，C13，C14		操作5	处理中
处理中	请求已批准	[结果=true]		C13, C14		FDC就绪
处理中	请求被拒绝	[结果=false]		C13, C14		FDC就绪

```
操作5 {
  输入 (目标, fdcd);
  输出 (fdc_结果);
  动作
  let
    val tid = #1 目标;
    val r = List.nth((listsub fdcd (GetFieldByID tid fdcd)),0);
  in
    (if (#4 r = "敌人") then true else false)
  end
}
```

表 43：FDC E-BOM

动作 5 用于处理目标。这里我们仅检查 FDC 内部数据，以确认目标是敌方单位。此方法可扩展为计算目标优先级，并处理其他战术和技术火力指向规则。

炮兵连工程物料清单						
SV：类型	{C_{15}(标识符:整数), C_{16}(当前目标:目标) }					
初始状态	{S_{19}: 准备开火 }					
变迁						
状态	**事件**	**守卫 {SV_{IN}} {SV_{OUT}}**		**动作**		**下一状态**
准备开火	分配目标		C16			准备火炮
准备火炮	Fire			C16	动作6	射击
射击	射击完成					准备开火

```
动作6 {
    输入 (assign_目标);
    输出 (开火);
    动作
    let
        val 炮台编号 = inst();
        val tid = #1 assign_目标;
        val 网格 = #2 assign_目标; val 撞击=网格;
    in
        (炮台编号, tid, 网格, 撞击)
    end
}
```

表 44：炮兵连工程物料清单

动作 6 用于发起开火。它会向战场组件创建一个类型为“开火”的令牌，其中包含射击炮兵连编号、目标编号、目标的网格位置以及撞击位置等信息。

8.3.3 E-BOM 到 CPN 转换

在此步骤中，所有扩展BOM模型（E-BOM）都会自动转换为我们提出的CPN组件，使相应 E-BOM 中的所有变量都添加到结构层（在下图中以红色显示），并将状态机转换为行为层（以绿色显示）。在通信层（以蓝色显示）中，接收事件会转换为输入端口，发送事件会转换为输出端口。图64、图65、图66、图67和图68展示了各组件的 CPN组件模型：

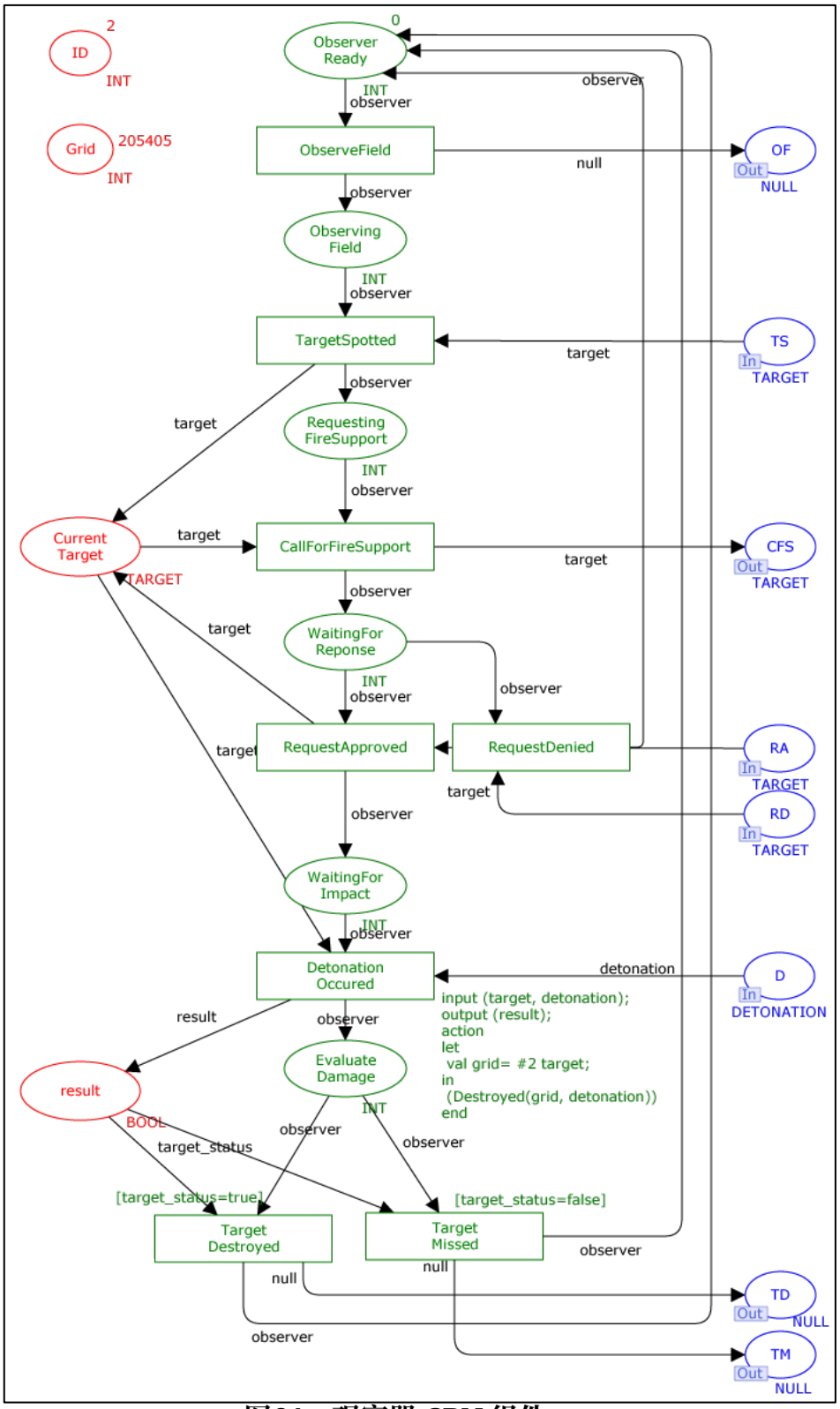


图64：观察器 CPN 组件

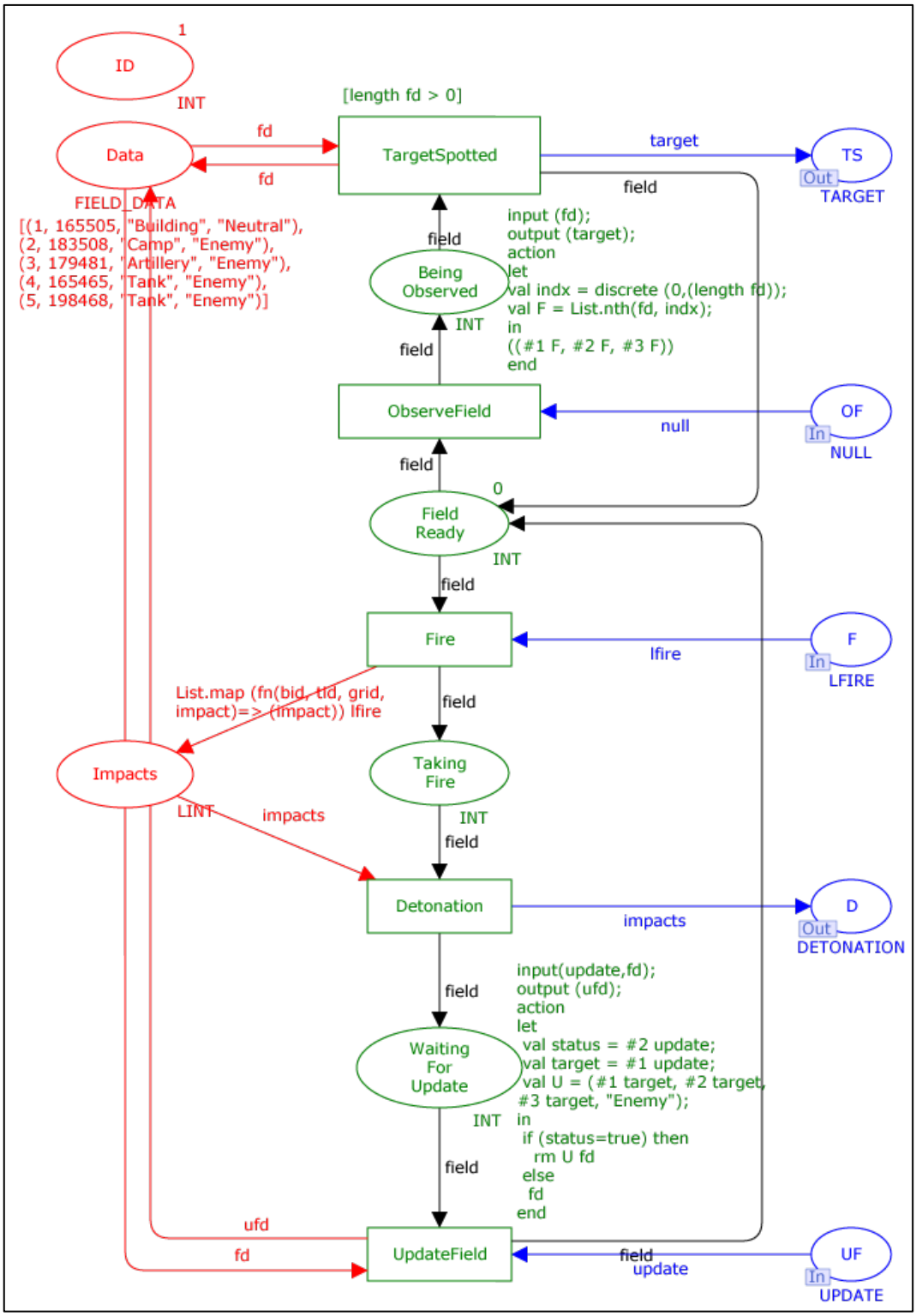


图65：Field CPN组件

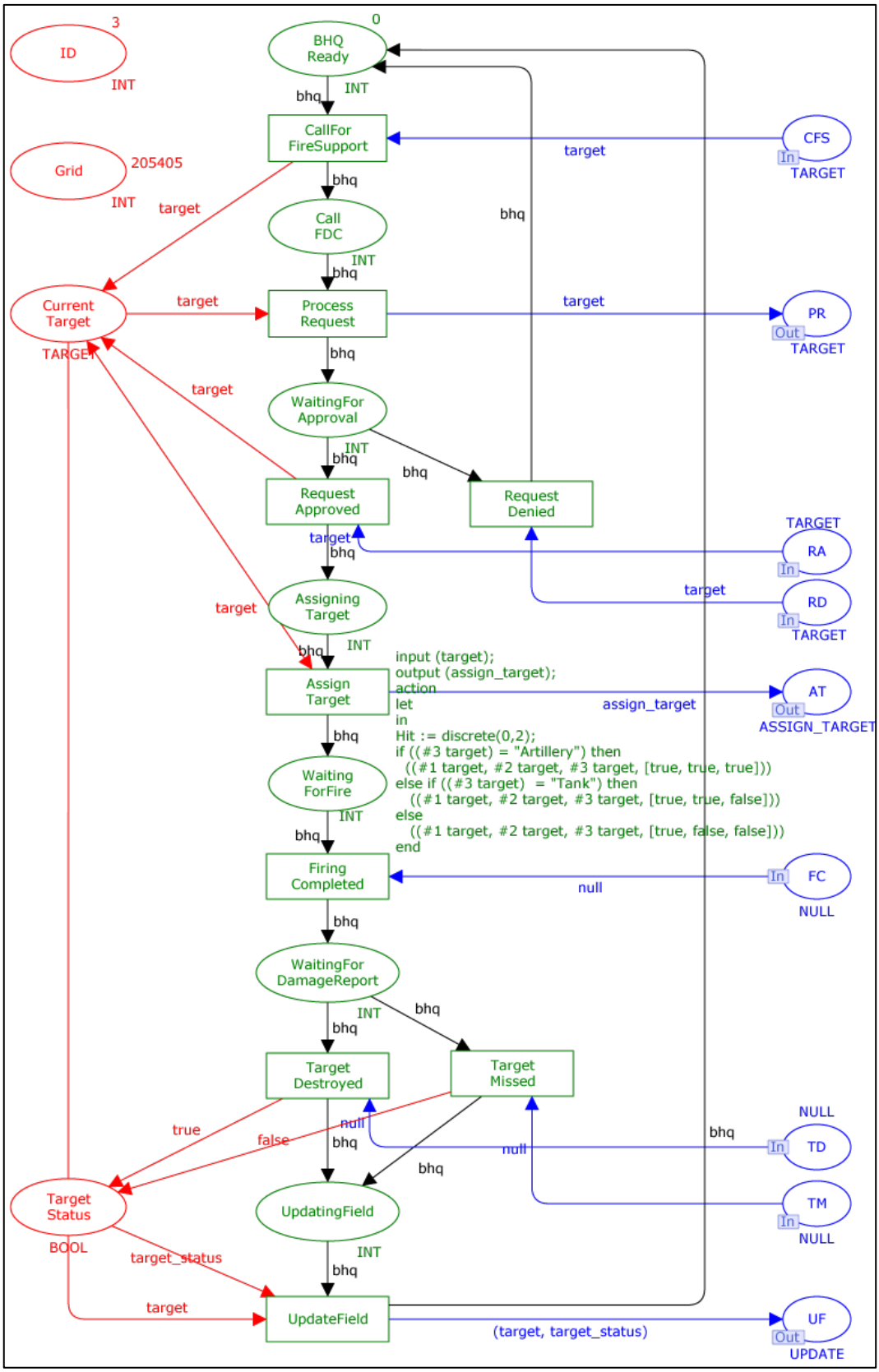


图66：BHQ CPN 组件

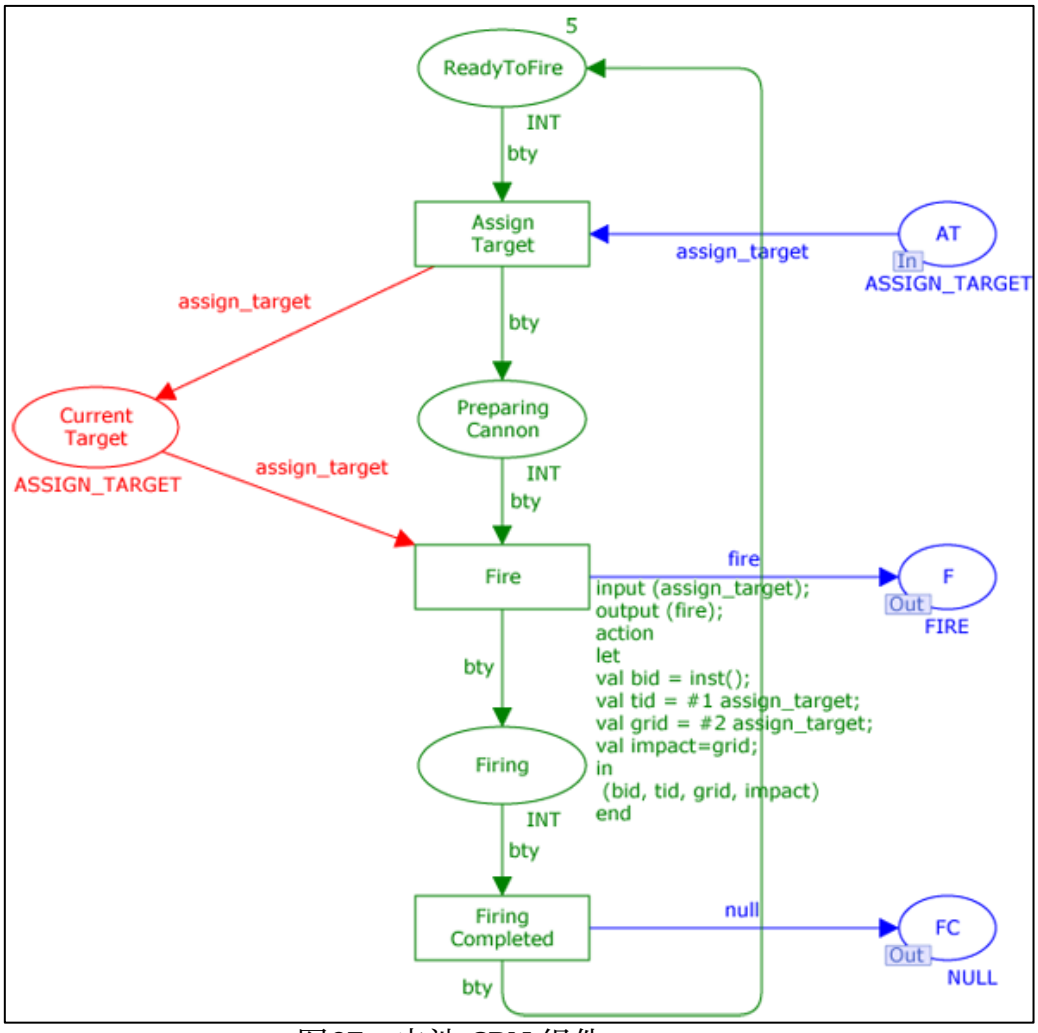

5
ReadyToFire
INT
bty
Assign Target
assign_target
AT
In
ASSIGN_TARGET
assign_target
bty
Current Target
ASSIGN_TARGET
Preparing Cannon
INT
bty
assign_target
Fire
fire
input (assign_target);
output (fire);
action
let
val bid = inst();
val tid = #1 assign_target;
val grid = #2 assign_target;
val impact=grid;
in
(bid, tid, grid, impact)
end
F
Out
FIRE
bty
Firing
INT
bty
Firing Completed
null
FC
Out
NULL
bty

图67：电池 CPN 组件

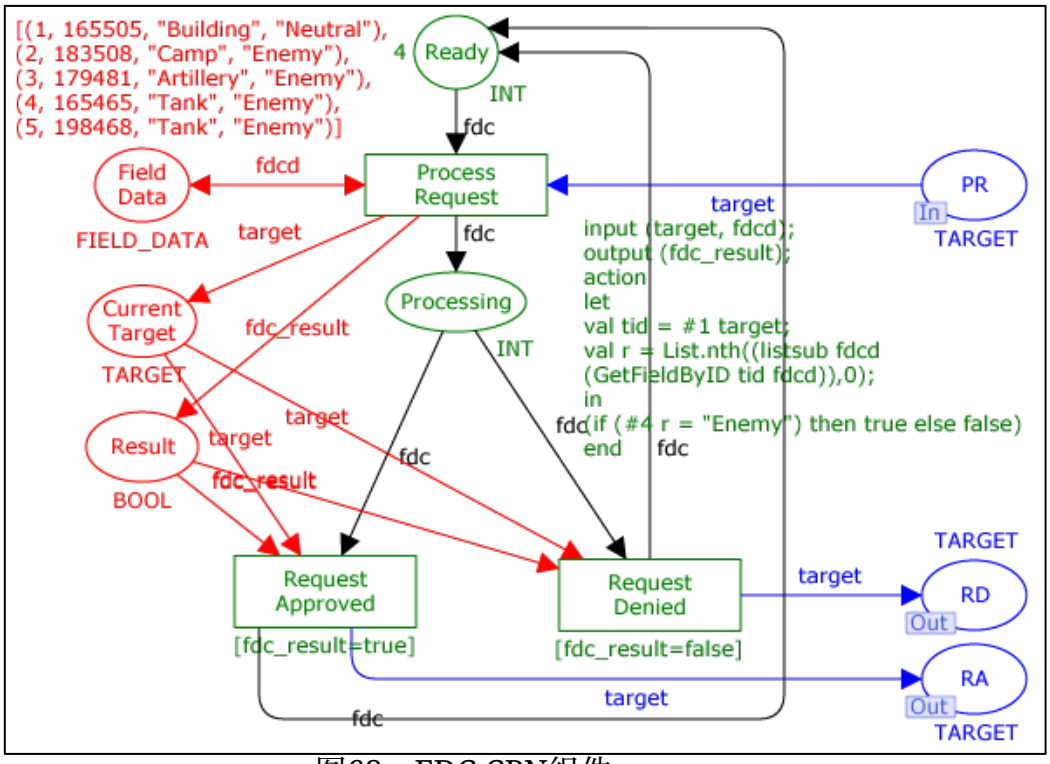


图68：FDC CPN组件

我们假设每个转换后的CPN组件均已通过结构评估（采用检查方法进行）和行为评估（采用功能测试方法进行），因此 S3b 也得到部分满足。

8.3.4 CPN组件的组合

在这一步中，所有 CPN模块通过套接字库所在 CPN组合模型中组合在一起，如图69所示。在该组合模型中，还引入了一些通用辅助组件，例如 Join 和 Fork，以便于组合。

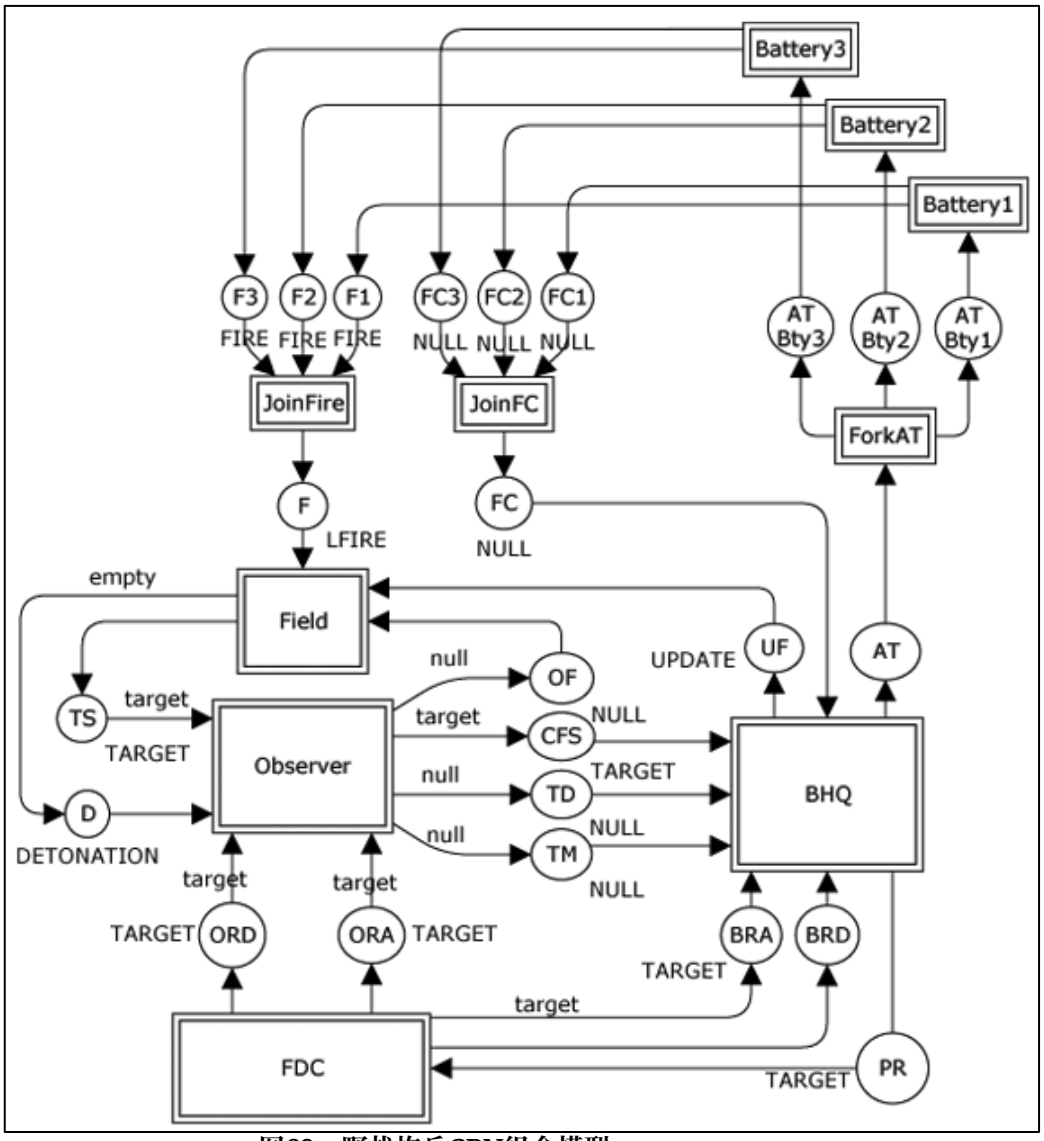


图69：野战炮兵CPN组合模型

模型组合完成后，会在 CPN 执行环境中执行。模型成功执行（根据 **图63**）即可完全满足 S3b。

8.3.5 状态空间分析

下一步，使用 CPN状态空间计算工具生成整个野战炮兵模型的状态空间，并用其进行验证。生成的状态空间图包含 1960 个节点和 6469 条弧，如**图70**所示。

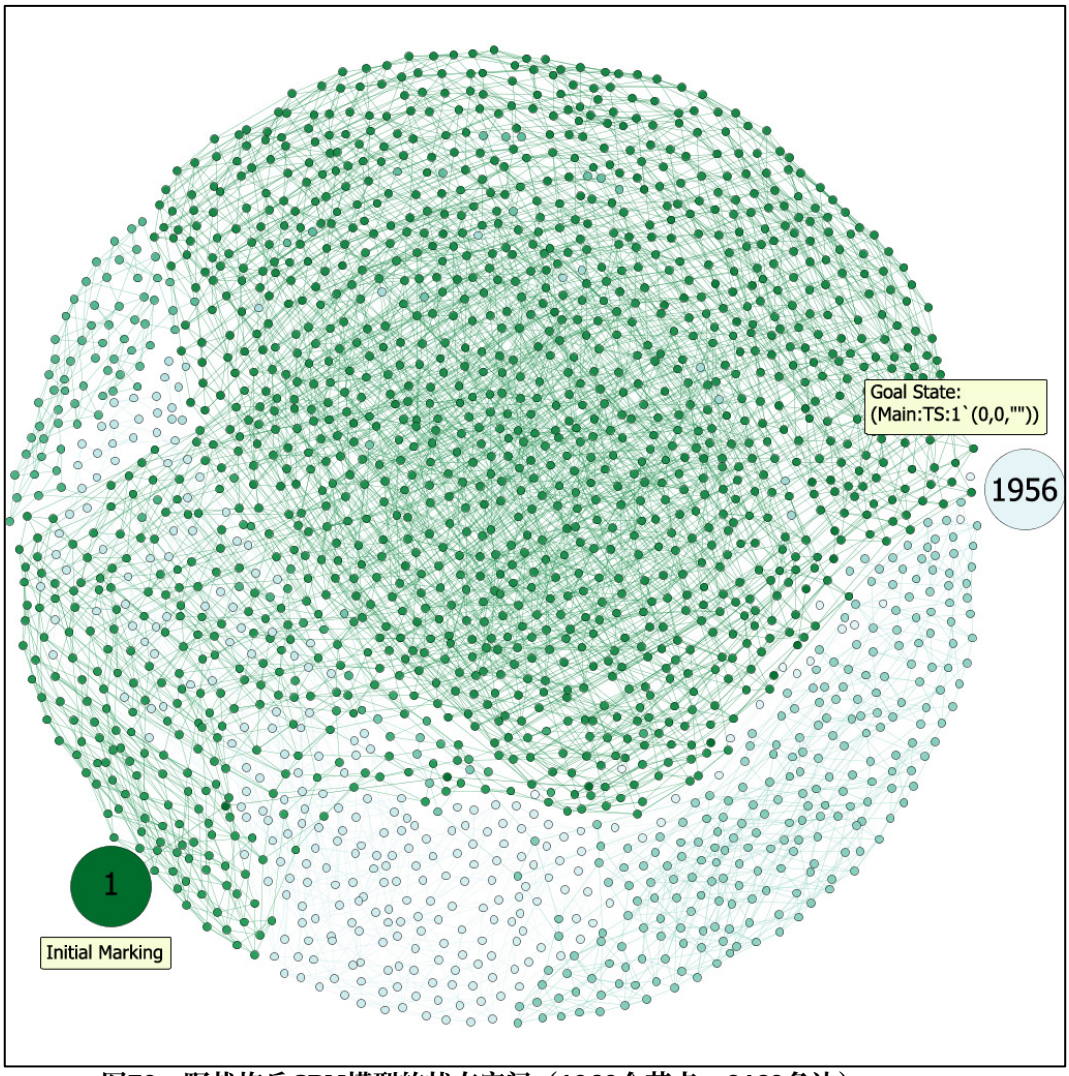


图70：野战炮兵CPN模型的状态空间（1960个节点，6469条边）

在 CPN 工具中构建状态空间后，将其导出为 GraphML 文件格式。需要注意的是，**图70**中的状态空间图使用 Gephi工具进行渲染。其中，“node-1”表示组合模型的初始标识，而“node-1956”表示目标状态（本节稍后将对此进行说明）。绿色深浅（从深到浅）表示与节点 1 的距离。所有节点均通过边相连（其中一些边可能因颜色较浅而不可见）。

将需求规格说明转换为 CPN 属性

为继续进行验证，我们首先将需求规范中定义的目标和约束转换为 CPN 属性。我们假设默认约束 S1、S2 和 S3 已经通过验证。

目标	目$_1$：所有敌方单位都必须被摧毁
CPN 转换	当观察员发现敌方单位时，因此我们认为，如果再也没有敌方单位能够被发现（因为场数据为空），那么所有敌人都应被摧毁。因此，应存在一个 TS（目标-已发现）库所含有空值令牌的标识。如果找到这样的标识，那么就认为已达成目标。可以使用以下 CPN 函数来验证这一性质。
CPN 函数	函数 AllTargetDestroyed() = let val token = 1`(0,0,""); /*创建搜索条件 */ val 谓词 = fn n => (Mark.Main'TS 1 n) = token; /*创建谓词函数*/ val TS = PredAllNodes (谓词); /* 内置-节点搜索函数 in if (length TS > 0) then true else false 结束;
结果	执行函数 AllTargetDestroyed() 时，它会返回 True。这一点从 **图70** 也可以明显看出，其中标识 1956 表示目标状态，并且可从初始状态 1 到达。

约束	S$_4$: 不应发生误伤。
CPN 转换	当“更新字段” (UF) 库所从 BHQ组件获取一个令牌时（该令牌将作为战场组件的输入），它会显示哪个野战单位已摧毁。我们可以在状态空间中收集所有此类节点（UF 战场有令牌时），并检查所有已摧毁的野战单位是否都属于“敌人”类型。如果此条件在整个状态空间中都成立，则 S4 成立。可以使用以下 CPN-ML 函数来检查是否发生过误伤。要满足 S4，结果应为 false 以满足 S4

CPN 函数	函数 CheckFriendlyFire() =
	let
	值 谓词 = 函数 n => IsNotEnemy(Mark.Main'UF 1 n) = 真;
	值 友方单位列表 = PredAllNodes（谓词）;
	in
	如果（友方单位列表的长度 > 0）/* 表示存在误伤 */
	那么为真
	else false
	结束;
	
	函数 IsNotEnemy（更新）=
	let
	val upd:更新 = List.nth(update, 0);
	val target = #1 upd; /*从库所：UF 的令牌中提取信息 */
	in
	if GetType(#1 target) <> "敌人" then true /*检查场单位类型*/
	else
	false
	end;

Result	函数 CheckFriendlyFire() 返回 false，因为没有发生这样的通信弧在提供给模型的数据（初始状态）中。 为检查该函数是否正常工作，我们构造了一个反例假设 FDC组件存在错误（即它错误地接受友方单位的火力支援请求），然后运行了该程序，并发现了发生误伤的轨迹。

由于所有约束和目标均已满足，我们认为该野战炮兵模型在所有层级上均可组合，并且已根据其给定的规格完成验证。因此，基于 BOM 的组合模型符合进一步实现的条件。

8.4 状态空间约简

本节介绍所提出的状态空间约简技术的应用。为了对使用 CPN 工具生成的野战炮兵模型进行状态空间约简，我们执行以下步骤。

1. 节点描述裁剪

CPN 状态空间图中的每个节点都有描述。这些描述主要说明模型所有库所中令牌（或多个令牌）的存在情况，这称为标识。如果模型包含许多库所或子模型，这些描述就会很长。在此步骤中，我们删除所有描述，只保留与主模型库所相关的信息。为执行此步骤，我们使用函数库函数 **`NodeDescriptorOptions()`**（参见手册 [73]）。裁剪描述后，我们只能获得与主库所相关的节点信息；否则，该信息将是一个“空值”字符串。（这是后续步骤中的一个重要区别）。

从概念上讲，我们假设，精简节点描述不会导致信息丢失，因为主库所中的信息以外的所有信息，都是由组合组件的内部逻辑生成的。由于组合组件被视为黑盒，它们最终会在任意一个主库所中输出重要信息（以令牌的形式）。这些信息足以回答与所考虑模型相关的任何验证查询。

2. 导出为 GraphML

下一步，我们将状态空间图导出到外部文件。由于无法在 CPN环境中直接操作 CPN 状态空间图，因此我们将该图导出为标准 GraphML格式 [128] ，其中包含精简后的节点描述和边的信息。为完成这一步，我们在 CPN-ML 中开发了一个 GraphML 写入函数。

3. 缩减算法

接下来，我们应用**表16**中指定的缩减算法。该算法通过一个使用 JUNG库进行图操作的 Java 应用实现。简而言之，所有描述为“空值”的节点都会被移除（因为它们无关紧要）。移除一个节点时，该节点所有的入边和出边也会一并移除。因此，为保留图的结构，我们将该节点的每个前驱节点与每个后继节点连接起来。检查完所有节点后，约简即告完成。野战炮兵模式缩减后的图如**图71**所示。

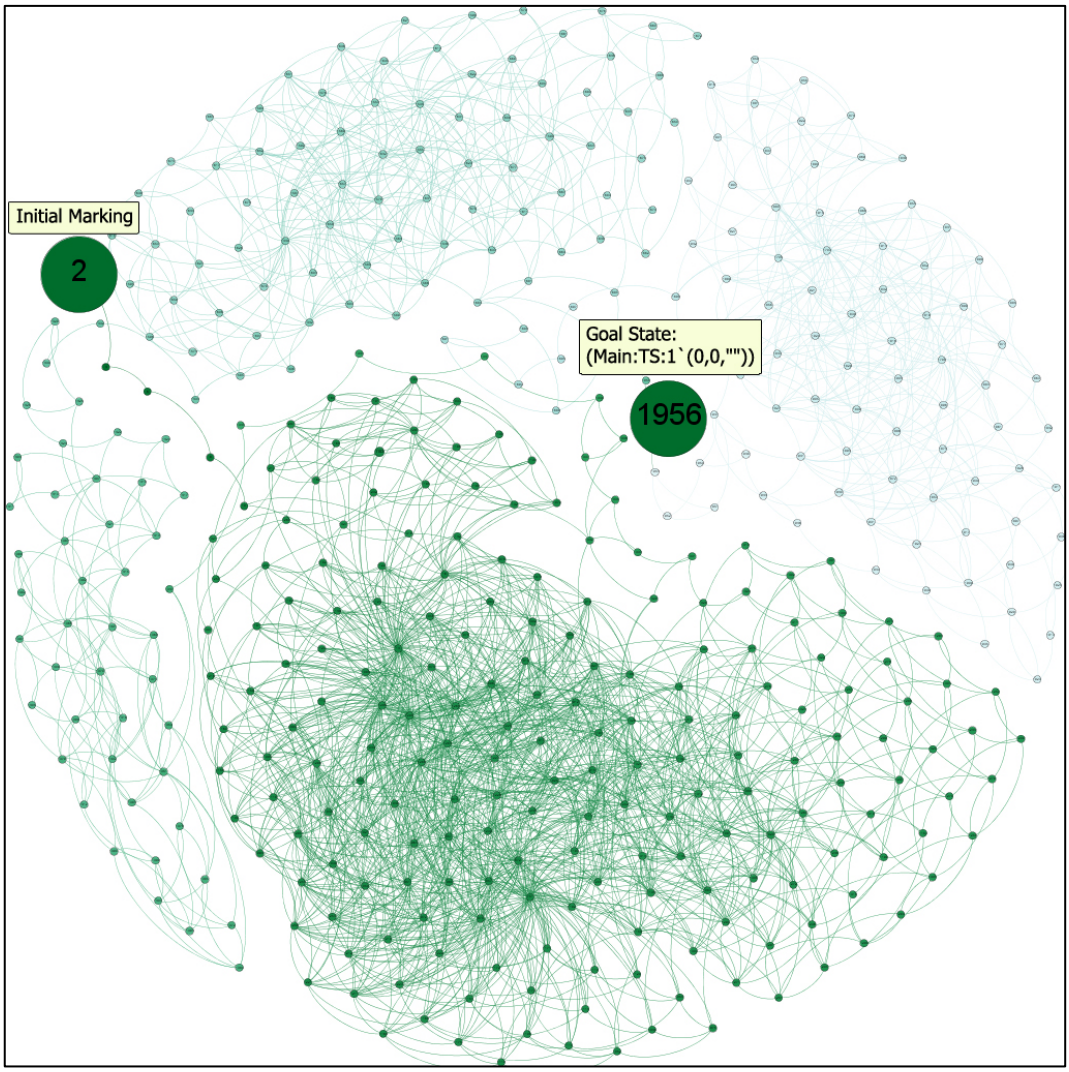


图71：野战炮兵模型的约简状态空间图

在**图71**中，节点-2表示初始标识（请注意，节点-1在约简过程中已被裁剪）。节点-1956仍表示目标状态。（请注意，约简过程中节点 ID 保持不变。）

	约简图	原始图	百分比
节点	4	1960	21%
边	2503	6469	38 %

表45：约简统计

表45显示，节点数量减少至原图的 21%，边数量减少至原图的 38%。

8.5 总结

本章使用基于CPN的状态空间分析技术，讨论基于 BOM 的组合模型的验证。文中介绍了一个野战炮兵示例模型，并将整个验证过程应用于该模型。文中还展示了如何将需求规范转换为 CPN 属性，以及如何使用状态空间分析和查询函数对其进行验证。

状态空间分析的优势在于其具有穷尽性，并能引导建模者了解组合模型在抽象层执行期间可能出现的所有情况。状态空间图有助于研究这些情况，并详细理解各组件的动态行为。此外，结合该方法提出的状态空间查询函数有助于回答不同的验证问题，并根据需求评估模型的正确性。然而，该方法容易受到状态空间爆炸的影响：对于一个简单的野战炮兵模型，生成了 2503 个节点和 6469 条边。为应对这一问题，我们提出了一种有效的状态空间缩减技术，它不仅能将大型状态空间缩减至合理规模，还能保留正确验证所需的重要信息。我们通过概念验证展示了如何将所提出的状态空间缩减技术应用于野战炮兵模型。

第9章 使用基于 CSP 的模型检验技术进行模型验证

Chapter 9
Model Verification using CSP based Model Checking Technique

由于模型检验相比传统形式化方法具有诸多优势，它正逐渐成为软件验证的标准方法。通信顺序进程（*CSP*）是一种基于事件的形式语言，用于描述并发系统中的交互模式；凭借进程代数（也称进程演算）的理论基础，它非常适用于并发行为规范与验证。将基于 *CSP* 的模型检验技术应用于可组合性验证也被证明非常有效，尤其是在关注动态语义可组合性级别时。本章重新采用第*8*章中介绍的野战炮兵模型，并对其进行扩展，加入用于刻画实时概率系统行为的信息。文中展示了如何将概率定时野战炮兵模型转换为组合 *CSP* 模型，并使用 *PAT* 对其进行验证。

本章将讨论第8章中介绍的修改版野战炮兵模型，并以此为例。该示例旨在描述一个具有时间约束和概率行为的系统模型。据我们所知，PN代数方法完全不支持对定时模型或概率系统进行验证。此外，基于CPN的方法对定时系统的验证支持有限，但不涵盖概率系统。因此，我们提出对实时概率系统应用模型检验，并展示如何使用名为 PAT 的基于 CSP 的模型检验器（见 3.2.7）验证此类系统的组合模型。

9.1 野战炮兵场景

野战炮兵模型的场景略有不同。假设一名士兵观察战场并发现敌方单位。发现目标后，他会呼叫 BHQ 请求火力支援，并提供目标详情。在军事实践中，目标时间（TOT）是多个射击单位遵循的一种野战炮兵协同协议。这项技术由美国陆军在第二次世界大战期间开发。该技术会精确预先确定预计准备时间，以及弹药从各炮兵连飞抵目标区域所需的飞行时间。指定目标时间（TOT）后，每个将参与攻击该目标的炮兵连都会从 TOT 中减去飞行时间，以确定开火时间。各射击单位发射炮弹，使所有弹药恰好同时抵达目标。这样做是为了实现最大的目标摧毁。如果多次撞击之间存在间隔，敌方士兵就有时间卧倒或躲进藏身处，而机动载具也可能逃脱 [129]。

BHQ 将目标分配给各炮兵连，并为其安排一个需要遵守的特定“目标时间”（TOT）。每个炮兵连都需要一定时间来准备装载适当的弹药，并根据利用距离和方位角（角度）计算出的射击解，设置炮管的正确对准方向和朝向

，以瞄准分配的目标。假设每个炮兵连都需要一段随机的准备延迟。各炮兵连就绪后，会按照各自的时间开火，以确保所有炮弹都在指定的目标时间命中目标。我们还假设，每个炮兵连命中目标确切位置的概率为“0.9”。与第8章的前一个场景不同，我们假设战场组件中只有一个目标，且三个炮兵连都参与射击行动。为此场景构建概念模型时，由以下 BOM组件组成：

战场：目标位置（我们假设只有一个目标）。**观察员**：向 BHQ 请求火力支援的士兵。**BHQ**：

负责监督整个火力支援行动、响应火力支援请求，并为炮兵连分配目标。

炮兵连：三个炮兵连单位（火炮和炮班）负责在指定时间恰好命中目标。

（请注意，组合中已移除 FDC组件。为简化起见，还减少了部分实体特征和事件参数）。野战炮兵概念模型修改后的 BOM组件正式定义如下：

观察员 = 〈 **EnT, EvT, S, AcT** 〉 其中：

EnT = 观察员 { C_0(目标)}

EvT = {E_0(请求火力支援, 观察员, BHQ, 目标), E_1(引爆, 战场, 观察员, 引爆)}

Act = {A_0(请求火力支援, 观察员, BHQ, E_0), A_1(引爆, 战场, 观察员, E_1)}

S = {S_0(观察员就绪, A_0, S_1), S_1(等待撞击, A_1, S_0) }

表 46：观察员基础物料清单

战场 = 〈 **EnT, EvT, S, AcT** 〉 其中：

EnT = 战场 {C_1(摧毁[3])}**EvT** = {E_2(开火, 炮兵连1, 战场, 炮台编号), E_3(开火, 炮兵连2, 战场, 炮台编号), E_4(开火, 炮兵连3, 战场, 炮台编号), E_5(引爆, 战场, 观察员, 摧毁)}**Act** = {A_2(开火, 炮兵连1, 战场, E_2), A_3(开火, 炮兵连2, 战场, E_3), A_4(开火, 炮兵连3, 战场, E_4), A_5(引爆, 战场, 观察员, E_5) }**S** = {S_2(场地就绪, {A_2, S_3}+{A_3, S_3}+{A_4, S_3}) , S_3(正在开火, A_5, S_2)}

表 47：野战基础物料清单

BHQ = 〈 EnT、EvT、S、AcT 〉，其中：

EnT =BHQ {C_2(目标时间) }

EvT = {E_6(请求火力支援, 观察员, 战场, 目标), E_7(分配目标, BHQ, 炮兵连1, 炮兵连2, 炮兵连3, 目标时间), E_8(射击完成, 炮兵连1, BHQ, 空值), E_9(射击完成, 炮兵连2, BHQ, 空值), E_{10}(射击完成, 炮兵连3, BHQ, 空值}

通信弧 = {A_6(请求火力支援, 观察员, 战场, E_6), A_7(分配目标, BHQ, 炮兵连1, 炮兵连2, 炮兵连3, E_7), A_8(射击完成, 炮兵连1, BHQ, E_8), A_9(射击完成, 炮兵连2, BHQ, E_9), A_{10}(射击完成, 炮兵连3, BHQ, E_{10})}

状态={S_4(BHQ就绪, A_6, S_5), S_5(分配目标, A_7, S_6), S_6(等待开火, {A_8, S_4} + {A_9, S_4} + {A_{10}, S_4})}

表 48：BHQ 基础物料清单

炮兵连1,2,3 = 〈 EnT, EvT, S, AcT 〉，其中：

EnT = 炮兵连1,2,3 { C_3(炮台编号), C_4(已摧毁) }

EvT = {E_{11}(分配目标, BHQ, 炮兵连1, 炮兵连2, 炮兵连3, 目标时间), E_{12}(准备开火, 炮兵连1/2/3, 炮兵连1/2/3, 空值), E_{13}(开火, Battery123, 战场, 炮台编号, 已摧毁), E_{14}(射击完成, Battery123, BHQ, 空值)}

动作 = {A_{11}(分配目标, BHQ, 炮兵连1, 炮兵连2, 炮兵连3, E_{11}), A_{12}(准备开火, 炮兵连1/2/3, 炮兵连1/2/3, E_{12}), A_{13}(开火, 炮兵连1/2/3, 战场, E_{13}), A_{14}(射击完成, 炮兵连1/2/3, BHQ, E_{14})}

状态 = {S_7(炮台空闲, A_{11}, S_7), S_8(准备中, A_{12}, S_9), S_9(准备开火, A_{13}, S_{10}), S_{10}(射击, A_{14}, S_7)}

表 49：炮兵连（1、2、3）基础物料清单

野战炮兵 = 〈 $_{IN}$, AcT_{OUT}, POI 〉 其中：

$AcT_{IN} = AcT_{OUT} = \varnothing$

POI = { POI-0(!A0, ?A6), POI-1(!A7, ?标识符11), POI-2(!A12), POI-3(!A13, {?A2, ?A3, ?A4}), POI-4(!A14, {?A8, ?A9, ?A10}), POI-5(!A5,?A1}

表50：野战炮兵组合物料清单

组合野战炮兵模型使用我们提出的图形表示法显示于**图72**中。

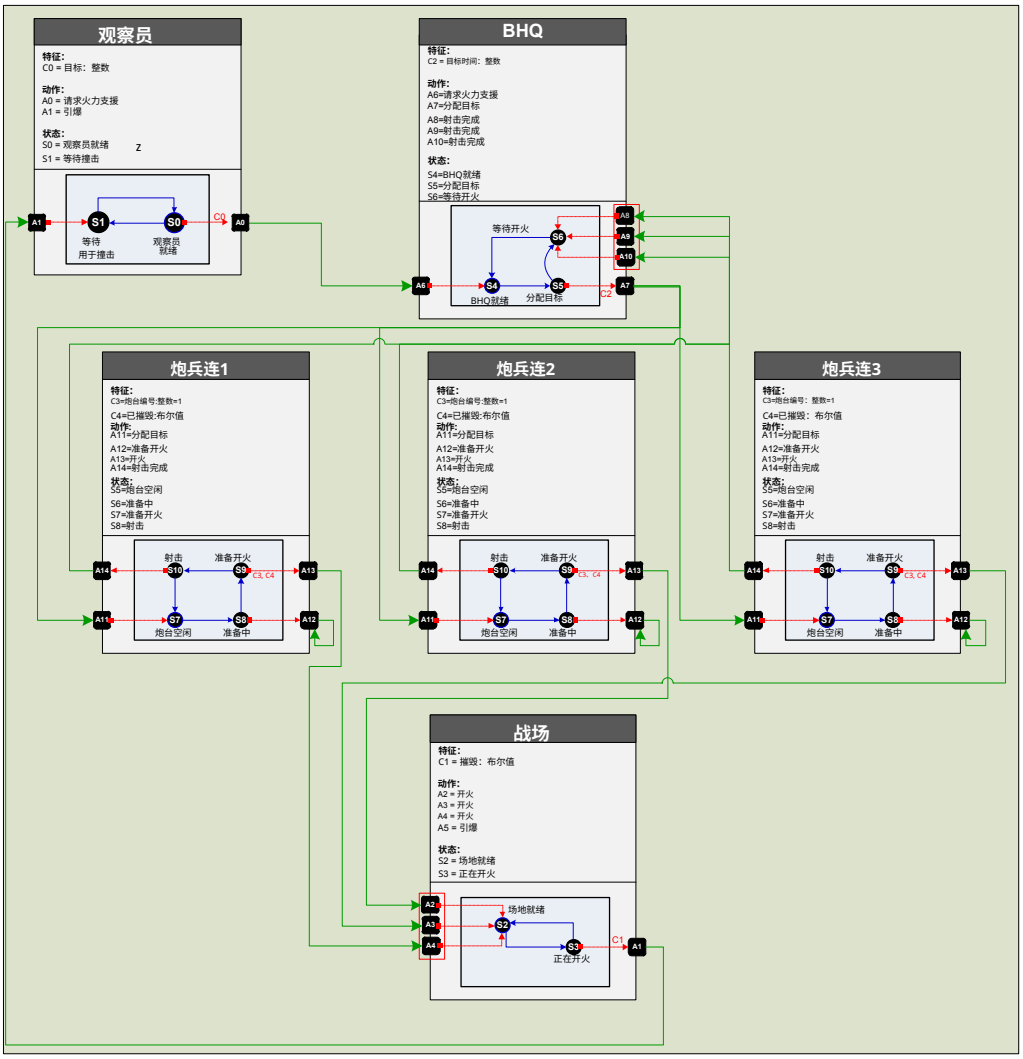


图72：野战炮兵组合模型

9.2 需求规范

我们将修改后的野战炮兵模型的需求规格定义为：

$RS_0 = \langle O, S \rangle$ where:

目标 $O = \{o_1, o_2\}$ 和系统约束 $S = \{s_1, s_2, s_3, s_4\}$ o_1： 所有射击单位都应恰好在

o_2： 所有射击单位都应在给定时间恰好向目标开火（即应满足目标时间属性）

$s_{1,2,以及3}$： 模型应在语法和静态语义层面上可组合。各状态机应相匹配，且可执行模式应正确表示概念模型。

9.3 使用模型检验进行验证

发现 BOM 后，根据仿真对象选出相应的 BOM，并将其组合起来构成概念模型。现在，该组合模型已就绪，可以进行验证。在此阶段，我们选择模型检验技术对其进行验证。

9.3.1 静态与动态分析

我们假设该模型通过了语法和静态语义分析。此外，当模型进行状态机匹配过程时，能够不断推进，直到达到目标状态。**图73**展示了状态机匹配过程中每个组件的状态机之间的交互。

基于约束S1、S2和S3a均得到满足这一事实，我们继续进行BOM到E-BOM扩展。

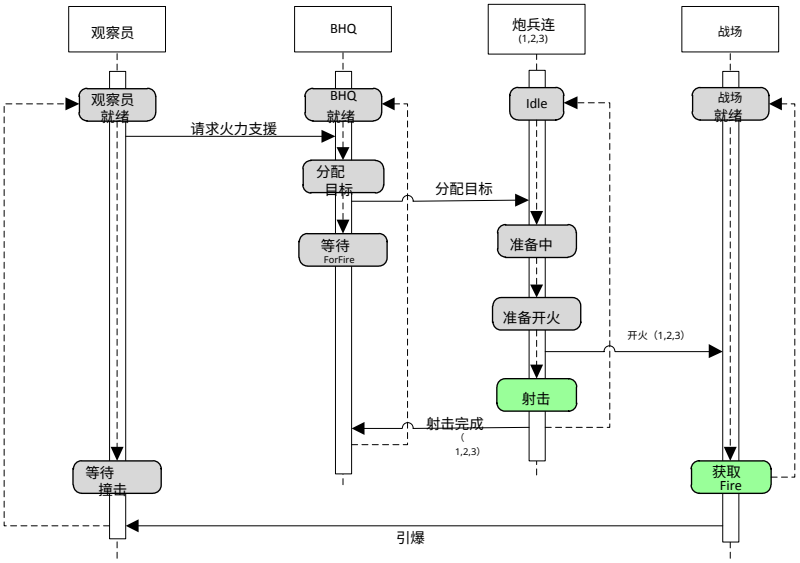


图73：野战炮兵模型的状态机匹配

9.3.2 BOM到E-BOM扩展

在此阶段，借助建模者的输入，所有BOM组件都扩展为我们提出的E-BOM扩展方案。这里提议纳入时间约束和概率因素等附加信息。下表展示了野战炮兵模型中各BOM的E-BOM扩展。

观察员工程物料清单					
状态变量和类型	{C_0(目标:整数}				
初始状态	{S_0:观察员就绪}				
变迁					
状态	**事件**	**Time**	**守卫**	**动作**	**下一状态**
观察员就绪	请求火力支援				等待撞击
等待撞击	引爆				观察员就绪

表51：观察员工程物料清单

现场电子物料清单	
状态变量和类型	{C_1(射击_结果_的_炮兵连1:布尔值}, {C_1(射击_结果_的_炮兵连2:布尔值}, {C_1(射击_结果_的_炮兵连3:布尔值}
初始状态	{S_2:场地就绪}

变迁

状态	事件	Time	守卫	动作	下一状态
场地就绪	开火(1)			动作1	正在开火
	射击(2)			动作2	
	射击(3)			Action3	
正在开火	引爆				场地就绪

```
/* 用于定义概率动作的 CSP 脚本，目标有 95% 的概率会
在接收到来自炮兵连1的开火事件时被摧毁，另有 5% 的概率
目标会未命中 */
动作1{
  概率选择
  {       [0.05] : 开火?1 → 原子型{tau{ 摧毁 [0]=False;} → Skip}
          default : 开火?1→ 原子型{tau{ 摧毁 [0]=True;}→ Skip}
        }
  }
/* 来自电池2 */
动作2{
  概率选择
  {       [0.05] : 开火? 2 → 原子型{tau{ 摧毁 [1]=False;} → Skip}
          默认：开火? 2→ 原子型{tau{ 摧毁 [1]=True;}→ Skip}
        }
}
/* 来自炮兵连 3 */
Action3{
  概率选择
  {       [0.05] : 开火? 3 → 原子型{tau{ 摧毁 [2]=False;} → Skip}
          默认：开火?3→ 原子型{tau{ 摧毁 [2]=True;}→ Skip}
        }
}
```

表52：现场电子物料清单

BHQ电子物料清单	
状态变量和类型	{C_2(目标时间:整数}
初始状态	{S_4: BHQ就绪 }

变迁

状态	事件	Time	守卫	动作	下一状态
BHQ就绪	请求火力支援				分配目标
分配目标	分配目标				等待开火
等待开火	射击完成(1)				BHQ就绪
	射击完成(2)				
	射击完成(3)				

表53：BHQ电子物料清单

炮兵连(1,2,3) E-BOM	
状态变量和类型	{C_3(炮台编号:整数)}
初始状态	{S_7: 炮台空闲 }

变迁

状态	事件	Time	守卫	动作	下一状态
炮台空闲	分配目标				准备中
准备中	准备开火	等待[ProB]		动作1	准备开火
准备开火	Fire				射击
射击	射击完成				炮台空闲

/* 用于定义概率等待动作的 CSP 脚本，每个炮兵连有 94% 的概率会在给定的目标时间恰好开火，另有 6% 的概率会
提前或延迟开火 */

```
动作1{
        概率选择{
            [0.01] : 等待[目标时间-3]; 准备开火 → 准备开火(i)
            [0.01] : 等待[目标时间-2]; 准备开火 → 准备开火(i)
            [0.01] : 等待[目标时间-1]; 准备开火 → 准备开火(i)
            [0.94] : 等待[目标时间]; 准备开火 → 准备开火(i)
            [0.01] : 等待[目标时间+1]; 准备开火 → 准备开火(i)
            [0.01] : 等待[目标时间+2]; 准备开火 → ReadyToFire(i)
            [0.01] : 等待[目标时间+3]; 准备开火 → ReadyToFire(i)
            };
    }
```

表 54：BHQ电子物料清单

9.3.3 E-BOM 到 CSP# 转换

在此步骤中，所有概率定时扩展 BOM 模型都会使用我们的自动 BOM 到 CSP 转换工具自动转换为 CSP#。**图 74**展示了全局代码块，其中定义了全局变量以及每个 BOM 发送-接收事件对的通信通道。

```
//------------全局块 ----------------------------------
#定义 TOT 30; //预定义目标时间常量
枚举 {命中, 未命中}; //命中或未命中标志
//每个炮兵连的命中/未命中比例为 = 95:5 %
变量 射击_结果_的_炮兵连1=未命中;
变量 射击_结果_的_炮兵连2=未命中;
变量 射击_结果_的_炮兵连3=未命中;

//为每个事件定义一个通道
通道 请求火力支援 0;
通道 引爆 0;
通道 分配目标 0;
通道 射击完成 0;
通道 开火 0;
```

图 74：野战炮兵模型的全局代码块

图 75展示了 Observer BOM 组件的 CSP 代码。发送事件会转换为带有发送运算符‘!’的通道，接收事件则会转换为带有接收运算符‘?’的通道。

```
//============================================================
//观察者组件
//============================================================

ObserverSM = 观察员就绪(); //初始状态

观察员就绪()=
 (请求火力支援!0 ->等待撞击());

等待撞击()=
 (引爆?0 -> 观察员就绪());

//============================================================
```

图75：观察者组件的CSP表示

```
//============================================================
//BHQ
//============================================================
BHQSM = BHQ就绪();//初始状态

BHQ就绪()=
(请求火力支援?0 ->分配目标());

分配目标()=
(分配目标!> 分配目标!2 -> 分配目标!3 ->
        等待开火());
//发送 分配目标 至多个接收方

等待开火()=
        射击完成?>射击完成?2->射击完成?3->
                BHQ就绪();
//从多个发送方接收射击完成

//============================================================
```

图76：BHQ组件的CSP表示

```
//=============================================================
//炮兵连 i={1,2,3}
//=============================================================
BatterySM(i) = 炮台空闲(i); //初始状态
炮台空闲(i)=
(分配目标?i->准备中(i));
准备中(i)= 概率选择{
              [0.01] :  等待[目标时间-3]; 准备开火-> 准备开火(i)
              [0.01] :  等待[目标时间-2]; 准备开火-> 准备开火(i)
              [0.01] :  等待[目标时间-1]; 准备开火-> 准备开火(i)
              [0.94] :  等待[目标时间]; 准备开火-> 准备开火(i)
              [0.01] :  等待[目标时间+1]; 准备开火-> 准备开火(i)
              [0.01] :  等待[目标时间+2]; 准备开火-> 准备开火(i)
              [0.01] :  等待[目标时间+3]; 准备开火-> 准备开火(i)
              };
// 目标时间是全局常量
//准备开火是内部事件
准备开火（i）= 开火! i->射击（i）;
射击（i）= 射击完成! i ->炮台空闲（i）;
//=============================================================
```

图77：电池组件的 CSP表示

```
//=============================================================
//战场组件
//=============================================================
FieldSM = 场地就绪(); //初始状态

场地就绪()=
        概率选择{
        [0.05]: 开火?1 ->
                原子型{tau{射击_结果_Of_炮兵连1=未命中;} -> Skip}
         默认： 开火? 1 ->
                原子型{tau{射击_结果_的_炮兵连1=命中;} -> Skip}
 }      |||
/* ||| 是同步事件之间的交错算子
射击(1)、射击(2)和射击(3) */
        概率选择
        {       [0。05]: 开火? 2 ->
                        原子型{τ{射击_结果_的_炮兵连2=未命中;} -> Skip}
                默认： 开火?2 ->
                        原子型{τ{射击_结果_的_炮兵连2=命中;} -> Skip}
 }      |||
        概率选择
        {       [0.05]: 开火?3 ->
                        原子型{τ{射击_结果_的_炮兵连3=未命中;} -> Skip}
                默认: 开火? 3 ->
                          原子型{tau{射击_结果_Of_炮兵连3=命中;} -> Skip}
 };
/* 此代码会为每次炮兵连的射击随机设置命中或未命中效果
炮兵连 */

引爆();
引爆()= 引爆! 0 -> 场地就绪();
//=============================================================
```

图78：战场组件的 CSP 表示

```
// 野战炮兵组合模型
//=====================================================================

野战炮兵= ObserverSM || BHQSM || FieldSM || BatterySM(1)||
BatterySM(2)|| BatterySM(3)

// || 是所有组件之间的并行运算符
// BatterySM 有三个实例，使用电池编号参数进行初始化。
```

图79：野战炮兵组合模型

图79展示了转换后的组件如何使用并行算子'||'组合的CSP表示。这意味着所有组件并行执行，但在各自的通信通道中交换事件时会进行屏障同步。

9.3.4 野战炮兵模型的模型检验

基于CSP的野战炮兵模型可在PAT工具中打开并执行。该模型成功编译表明其中没有错误。执行该模型时，如果每个组件都到达其最终状态，则说明需求规范的约束S3b得到满足，也就是说，转换后的可执行模型正确地表示了其概念模型的行为。

在验证过程中，我们定义以下断言，并使用 PAT 内置的模型检验器进行验证。由于输入模型具有概率性和实时性，我们使用 PAT工具的 **概率实时模块**。

图80展示了如何使用 PAT 的概率 c
CSP 线性时序逻辑规范。

```
//=============================================================
// 野战炮兵可组合性验证
//=============================================================

// 目标状态可达性
#断言 野战炮兵 |= [](请求火力支援.0 -> <>引爆.0);

// 目标定义
#定义 目标 (射击_结果_的_炮兵连1==命中
                && 射击_结果_的_炮兵连2==命中
                && 射击_结果_的_炮兵连3==命中);

//ASSERT2: //目标可达性
#断言 野战炮兵 |= <>目标 with prob;
```

图80：野战炮兵验证断言

断言1使用线性时序逻辑构造来验证：如果发生“请求火力支援”，那么最终会在目标位置引爆。如果断言1成立，则表明存在一条通向目标状态的有效执行路径。

PAT模型检查器的结果如图81所示，这表明目标状态是可达的。

```
********验证结果********
断言 (野战炮兵() |= [](( 请求火力支援.0-><> 引爆.0)) 为 有效。

********验证统计********
已访问状态：160477
转换总数:475422
耗时:8.8263759s
估计内存使用量:111668.696KB
```

图81：断言1的验证结果

断言2使用 LTL 结构来验证“目标”是否最终可达，其中“目标”定义为所有炮兵连都在目标的精确位置成功命中的条件。请注意，断言2使用**“概率构造”**，因此它是一条 PLTL 语句。图82显示的验证结果意味着，到达目标的概率介于77%和94%之间。

```
********验证结果********
断言 (野战炮兵 () |= <> 目标概率) 为有效 的概率为[0.77378, 0.94526];

********验证统计********
已访问状态：84019
转换总数：245561
MDP迭代次数：63123
耗时：5.5017483s 估计内存使用量：
97028.192KB
```

图82：断言2的验证结果

现在，我们检查是否能在目标时间属性定义的时间约束内到达目标。为进行此评估，我们使用 PAT 的 截止期限 运算符，如图83所示。我们定义了三个断言：提前、恰好和延迟。

```
//============================================================
// 野战炮兵可组合性验证
//============================================================

//目标时间约束下的目标可达性
提前 = 野战炮兵截止期限[目标时间-3];
恰好= 野战炮兵截止期限[TOT];
延迟= 野战炮兵截止期限[TOT+3];


//ASSERT3: //目标时间的目标可达性
#断言提前达到目标的概率;
#断言恰好达到目标的概率;
#断言延迟达到目标的概率;
```

图83：带有目标时间的野战炮兵验证断言

断言3的验证结果如图84所示，据此，目标的早期可达性是不可能的。而恰好在 TOT 时刻到达目标的最大概率为 86%，满足目标 O_1 和 O_2。然而，在较晚时刻到达目标的最大概率为 94%，满足 O_1 的概率更高，但不满足 O_2。

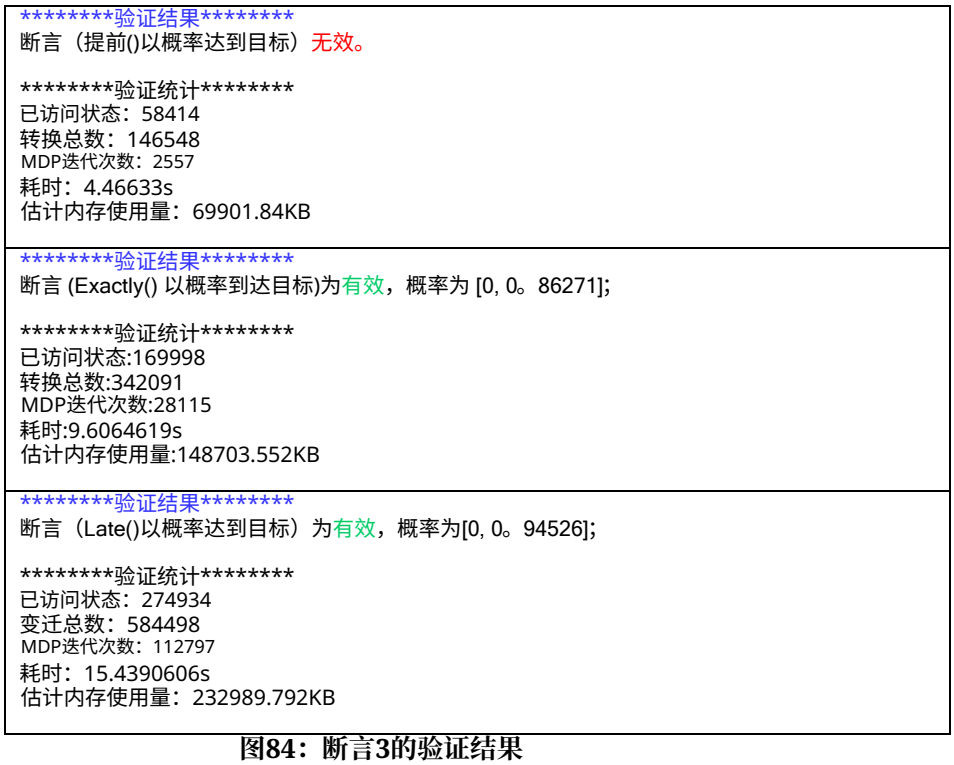


图84：断言3的验证结果

根据验证结果，我们可以说，野战炮兵模型以一定的概率满足给定的需求。由于这是一个非确定性模型，成功的可靠性取决于 BHQ 设定的目标时间期限有多紧，以及炮兵连准备得有多高效、向目标开火有多准确。

9.4 总结

本章通过一个示例介绍模型检验方法，并使用流程分析工具包（PAT）进行验证。我们修改了野战炮兵模型（第8章中的示例），将其构建为概率定时模型，以说明如何利用 PAT 采用基于 CSP 的模型检验方法，有效地进行可组合性验证。借助野战炮兵模型这一示例，本章阐述了如何验证时间约束，以及如何使用 PLTL 按概率验证不同的性质断言。若所有需求规范中定义的断言均以可接受的概率因子得到满足，则该方法验证成功，也就表明这些组件具有可组合性。

第10章总结与结论 Chapter 10
Summary and Conclusion

本章比较了本文提出的各种可组合性验证方法，并针对不同的组合模型特点，为如何选择适当的方法提供了一些指南。随后，本章总结了论文的主要贡献，并对该领域未来的研究提出了一些建议。

本论文提出了一个验证框架，该框架遵循建模与仿真领域关于模型正确性概念的基本原理。它整合了多种方法、技术和工具，以支持组合模型多层级可组合性验证过程中的不同任务。该框架还借鉴了其他领域在模型验证方面的有益技术特征，例如 Petri网、模型检验和进程代数领域，并利用这些领域积累的知识来验证基于组件的仿真模型。这些仿真模型被称为基于组件的模型，因为它们以组件形式设计，并可进一步组合以构建复杂模型（称为组合模型或组合体）。为确保正确性，组合仿真模型需要在其不同的可组合性层级上进行验证，而每个层级的验证都具有一定难度。可组合性的初始层级要求组合中的所有组件都能通过有效接口进行语法连接，并能以有效语义正确通信。更深层级的可组合性则是动态语义可组合性，它要求所有组合组件都具备适当的行为，以便彼此正确交互并实现共同目标。组件组合中行为的有效性取决于两个因素：（i）每个组件在与其他组件交互时始终处于正确状态；（ii）组合体满足需求规范中规定的行为属性要求。

所提出的验证框架不仅全面支持初始可组合性层级的验证，其最重要的特征还在于能够在更深层次上验证组合模型组件的动态语义可组合性。这一层次的可组合性验证是一项艰巨的任务，需要采用动态分析方法。组件的行为可以通过观察它们在执行环境中相互作用时的表现来研究：它们通过交换事件进行通信，并通过内部状态的变化不断推进。因此，需要一种恰当的动态分析方法，不仅能提供合适的执行环境，还能支持内置验证技术，以评估运行时的可组合性行为。

根据我们的研究结果，由于动态语义可组合性层本身具有复杂性，没有一种方法能够完全涵盖证明正确性所需的所有复杂细节。某种方法的有效性也会因组合模型及所用建模形式体系的不同而有所变化。由于

有些模型结构复杂，在抽象系统时需要对以数据为中心的细节具备丰富的表达能力；而另一些模型则具有复杂的行为特性，包括并发和时间约束等概念。此外，系统行为可能是确定性的，也可能是随机性的。因此，对于动态语义可组合性验证这一具有挑战性的任务，很难依赖单一方法。

因此，我们在框架中研究了三种不同的动态分析方法，即：（i）基于PN的代数分析；（ii）基于CPN的状态空间分析；（iii）基于CSP的模型检验技术。这些方法继承了各自源领域（如PN、CSP和模型检验）的理论、方法、工具和技术。我们对这些继承而来的资源进行调整，并将其整合到框架中。此外，我们还在每种方法中提出了若干扩展，以满足动态语义可组合性验证的需求。其中一些扩展如下：

▪ 本文提出了一种基于组件的描述格式，用于以所需形式表示基于BOM的组合模型，以便应用选定的方法。例如，本文提出了一种基于CPN的组件模型，用CPN模型的形式表示BOM组件的结构和行为方面。类似地，对于CSP，本文引入了一种面向组件的CSP进程模型，使用CSP表示法描述BOM组件。▪ 针对每种方法，本文提出了一种基于规则的转换技术，可将BOM组件转换为相应方法的描述格式，同时保留模型的结构和行为。为确保这一点，本文提出了比较原始模型（BOM）与转换后模型的方法，以确认后者能够正确表示前者。▪ 针对PN代数方法，本文提出了用于自动化这一过程的算法。此外，还开发了一个函数库，以便于重复执行验证任务。▪ 对于状态空间分析，本文提出了一种约简技术，有助于缩减大型状态空间，从而简化验证过程。

这三种方法的优缺点分类如下：

优势　劣势　中性

类别：	可验证的属性类型
PN代数分析	这种方法只能验证有限数量的属性，因为它依赖于 取决于底层数学定理的适用性，而这些定理数量有限， 可能无法涵盖所有类型的属性
CPN 状态-空间分析	它构建系统可能处于的所有状态所组成的状态空间。因此 它允许指定并验证不同类型的一般系统性质，例如 以及场景特定属性。
基于 CSP 的模型检验	在这种方法中，验证取决于属性规范 使用线性时序逻辑或计算树逻辑断言，这些断言及其各种扩展 为定义不同类型的属性提供了丰富的表达力。因此，它 涵盖范围更广的验证问题，既包括通用方面， 也包括场景特定属性

表 55：可验证的属性类型　　ied

类别：	可验证的模型类型
PN代数分析	此方法支持简单的事件驱动PN模型，但不支持包含丰富数据的模型，也不支持实时系统或概率系统的模型。
CPN 状态-空间分析	此方法支持具有丰富的以数据为中心的结构和行为的模型，因为它支持复杂数据类型令牌的流动及其操作在变迁期间。它还对定时系统提供有限支持。然而，它不支持概率性质的模型验证。
基于 CSP 的模型检验	这种方法限制了模型中的信息量，并且不支持具有丰富以数据为中心的表达能力的模型。然而，它提供了多种可验证的系统类型，例如反应式系统、实时系统、概率系统和随机系统。因此，这种方法在验证不同类型的系统方面更为强大。

表 56：可验证的模型类型

类别：	可扩展性
PN代数分析	验证取决于Petri网模型的结构（即库所和变迁的数量）。这一因素远小于其他方法所产生的可达标识数量。因此，对于较大的模型，这种方法证明具有可扩展性
CPN 状态-空间分析	验证取决于状态空间，而状态空间往往会在即使是普通模型中也变得很大，因此很容易发生状态空间爆炸。一些约简技术（包括我们自己提出的一种）可以降低这种风险，但无法将其彻底消除。
基于 CSP 的模型检验	模型检验同样面临状态空间爆炸，不过它经历了算法的持续改进以及紧凑数据结构的发展，从而降低了这一风险。因此，与状态空间分析相比，它有望更好地解决可扩展性问题。

表57：可扩展性

类别：	无限模型验证
PN代数分析	无论模型是有限的还是无限的，其推理过程都不会受到影响，因为在在大多数情况下，它使用由以下内容推导出的不变量进行推理：代数计算，且不依赖于可达系统状态的数量
CPN 状态-空间分析	如果模型是无限的，则需要构造无限状态空间这是不可行的。
基于 CSP 的模型检验	可以通过采用有界模型检验，或将无限系统抽象为有限系统，来使用这种方法验证无限模型；不过这可能导致结果仅具备部分正确性。

表58：无限模型验证

类别：	可用性
PN 分析	这种方法因涉及复杂的数学内容和需求而难以使用，并且需要运用相关定理进行正确推理。
CPN 状态-空间分析	这种方法易于使用。大多数操作都是自动完成的。
基于 CSP 模型检验	这种方法需要花费一些精力来理解所用的形式化方法，以便进行模型输入和性质规范。不过，其操作简单，而且全部都在黑盒中运行，即，模型检验器。

表59：可用性

类别：	自动化
PN代数 分析	这种方法并非自动化，因为性质的定义及其定理适用性需要人工完成。当定义了某个性质，并且选择定理后，建模者必须进行数学计算并手动推断是否满足某个条件。
CPN 状态-空间分析	这种方法是半自动的，因为定义验证任务以及合适的验证函数需要建模者提供输入。然而，执行函数的这一部分是自动的，它会搜索整个状态空间并返回结果。
基于 CSP 的模型检验	这种方法完全自动化。一旦定义了时序逻辑断言，它就会由模型检验器自动执行，以确定它是否满足要求；否则会生成一个反例。

表 60：自动化

表 55从可验证属性的不同类型方面对所提出的方法进行了比较。结果表明，PN代数技术仅限于验证一般属性（如死锁、活性、公平性），因为它依赖于底层定理来证明这些属性的可满足性。相比之下，另外两种方法在规范和验证不同类型的属性方面更加灵活，既包括一般属性，也包括场景特定属性。**表 56**从模型类型方面对所提出的方法进行了比较。PN代数方法仅支持具有简单事件且不带任何参数、守卫条件、动作或输入/输出状态变量的Petri网模型。基于CPN的状态空间分析方法则支持这些特性，也有限度地支持基于时间的 CPN 模型。不过，对于复杂实时系统或概率系统的模型，模型检验方法是更合适的选择。

表57从模型可扩展性的角度比较了这些方法。对于代数技术，属性验证中的大多数操作都需要进行矩阵计算，例如关联矩阵、P-不变量和T-不变量。因此，可扩展性取决于矩阵的规模，即组合模型中的库所数× 和变迁数。因此，代数技术的可扩展性相对较好。就可扩展性而言，基于CPN的状态空间方法由于具有丰富的数据表达能力和枚举特性，存在严重局限。据报告[**130**]，如果模型非常庞大，生成的状态空间约有10^5-10^6 个节点。因此，普通个人计算机难以处理如此庞大的状态空间。不过，也有不同的方法可以提升其可扩展性。我们还认为

如果将我们提出的状态空间缩减技术直接应用于CPN工具环境，这一局限性还可以进一步缓解。模型检验技术的可扩展性相对较好，因为它依赖于PAT工具，而该工具能够在合理的时间内处理约 10^7 个状态 [98]。这应足以验证大多数工业规模系统模型。

根据**表58**，无论模型本质上是有限还是无限，代数方法都同样适用。无限模型是不断演化、永不终止的模型。由于无法构造其状态空间，使用状态空间方法难以验证这类模型。尽管已有一些技术（如可覆盖图）被开发出来以解决这一问题，但在某些情况下仍会失效，例如定时模型。通过有界模型检测，或将无限系统抽象为有限系统，可以在一定程度上使用模型检验来验证无限模型。然而，这可能只能得到部分正确性的结果，因为在推理属性是否正确时，只能考虑系统状态空间的一部分。**表59：可用性表59**和**表 60**比较了这些方法在验证任务中的应用难易程度，以及它们所提供的自动化程度。

简而言之，没有一种方法能够胜出一切；正确选择方法完全取决于所研究的模型类型以及待验证的属性类型。虽然没有确切的规则，但可以依据一些基本指南，帮助建模者选择合适的方法：

10.1 方法选择指南

本节为建模者进行适当选择提供了一些基本指南

10.1.1 PN代数技术

当需要分析具有简单状态/变迁且不需要任何扩展的 BOM组合时，此方法最为适用（即不包含状态变量，也不包含带参数、守卫条件、动作、输入和输出等复杂概念的变迁）。此外，其需求规范包含可以转换为 PN 属性的属性（对此存在 PN 代数验证的解法）。因此，当需求规格可以用 PN 属性来定义时，应采用此方法。例如，在第7章中，目标被转换为“公平性”，这意味着如果模型是公平的，则这些目标可以得到满足，因此验证的目标是证明这一假设，而这可以使用 PN代数方法完成。此外，该方法不受模型规模影响，因为它对阶数为（库所数 × 变迁数）的矩阵执行计算，而矩阵的阶数保持不变，因此也可用于规模较大的模型。

如果需求规范包含可达性属性，则不应使用此方法。虽然可以使用 PN 状态方程验证这些属性，但与状态空间分析方法相比，这样做相当困难且效率低下。如果组合模型包含时间概念、彩色令牌（即 BOM 事件带有参数）或非确定性，则无法使用此方法。

10.1.2 基于CPN的状态空间分析技术

当给定模型具有（或需要）丰富的以数据为中心的结构和行为（例如状态变量、事件参数、守卫条件和动作）时，此方法最为适用。在这种情况下，需要扩展 BOM 组件以捕获更多细节。如果建模者掌握扩展 BOM 组件所需的信息，就应采用此方法；否则，应选择代数技术。如果建模者希望在抽象层执行组合模型，以便在实际实现组件之前研究其行为，此方法也很适用。尽管其他提出的方法也提供执行／仿真环境，但基于 CPN 的执行更为详细、全面，有助于研究组合组件之间的交互：它为 CPN 组件提供分层互连，并通过数据流展示组件的执行过程；这些数据以彩色令牌的形式在各组件的输入和输出之间流动，执行方式可以是交互式、逐步式或自动式。这样，建模者就能在运行时环境中仔细检查组合及其动态特性。从基于构件的开发角度来看，采用此方法有许多益处；而我们提出的名为“组合状态空间”的状态空间缩减技术进一步提高了其成功的可能性，并降低了状态空间爆炸的风险。

由于 CPN环境支持对定时系统进行建模和验证，因此该方法也可用于定时系统。不过，这种方法存在一些局限性：定时系统的状态空间在计算和内存方面开销更大，因为每个状态都附带时间戳开销，所以即使是较简单的模型，其状态空间也会比没有时间因素的类似模型大得多。此外，如果模型中存在一个非终止循环，就无法构建其状态空间，因为时间戳不断递增会使状态空间无限增长。（也就是说，系统可能在循环中返回先前的状态，状态空间中并未添加新状态，但时间在增加，因此时间戳会持续递增。因此，同一状态会因时间戳不同而被无限次添加。）

然而，当模型中涉及某些非确定性时，该方法会完全失效。尽管 CPN 规范允许使用不同的概率分布函数，但使用这些函数后生成的状态空间会有所变化，因此无法用于验证推理。因此，如果模型具有随机性，我们不建议采用该方法。

10.1.3 基于CSP的模型检验技术

这种方法通常受到软件验证领域大多数人的青睐。它在采用满足特定需求的新技术或算法方面也具有更大的灵活性，因此可用于各种语境。这种方法允许建模者使用 PAT模拟器（见 3.2.7）执行组合模型，因此在运行时研究系统行为方面，也能与基于CPN的状态空间分析相媲美。然而，这种方法的主要优势在于，它能通过模型检验，针对各种验证问题以及各种类型的系统（实时、概率等）提供答案。

然而，这种方法限制了模型的表达力，因为它限制了字符串、产品和记录等数据类型的使用（与 CPN 不同）。这要求建模者付出额外努力，使用较小的数据类型，以简化或紧凑的形式表示模型。

例如，可以使用布尔标志代替字符串参数，例如，一对字符串参数：“目标_已摧毁”、“目标_未命中”可以用 True/False 表示。类似地，一组字符串参数：{“红色”, “蓝色”, “绿色”}可以用相应的整数值 {0, 1, 2} 表示。要使这种方法正确运行，就需要进行这类简化。
例如，在第8章中，我们提出了一个以数据为中心的野战炮兵详细模型，并采用基于CPN的方法进行验证。但当需要验证某个特定的定时性质（具有非确定性）时，我们便删去了不必要的细节，并在第9章中提出了一个更简单的野战炮兵模型原型，只关注与目标性质相关的行为。通过这一修改，该模型得以适用于此方法，并成功验证了无法使用基于CPN的方法验证的需求性质。

最后需要指出的是，每种方法都有其优点和缺点，具体选择取决于建模者的目标、任务性质以及可用信息。不过，我们也鼓励针对单项任务采用多种方法，并对结果进行比较。这样可以从不同角度进行考察，更有助于确认正确性。

10.2 论文贡献

基于组件的建模与仿真是开发和仿真系统模型的一种很有前景的方法。它具有诸多优势，例如模块化设计、逻辑分离、灵活的变更管理、现有组件的可重用性以及跨领域模型集成，因此有助于降低成本、时间消耗和系统复杂性。这种高效范式的一项关键特征是可组合性，即通过添加或选择并组装可复用组件来满足用户需求的能力。在本文中，我们主要致力于研究基于组件的模型设计这一质量特征的不同方面，并提出了一个用于评估其正确性的可组合性验证框架。我们提出的框架采用基础对象模型（BOM）——一种用于基于组件的建模的SISO标准——并根据给定的需求规范，对基于BOM的模型组合进行可组合性验证。为证明一组BOM组件的可组合性正确性，我们的框架遵循规定的验证过程，该过程包含多个阶段：从系统抽象、需求收集、BOM组件选择，到将其组合形成概念模型，再以迭代的自顶向下细化方式验证其不同层次的可组合性。整个过程成功完成后，即可认为组合模型已根据其规格通过验证，并可使用实现架构（如高层体系结构）进行实现和仿真，以达到预期目的。

以下是本论文的主要贡献：
- 我们开发了一种可组合性验证框架，它建立在基本验证原理之上，并以建模与仿真的理论基础为支撑，其详细内容主要在第一部分中介绍。该框架整合了不同的方法、技术、范式、算法、形式化方法、模板、工具和 3rd 方库（或 API），以支持多层级可组合性中的不同任务

组合模型相对于其需求规范的验证过程。

- 我们通过对基于组件的建模与仿真（CBM&S）生命周期的不同阶段及各阶段的活动进行分类，概述了该生命周期。我们使用图示说明了各阶段开展的不同任务。该生命周期为使用框架的各种功能提供了指南，并使用户能够以系统化的方式执行验证操作。

- 本文提出了一种以形式化方式定义和表达需求的模板。我们的需求规范模板可用于指定一组目标和系统约束。目标可视为最终目的，而约束则是为实现这些目标必须满足的必要质量要求。

- 受模型开发的发现、匹配与组合（DMC）范式启发，[19]，我们提出了一种快速开发基于BOM的概念模型的方法。

- 我们提出了BOM组件及其组合体的形式化描述，以供文档记录之用。我们还提出了一种图形表示法[38]，用于描述BOM组件的结构，并以紧凑的形式展示它们之间的连接方式。此表示法可作为不同模型组合的蓝图，并可在不同团队之间共享，或存档于存储库中以供参考。

- 我们提出采用基于规则的静态分析技术来评估组合BOM的结构一致性。结构分析包括检查组件是否正确连接，以及它们能否以正确的语义彼此通信。对于语义分析，我们提出一种基于 OWL 的差异比较方法，用于检查组件之间的通信在语义上是否一致、有意义，并且是否按预期得到理解。

- 我们提出一种行为评估技术，用于判断组件能否按照正确的因果顺序彼此交互，从而到达最终状态或经过目标状态。为此，我们提出状态机匹配过程：将各组件的 BOM状态机转换为可执行的 SCXML格式并执行，以分析它们之间的交互。如果不存在死锁，且所有状态机都取得了所需进展，则报告组件的行为是一致的。

- 为评估动态语义可组合性层，我们的框架纳入了三种主要方法：（a）PN代数技术；（b）基于CPN的状态空间分析技术；（c）基于 CSP 的模型检验。这三种方法可作为彼此的替代方案，其选择取决于所研究模型的性质以及建模者的决定。我们还提出了基本指南，帮助建模者选择合适的方法。

- 针对每种方法，我们都开发了自动转换工具，将基于 BOM 的组合模型转换为相应的可执行模型描述形式。此方法受到模型驱动架构的启发，在该架构中，平台无关模型通过

[38] 需要注意的是，状态图和顺序图等不同的 UML 图被用于非形式化地描述 BOM。我们的图形表示法遵循基于组件的软件工程的模式。

一些转换规则转换为平台相关模型。我们还提出了基于某些额外细节的BOM 扩展，这些细节是正确转换所必需的。为此，我们开发了 BOM 扩展编辑器，用于接收建模者输入以扩展 BOM 组件。

- 我们分别在第7、8和9章中讨论的三个不同案例研究中应用了所提出的方法。每个案例研究都提供了概念验证，并验证了框架的特定特征。对于基于 PN 的代数技术，我们展示了一个制造系统，并验证了其中的公平性。对于基于 CPN 的状态空间分析方法，我们展示了一个野战炮兵模型，并验证了一组场景特定属性。对于模型检验，我们将同一个野战炮兵场景改为定时非确定性模型，并在若干概率假设下验证了一项特定的时间属性。

- 我们引入了一种基于 CPN 的组件模型，用于将 BOM组件（或任何其他仿真组件）描述为可执行模型，并通过 CPN 执行环境运行。该 CPN组件模型还可以表示任何其他仿真组件，它由三个层构成，即：(i) 结构层：用于定义组件属性和变量；(ii) 行为层：用于描述组件的状态机；(iii) 通信层：用于描述组件接口，以及组件如何与其他组件连接和通信。我们将所有 BOM组件转换为所提出的基于 CPN 的组件模型，并将其组合成一个组合模型。该模型可在 CPN环境中运行，并使用基于 CPN 的状态空间分析技术进行验证。

- 我们介绍一种称为组合状态空间的状态空间缩减技术。该技术假设所有组合组件均为黑盒，其输入和输出在主模型中进行交换。因此，我们可以从状态空间中选出与主模型中发生的任何活动相关的所有节点，并通过将其他节点替换为边来过滤掉这些节点。所得图是一个约简状态空间，仅表示描述主模型中组件交互的节点，并为可组合性验证提供足够的信息。

10.3 结论

本论文提出的验证框架加快了基于 BOM 的组合模型相对于需求规范的可组合性验证过程。经过验证的组合模型能够确保结构和行为一致，并保证满足其目标和所需约束。借助发现、匹配与组合范式快速开发概念模型，将其自动转换为可执行形式，并对其进行可组合性验证，有助于根据给定的需求规范研究其结构和行为正确性。这有助于在模型设计真正投入实施和仿真、发挥其作用之前，纠正模型中可能存在的缺陷，从而节省大量时间和成本，并增强稳健性。此外，这一过程有力地支持可重用性，因为整个过程可以轻松重复，以便针对不同场景和不同配置，或不同需求规范，组合相同的组件（如第8章和第9章所述）。

整个可组合性验证框架依托于系统化的基于组件的建模与仿真生命周期，该生命周期概述了基于组件的建模与仿真开发过程中的不同阶段及其各自的活动。该生命周期继承了现有的一些建模与仿真开发生命周期以及模型驱动架构的重要特征，并通过扩展基于组件的模型开发，为建模者提供必要的指导，帮助他们在不同阶段执行不同任务。

这一生命周期的一个重要特征是软件工程中的自顶向下细化原则。根据这一原则，概念模型通过若干中间步骤细化为可执行形式。每一步都会生成抽象模型的一个相对详细版本，并且基于先前已验证版本的假设，更容易对其正确性进行推理。例如，当状态机匹配过程成功后，我们便可以在假设组合组件的行为一致的前提下，进一步进行更详细的动态层级执行／验证。

我们在三种不同动态分析方法上的经验，证明这些方法对可组合性验证非常有益。每种方法都以自己的方式显著提升了对模型正确性进行推理的效率和准确性。我们认为，跨领域共享现有知识并借鉴其他领域（例如本研究中的Petri网、CSP 和模型检验）的宝贵成果，有助于加强问题求解中的合作，并推动高质量研究的开展。

10.4 未来方向

本研究未来的一些主要方向包括：

- 我们计划将可组合性验证框架应用于不同的应用领域，以评估其潜力，并在验证中利用其有价值的特性。其中一个领域是机器人应用的基于组件的设计。许多机器人应用的软件架构都支持面向组件的设计，因此可以探索如何采用我们的可组合性验证过程，例如研究不同机器人应用中行为可组合性的各个方面。

- 在改进验证框架方面，以下是一些重要的未来方向：
  - 我们计划纳入对需求规范的验证。正确性是成功进行验证的必要条件。
  - 我们还计划提出一种可行的解决方案，用于根据真实系统验证组合模型。
  - 我们在第2章中定义了实用组合性级别，不过这一层级的可组合性验证仍在研究中。我们打算未来探索这一方向。

- 总的来说，我们希望探索基于组件的设计优化领域，并研究面向多目标优化的组件设计的可组合性。

参考文献 References

[1] Eric Winsberg, 《计算机模拟时代的科学》。芝加哥大学出版社，2010年。[2] Louis G. Birta and Gilbert Arbez, 《建模与仿真：探索动态系统行为》，第1版。施普林格，2007年。[3] Christopher A. Chung, 《仿真建模手册：一种实用方法》，第1版。CRC出版社，2004年。[4] Catherine M. Banks, “什么是建模与仿真？”，载于《建模与仿真原理：一种多学科方法》。弗吉尼亚州诺福克：Wiley，2009年，第1章。[5] O Balci, J D Arthur, and W F Ormsby, “借助仿真概念模型实现可重用性与可组合性”, 《仿真学报》，第5卷，第3期，第157-165页，2011年8月。[6] Charles W. Krueger, “软件复用”, 《ACM计算评论》，第24卷，第2期，第13183页，1992年。[7] Johannes Sametinger, 《采用可复用组件的软件工程》，第1版。施普林格，2001年5月25日。[8] Robert G. Bartholet, David C. Brogan, Paul F. Reynolds, and Joseph C. Carnahan, “寻找点金石：仿真可组合性与基于组件的软件设计之比较”，载于《秋季仿真互操作性研讨会论文集》，佛罗里达州奥兰多，2004年。[9] Ivica Crnkovic, Brahim Hnich, Torsten Jonsson, and Zeynep Kiziltan, “基于组件的软件工程基本概念”，载于《构建可靠的基于组件的软件系统》。美国马萨诸塞州：Artech House，2002年，第1章。[10] Clemens Syperski, 《面向对象编程之外的组件软件》，第2版。纽约：艾迪生-韦斯利，2002年。[11] Elfatatry Ahmed, “应对变化：组件与服务之比较”, 《ACM通讯》，第50卷，第8期，2007年8月。[12] Marko Hofmann, “基于组件的军事仿真：地面作战仿真系统的经验教训”，载于《第15届欧洲仿真研讨会论文集》，荷兰代尔夫特，2003年。[13] Andreas Tolk, “互操作性与可组合性”，载于《建模与仿真基础：理论基础与实践领域》。约翰·威立，2010年，第12章。[14] Judith A. Stafford and Kurt Wallnau, “组件组合与集成”，载于《构建可靠的基于组件的软件系统》。美国马萨诸塞州：Artech House公司，2002年，第179 - 192页。[15] Scott A. Hissam, Gabriel A. Moreno, Judith Stafford, and Kurt C. Wallnau,

“利用预测赋能构件技术封装可预测装配”，卡内基梅隆大学，匹兹堡，宾夕法尼亚州，技术报告 CMU/SEI-200TR-024，2001年11月。[16] Stephen Kasputis，“可组合仿真”，载于冬季仿真会议，美国奥兰多，2000年，第1577–1584页。[17] Paul K. Davis 和 Robert H. Anderson，提升国防部模型与仿真的可组合性。兰德国家防务研究所，2003年。[18] Hessam S. Sarjoughian，“模型可组合性”，载于冬季仿真会议，美国加利福尼亚州蒙特雷，2006年。[19] Farshad Moradi，“利用BOM和语义网技术实现基于组件的建模与仿真框架”，瑞典皇家理工学院信息与通信技术学院，博士学位论文 KTH/ICT/ECS AVH-08/05— SE，2008年。[20] Claudia Szabo，“可组合仿真模型及其形式化验证”，新加坡国立大学计算机科学系，新加坡，博士学位论文，2010年。[21] Andy Ju An Wang 和 Kai Qian，面向组件编程，第1版：约翰·威立父子出版公司，2005年。[22] Ernest H. Page，“仿真互联理论与实践：国防仿真中的互操作性与可组合性”，载于《动态系统建模手册》：Chapman & Hall，2007年，第16章。[23] Andreas Hansson 和 Kees Goossens，采用 *Aelite* 的片上互连：可组合且可预测的系统，第1版。纽约：施普林格，2011年。[24] Hsu-Chun Yen，“Petri网理论导论”，载于《形式语言与应用的最新进展》：施普林格柏林，2006年，第25章。[25] Christos G. Cassandras 和 Stéphane Lafortune，离散事件系统导论，第2版：施普林格，2008年。[26] James L. Peterson，*Petri*网理论与系统建模，第1版：普伦蒂斯霍尔公司，恩格尔伍德克利夫斯，新泽西州 07632，1981年。[27] Jos C M Baeten，“进程代数简史”，《理论计算机科学——进程代数》，第335卷，第2–3期，第131–146页，2005年5月。[28] Osman Balci，“仿真模型的验证、确认与认证”，载于冬季仿真会议论文集，亚特兰大，佐治亚州，1997年。[29] Mikel D. Petty，“验证与确认”，载于《建模与仿真原理》：约翰·威立父子出版公司，2009年，第6章。[30] Stewart Robinson、Roger Brooks、Kathy Kotiadis 和 Durk-Jouke Van Der Zee，离散事件仿真的概念建模。美国佛罗里达州：CRC出版社，

博卡拉顿，2010年。

[31] Paul A. Fishwick，仿真模型设计与执行：构建数字世界（第*1*版）。美国新泽西州：Prentice Hall PTR，1995年。[32] Stephan MERZ，《模型检验导论》，载于实时系统建模与验证，Nicolas Navet 和 Stephan Merz 编：Wiley，2010年，第3章。[33] Susan Harkrider 和 Lunceford H. W.，《建模与仿真的可组合性》，载于军方/工业界训练、仿真与教育会议论文集，佛罗里达州奥兰多，1999年。[34] Mikel D. Petty 和 Eric W. Weisel，《仿真可组合性理论》，老道明大学弗吉尼亚建模、分析与仿真中心，弗吉尼亚州诺福克，2004年。[35] Ernest H. Page 和 Jeffrey M. Opper，《关于可组合仿真复杂性的观察》，载于冬季仿真会议论文集。，新泽西州，1999年，第553–560页。[36] David R. Pratt、Charles L. Ragusa 和 Sonia von der Lippe，《可组合性作为架构驱动因素》，载于军方/工业界训练、仿真与教育会议 *(I/ITSEC)*，1999年。[37] Paul K. Davis、Paul A. Fishwick、Michael C. Overstreet 和 Dennis C. Pegden，《模型可组合性作为研究投资：对专题论文的回应》，载于冬季仿真会议，佛罗里达州奥兰多，2000年，第1585–1591页。[38] Mikel D. Petty 和 Eric W. Weisel，《可组合性词汇表》，载于*2003*年春季仿真互操作性研讨会论文集，佛罗里达州奥兰多，2003年4月。[39] 可扩展建模与仿真框架 (XMSF)。
https://www.movesinstitute.org/xmsf/xmsf.html

[40] DEVS。 http://en.wikipedia.org/wiki/DEVS

[41] OSA. http://osa.inria.fr/publications.html

[42] 基础对象模型。 www.boms.info/

[43] Paul Davis，《可组合性》，载于国防建模、仿真与分析：迎接挑战。华盛顿哥伦比亚特区：美国国家科学院出版社，2006年。

[44] Jay Larson、Robert Jacob和Everest Ong，《模型耦合工具包：用于构建多物理场并行耦合模型的新型Fortran90工具包》，《国际高性能计算应用杂志》，第19卷第3期，第277–292页，2005年。[45] Simon Portegies Zwart、Steve McMillan等（共24位作者），《用于天体物理系统建模的多物理场与多尺度软件环境》，载于第八届国际计算科学会议，柏林，2008年，第207–216页。

[46] Rob Armstrong等，《面向高性能科学计算的CCA组件模型》，《并发计算：实践与经验》，第18卷第2期，第215–229页，2006年。[47] Maciej Malawski、Marian Bubak、Michal Placek、Dawid Kurzyniec和Vaidy Sunderam，《跨越网格边界的分布式组件计算实验》，载于*HPC-GECO/CompFrame*研讨会论文集，法国巴黎，2006年。[48] Jan Hegewald、Manfred Krafczyk、Jonas Tölke、Alfons Hoekstra和Bastien Chopard，《复杂自动机的基于智能体的耦合平台》，载于第八届国际计算科学会议，柏林，2008年，第227–233页。[49] Katarzyna Rycerz和Marian Bubak，《构建和运行协作式分布多尺度应用》，载于复杂系统仿真的大规模计算技术，Albert Y. Zomaya编：约翰·威立父子出版公司，2012年，第6章。[50] Eric W. Weisel、Mikel D. Petty和Roland R. Mielke，《语义中的模型及模型类别的有效性》，载于秋季仿真互操作性研讨会，佛罗里达州奥兰多，2003年。[51] Stewart Robinson、Richard E. Nance、Ray J. Paul、Michael Pidd和Simon J.E. Taylor，《仿真模型复用：定义、优势与障碍》，《仿真建模实践与理论》，第12卷第7–8期，第479–494页，2004年11月。[52] Ernest H. Page、Richard Briggs和John A. Tufarolo，《迈向仿真互连问题的成熟度模型系列》，载于仿真互操作性研讨会论文集，阿灵顿，弗吉尼亚州，2004年。[53] Brahim Medjahed和Athman Bouguettaya，《语义网服务的多层可组合性模型》，《*IEEE*知识与数据工程汇刊》，第17卷第7期，（2006年7月。[54] Farshad Moradi、Rassul Ayani、Shahab Mokarizadeh、Gholam Hossein Akbari Shahmirzadi和Gary Tan，《BOM语法与语义组合的基于规则的方法》，载于第十一届*IEEE*分布式仿真与实时应用研讨会，干尼亚，2007年。[55] Robert Porzel，上下文计算模型与应用，第1版。柏林海德堡：施普林格出版社，2011年。[56] Micheal Axelsen，《信息请求歧义与终端用户查询性能：理论与实证证据》，昆士兰大学商学院，澳大利亚昆士兰州，硕士论文，2000年。[57] Bernard P. Zeigler、Herbert Praehofer和Tag Gon Kim，建模与仿真理论，第2版：学术出版社，2000年。[58] Paul Gustavson，《为联合训练构建和使用基础对象模型（BOM）》，载于跨军种/工业界训练、仿真与教育会议（*I/ITSEC*），

佛罗里达州奥兰多，2005年。[59] SISO-I，《基础对象模型（BOM）模板规范》，仿真互操作标准组织（SISO），美国佛罗里达州奥兰多，2006年。[60] SISO-II，《基础对象模型（BOM）使用与实现指南》，佛罗里达州奥兰多，2006年。[61] Paul Gustavson 和 Tram Chase，《在概念空间与实现空间之间构建可组合的桥梁》，载于《冬季仿真会议论文集》，美国华盛顿特区，2007年。[62] Mikel D. Petty 和 Paul Gustavson，《运用高层体系结构和基础对象模型进行作战建模》，载于《作战建模与分布式仿真的工程原理》，Andreas Tolk 编：约翰·威立国际出版公司出版，2012年，第19章。[63] Robinson Stewart、Roger Brooks、Kathy Kotiadis 和 Durk-Jouke Van Der Zee，《离散事件仿真的概念建模》。美国佛罗里达州：CRC出版社，2010年。[64] Fishwick A. Paul，《仿真模型设计与执行：构建数字世界》，第1版。美国新泽西州：Prentice Hall PTR，，1995年。[65] BOM Works. http://www.simventions.com/bomworks/[66] Hruz Branislav 和 Meng Chu Zhou，《基于Petri网及其他工具的离散事件动态系统建模与控制》，第1版：施普林格，2007年。[67] ZhiWu Li 和 Meng Chu Zhou，《自动化制造系统中的死锁解除：一种新型佩特里网方法》，第1版：施普林格出版社，2009年。[68] René David 和 Hassane Alla，《离散、连续与混合佩特里网》，第1版：施普林格，2010年。[69] Girault Claude 和 Valk Rüdiger，《面向系统工程的Petri网：建模、验证与应用指南》：施普林格出版社，2001年。[70] Tadao Murata，《Petri网：属性、分析与应用，77(4)，541–580（1989）》，载于《IEEE论文集》，1989年。[71] Gianfranco Balbo，《随机佩特里网简介》，载于《形式化方法与性能分析讲义》，Holger Hermanns 和 Joost-Pieter Katoen 编：计算机科学讲义，施普林格柏林，2001年，第84-155页。[72] Zohar Manna 和 Amir Pnueli，《反应式与并发系统的时序逻辑：规范》，第1版：施普林格出版社，1992年。[73] Kurt Jensen、Søren Christense 和 Lars M Kristensen，《CPN Tools状态空间手册》，丹麦奥胡斯，手册，2006年。[74] Lars Michael Kristensen，《彩色 Petri 网的状态空间方法》，奥胡斯大学计算机科学系，丹麦奥胡斯，博士学位论文，2000年。

[75] Søren Christensen、Lars Kristensen 和 Thomas Mailund，《一种用于状态空间探索的扫描线方法》，载于系统构造与分析工具和算法：施普林格柏林 / 海德堡，2001年，第450-464页。[76] Michael Westergaard、Lars Michael Kristensen、Gerth Stølting Brodal 和 Lars Arge，《ComBack方法——通过回溯扩展哈希压缩》，载于*Petri*网与其他并发模型：施普林格柏林 / 海德堡，2007年，第445-464页。[77] Louise Elgaard，《彩色 Petri 网的对称性方法：理论、工具与实际应用》，丹麦奥胡斯，博士学位论文，2002年7月。[78] Kurt Jensen 和 Lars M Kristensen，彩色 *Petri* 网：并发系统的建模与验证：施普林格，2009年。[79] Kevin Mcleish。《Petri网》 http://www.peterlongo.it/Italiano/Informatica/Petri/index.html

[80] Standard ML。 http://www.smlnj.org/

[81] Kurt Jensen，《彩色 Petri 网》，奥胡斯大学计算机科学系，技术报告。[82] CPN 工具。 http://cpntools.org/[83] Wojciech Penczek 和 Agata Półrola，时序逻辑方法：时间 *Petri* 网和定时自动机验证进展，第1版：施普林格，2006年。[84] Charles Antony Richard Hoare，通信顺序进程，第1版：Prentice Hall International，1985年。[85] Stephen D. Brookes 和 Charles Antony Richard Hoare，《通信顺序进程理论》，《*ACM* 杂志》，第31卷，第3期，第560–599页，1984年7月。[86] Christel Baier 和 Joost-Pieter Katoen，模型检测原理，第1版：麻省理工学院出版社，2008年4月。[87] Ganesh Gopalakrishnan，《模型检验：基础》，载于计算工程：应用自动机理论与逻辑，Ganesh Gopalakrishnan 编。犹他大学，犹他州盐湖城：施普林格，2006年，第21章。[88] Malay Ganai 和 Aarti Gupta，基于 *SAT* 的可扩展形式化验证方案，第1版，Anantha Chandrakasan 编：施普林格，2007年。

[89] BDD. http://en.wikipedia.org/wiki/Binary_决策_图

[90] 布尔可满足性。 http://en.wikipedia.org/wiki/Boolean_可满足性_问题

[91] Fred Kroger 和 Stephan Merz，《时序逻辑与状态系统》，第1版：施普林格，2008年。

[92] George M. Reed 和 William A. Rosco，《通信顺序进程的定时模型》，载于自动机、语言与编程，第*13*届国际研讨会，*ICALP86*，法国雷恩，1986年，

第314–323页。

[93] 通信顺序进程http://en.wikipedia.org/wiki/Communicating_顺序_进程#分析工具

[94] 形式系统。 http://www.fsel.com/software.html

[95] 阿德莱德精化检查器 http://cs.adelaide.edu.au/~esser/弧.html

[96] ProB 动画器与模型检查器。 http://www.stups.uni-duesseldorf.de/ProB/index.php5/Main_Page

[97] PAT：进程分析工具包。http://www.comp.nus.edu.sg/~pat/

[98] Jun Sun、Yang Liu 和 Jin Song Dong，《重新审视 CSP 模型检查：介绍进程分析工具包》，载于第三届形式化方法、验证与确认应用国际研讨会，希腊，2008年。

[99] Janusz Laski 和 William Stanley，《软件验证与分析：集成式实践方法》，第1版：施普林格出版社，2009年。

[100] Jerry Banks，《仿真手册：原理、方法论、进展、应用与实践》，第1版：Wiley，1998年。

[101] 软件错误集。http://www5.in.tum.de/~huckle/bugse.html

[102] Klaus Schneider，《反应式系统验证：形式化方法与算法》，第2版：施普林格出版社，2004年。

[103] William L. Oberkampf 和 Christopher J. Roy，科学计算中的验证与确认，第1版：剑桥大学出版社，2010年。

[104] Avner Engel，工程系统的验证、确认与测试，第1版，Andrew P. Sage 编：约翰·威立国际出版公司出版物，2010年。

[105] Steve McConnell，代码大全，第2版：微软出版社，2004年。

[106] Bertrand Meyer，面向对象软件构造，第2版：普伦蒂斯霍尔，1997年。

[107] Matthew Wilson，《质量至关重要：正确性、健壮性与可靠性》，*Overload* 期刊 *Process Topics*，第93期，2009年10月。[108] Robert G. Sargent，《仿真模型的验证、确认与认可》，载于*2000*年冬季仿真会议论文集，佛罗里达州奥兰多，2000年。[109] Mourad Debbabi、Fawzi Hassaïne、Yosr Jarraya、Andrei Soeanu 和 Luay Alawneh，《系统工程中的验证与确认：评估 UML/SysML 设计模型》，第1版：施普林格，2010年。[110] Pallab Dasgupta，《形式化属性验证路线图》，第1版：施普林格，

2006年。

[111] Didar Zowghi 和 Vincenzo Gervasi，《需求的三个 C：一致性、完备性和正确性》，载于第八届需求工程国际研讨会：软件质量基础（*REFSQ'02*）论文集，2002年。

[112] OWL-API。 http://owlapi.sourceforge.net/

[113] Imran Mahmood、Rassul Ayani、Vladimir Vlassov 和 Farshad Moradi，《基于BOM的模型组合中的状态机匹配》，载于*2009*年第十三届*IEEE/ACM*分布式仿真与实时应用国际研讨会论文集（*DS-RT '09*），新加坡，2009年。

[114] 状态图XML。http://www.w3.org/TR/scxml/

[115] PNML。http://www.pnml.org/

[116] 平台无关Petri网编辑器API。 http://pipe2.sourceforge.net/

[117] Julius Farkas，《简单不等式理论》，《纯粹与应用数学杂志》，第1902卷，第124期，1902年。[118] Mario D'Anna，《使用Petri网进行并发系统分析：一种寻找网不变量的优化算法》，《计算机通信》，第11卷，第4期，1988年8月。[119] Imran Mahmood、Rassul Ayani、Vladimir Vlassov和Farshad Moradi，《使用有色Petri网验证基于BOM的组合模型的动态语义可组合性》，载于即将发表于：第*26*届高级与分布式仿真原理研讨会，中国张家界，2012年。[120] V S Alagar和K Periyasamy，《扩展有限状态机》，载于《软件系统规范》，第*2*版。施普林格，2011年，第7章。[121] Shao Jie Zhang和Yang Liu，《一种对UML状态机进行模型检验的自动化方法》，载于*2010*年安全软件集成与可靠性改进伴会，新加坡，2010年6月。[122] Marta Kwiatkowska，《公平性概念综述》，《信息与软件技术》，第31卷，第7期，1989年9月。[123] Tadao Murata和Zhehui Wu，《Petri网中的公平关系与修正后的同步距离》，《富兰克林研究所学报》，第320卷，第2期，1985年8月。[124] Imran Mahmood、Rassul Ayani、Vladimir Vlassov和Farshad Moradi，《使用Petri网验证基于BOM的组合模型的公平性》，载于*IEEE*高级与分布式仿真原理研讨会（*PADS*），法国尼斯，2011年6月。[125] Andreas Tolk，《作战建模与分布式仿真的挑战》，载于《作战建模与分布式仿真的工程原理》。John Wiley & Sons, Inc. 出版，2012年，第1章。

[126] 美国陆军，《野战炮兵连战术、技术与程序》（美国陆军野战手册，*FM 6-50*）。华盛顿：海军陆战队作战出版物，1996年。

[127] 间接射击。http://www.fas.org/man/dod-101/sys/land/indirect.htm

[128] GraphML 文件格式。http://graphml.graphdrawing.org/

[129] 目标时间。http://www.answers.com/topic/artillery#Time_于_目标

[130] Michael Westergaard 和 Fabrizio Maggi，《使用彩色 Petri 网对作战支援协议进行建模与验证》，载于*Petri* 网的应用与理论，Lars Kristensen 和 Laure Petrucci 编：施普林格柏林 / 海德堡，2011年。

[131] 军事网格参考系统http://en.wikipedia.org/wiki/Military_网格_参考_系统

[132] Thomas H. Cormen、Charles E. Leiserson、Ronald L. Rivest 和 Clifford Stein，算法导论，第 3 版：MIT 出版社，2009年。