

本科毕业论文

课题名称: 面向人机物协同的软件自动生成技术

学员姓名:	刘珺	学号:	201802001018
首次任职专业:		学历教育专业:	软件工程
命题学院:	计算机学院	年 级:	2018 级
指导教员:	董威	职 称:	教授
所属单位:	国防科技大学计算机学院计算科学系		

国防科技大学教务处

目 录

摘要	i
ABSTRACT	ii
第1章 绪论	1
1.1 研究背景及意义	1
1.1.1 研究背景与研究思路来源	1
1.1.2 研究意义与价值	2
1.2 相关研究介绍	3
1.2.1 程序综合	3
1.2.2 反应式综合	4
1.2.3 人机物融合系统与软件定义	5
1.3 论文研究内容	6
1.4 论文组织结构	7
第2章 预备知识	8
2.1 Generalized Reactive(1) 博弈	8
2.1.1 博弈结构与获胜策略	9
2.1.2 GR(1) 博弈与求解	10
2.2 组件驱动的分布式控制框架 F Prime	11
2.2.1 基本概念与特点介绍	11
2.2.2 基于模型的代码框架生成	13
2.3 本章小结	14
第3章 面向程序自动生成的无人控制系统架构设计	15
3.1 分层结构设计和功能介绍	16
3.2 F Prime 系统架构的形式化	17
3.3 主体逻辑控制模块设计实现	18
3.3.1 策略信息解析库	18
3.3.2 基于组件交互的逻辑控制实现	20
3.4 其它模块功能设计实现	23
3.4.1 全局运行控制与环境设置	23
3.4.2 软件定义动作执行模块	23
3.5 本章小结	26

第 4 章 基于规约的无人控制系统代码生成与演化方法	27
4.1 基于规约的单机器人控制逻辑综合	27
4.1.1 问题建模	28
4.1.2 规约设计与生成	30
4.1.3 基于 JTLV 套件的 GR(1) 博弈策略综合	32
4.2 基于架构模型的无人控制系统代码自动生成	34
4.2.1 单机器人场景下无人控制系统生成	34
4.2.2 多机协同场景下无人控制系统实现	36
4.3 无人控制系统代码生成集成工具原型实现	38
4.3.1 工具原型功能	38
4.3.2 工具原型使用	39
4.4 本章小结	40
第 5 章 基于 ROS 的无人控制系统综合仿真	41
5.1 ROS 机器人操作系统	41
5.1.1 ROS 通信机制与文件结构	41
5.1.2 基于 ROS 的仿真环境	42
5.1.3 ROS 导航功能包	42
5.2 单机控制系统仿真案例	44
5.2.1 仿真环境搭建	44
5.2.2 动作执行模块实现	45
5.2.3 仿真结果与分析	46
5.3 多机协同控制系统仿真实例	47
5.3.1 案例说明	47
5.3.2 协同系统实现	48
5.3.3 仿真结果与分析	48
5.4 本章小结	49
第 6 章 总结与展望	50
6.1 主体工作总结	50
6.2 未来研究展望	51
致谢	52
参考文献	52

摘要

随着信息化数字化建设逐步推进以及新型人机物三元融合理念的提出，无人系统设计开发再一次成为了软件应用领域备受关注的研究方向。区别于传统的机器主导的信息融合系统，人机物协同场景下的无人系统设计强调人类参与和多机协同。同时，随着系统规模和交互方式的复杂度提升，这对无人系统开发的规范性、高效性、可移植性和安全性提出了更高的要求。

针对上述问题，本文将组件驱动软件架构和形式化反应式综合理论相结合，以软件定义的思想抽象化具体动作行为从而实现上层无人系统和底层硬件控制之间的解耦合，针对复杂场景下的行为和协同要求，提出了一套面向人机物协同的基于规约的无人控制系统生成与演化方法。论文创新性地将反应式综合领域经典开源框架 LTLMoP 和由 NASA 喷气推进实验室 (JPL) 最新维护的 F Prime 框架相结合，具体研究内容包括：

1) 设计实现了基于 F Prime 的无人控制系统架构。该系统面向程序自动生成任务，通过分层划分为全局控制模块、逻辑控制模块以及动作执行模块，充分结合组件驱动的架构特性以及软件定义动作行为的思想，实现了综合生成的策略自动机在基于 F Prime 的无人控制系统中的解析运行。

2) 提出了基于规约的无人控制系统代码生成与演化方法。本文总结了人机物协同场景下的问题建模以及规约生成方法，将反应式综合用于单机器人逻辑控制生成，同时基于 1) 中设计实现的架构模型提出了单机器人无人控制系统架构自动生成方法。最后，多机协同机制可以通过实现 F Prime 框架下的端口完成。

3) 完成了软件原型开发以及基于 ROS 的仿真验证。论文扩展实现了用于无人控制系统自动生成的集成工具原型，实现了任务驱动的软件定义。同时，在 ROS 平台下的 Gazebo 和 Rviz 模拟器中完成了对于简单单机器人和复杂多机协同场景的可视化仿真。

该无人控制系统生成方法既具备形式化方法理论完备性和安全性，又充分继承了可重用、可扩展的框架特性，规范化了人机物协同场景下的反应式系统综合应用。

关键词: 人机物协同；反应式系统综合；软件定义；组件驱动软件架构

ABSTRACT

With the gradual advancement of informatization and digital construction as well as the proposal of the new concept of human-cyber-physical ternary integration, the design and development of unmanned systems has once again become a research direction that has attracted much attention in the field of software applications. Different from the traditional machine-led information fusion system, human participation and multi-machine collaboration play an much more important role in the design of the unmanned system under human-cyber-physical collaborative scenarios. Moreover, the increasing complexity of system scale and interaction methods puts forward higher requirements for the standardization, efficiency, reusability and security of unmanned system development.

In order to solve above problems, this paper combines the component-driven software architecture with the formal reactive synthesis theory. By introducing the idea of software define, the upper-level unmanned system and the underlying hardware control are decoupled. Aiming at the behavior and cooperation requirements in complex scenarios, we proposed a specification-based architecture generation and evolution method for unmanned control system. This paper innovatively fuses the classic open source framework LTLMoP in the field of reactive synthesis with the F Prime framework, which is newly maintained by NASA's Jet Propulsion Laboratory (JPL). Our work is composed of three themed parts:

Firstly, a single-robot unmanned control system architecture based on F Prime is designed and implemented. By dividing the model into global control module, logic control module and action execution module, we advanced an unmanned control system which is responsible for parsing and executing synthesized strategy in the form of automata. Generally speaking, the system has fully integrated the component-driven architecture and the idea of software-defined behaviors.

Secondly, we summarized a specification-based synthesis and evolution method for unmanned control systems. After modeling the problem and generating task-defined specification, we applied reactive synthesis to generate single-robot logic control, as well as presenting a single-robot unmanned control system architecture auto-generation method based on the architecture model proposed before. In addition, the mechanism of multi-robot collaboration is defined over ports in F Prime framework.

Finally, we developed a software prototype and completed ROS-based simulation verification. The tool protocol is designed for automatic generation of unmanned control system, which is realized under the idea of task-driven software define. As for the visual simulation verification, we implemented the control system in a variety of scenarios in Gazebo and Rviz simulators by making full use of ROS development.

The method proposed in this paper not only has the theoretical completeness and security derived from the formal methods, but also fully inherits the reusable and extensible framework characteristics, while standardizing the application of reactive system synthesis in various human-cyber-physical collaboration cases.

KEY WORDS: Human-Cyber-Physical Collaboration, Reactive Synthesis, Software Define, Component-driven Software Architecture

第 1 章 绪论

随着计算机科学技术的发展，软件逐步突破演变为当今社会的基石。如今，软件已经以各种形式参与、丰富、支撑着我们社会的运行，“软件定义一切”（SDE）这一理念应运而生。不论是一直所推行的数字化、信息化时代，还是最近兴起的“数字孪生”建设，这些趋势都反映着软件无处不在，它为我们的社会、每一个人赋能、赋值、赋智。

软件最初的目标就是让机器代替人类去执行重复乃至复杂的工作，因此，与机器控制直接相关的无人系统控制领域的软件研究一直是学术界和工业界的热门方向。从家用扫地机器人到无人工厂中的机器设备，乃至执行探测任务的火星车、无人机，无人系统研究已经有许多成果被应用于人们的日常生活。同时，随着新型人机物融合系统^[1] 的提出，信息技术及其应用开始从“自然世界信息化”向“信息世界自然化”转变，人们对无人系统的功能性和可验证性提出了更高的要求。

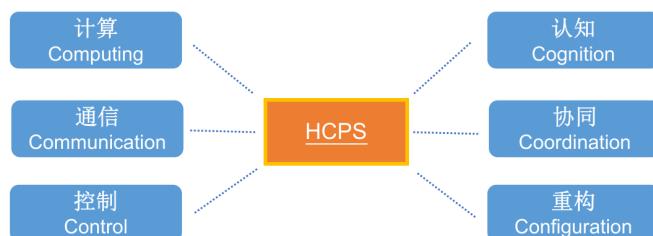


图 1-1 HCPS 的六大系统特征 (6C)

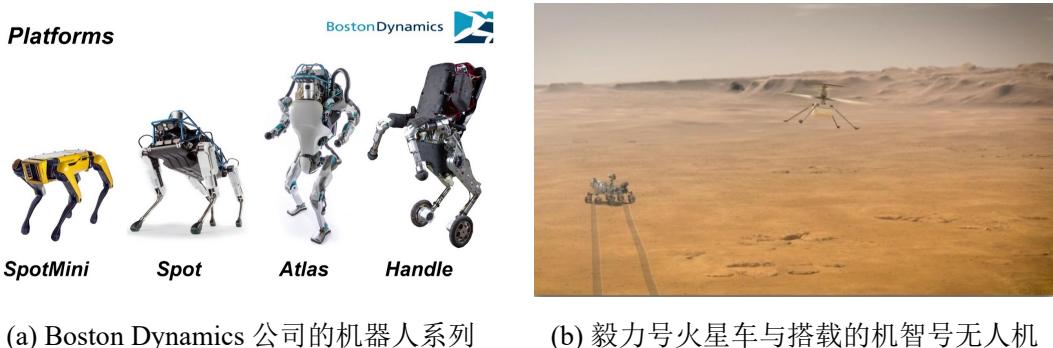
在无人系统中，控制程序作为核心软件，它控制机器获取环境感知信息并且对这些信息做出正确的行为。控制程序的功能性和安全性直接决定了无人系统的质量。因此，对控制程序的设计生成具有很重要的意义。为了规范化控制程序设计实现，同时满足“低代码”开发需求，本课题考虑将软件自动生成（程序综合）技术应用于控制程序合成。近年来，反应式综合在无人控制程序综合领域的研究进展^[2] 以及已有的工具技术^[3, 4] 对本课题提供了有力的理论支撑。

1.1 研究背景及意义

1.1.1 研究背景与研究思路来源

就国内而言，2017 年，国务院在《新一代人工智能发展规划》中就对轨道交通无人控制、服务机器人、海陆空探测机器人、智能化无人车间和高端无人控制

系统提出了明确的建设要求。而国际上，以波士顿动力为代表的顶尖机器人公司一直备受各界关注，同时，NASA 喷气推进实验室（JPL）设计实现的火星车和无人机在前段时间成功完成探测任务，无人系统一直是经济科技强国的竞争焦点，无论是民用还是军用，无人系统都具有极高的研究价值。



(a) Boston Dynamics 公司的机器人系列 (b) 豪力号火星车与搭载的机智号无人机

图 1-2 国际上具有代表性的无人系统应用实例

随着各类无人系统的智能化以及执行任务复杂化，开发和研究人员在构建无人系统时也开始广泛采用“软件定义”的方式，让无人系统以更符合软件工程规范的方式进行设计实现，这对于无人系统的安全性验证，高效开发都具有很重要的意义。本课题基于“软件定义无人系统”的思想，着眼于人机物协同场景下的“低代码开发”应用任务，即：面向用户需求，利用软件自动生成技术，让作为非开发者的用户尽可能高效简单地参与完成人机物协同任务开发。

区别于传统的开发者全程参与的无人系统控制代码开发，通过使用软件自动生成技术和软件定义技术，用户只需要描述完整的需求规约就能得到可执行的控制程序代码。而软件定义在这个过程中将上层控制程序和底层硬件驱动分离开来，通过类 API 的方式对机器具体行为进行抽象化提取，让控制系统具有更强的可移植性和复用性，有利于生成硬件无关的控制程序。

一直以来，软件自动生成技术就是软件研究领域的一个极具挑战性的任务，本课题聚焦于基于规约的无人系统控制代码自动生成，一定程度上降低了任务的复杂程度，具有较高的可行性。

1.1.2 研究意义与价值

由于环境的随机性以及任务的复杂性，大部分无人系统经常会在运行过程中出现一些非预期行为，有些非预期行为会让系统运行停滞，甚至引发错误的动作造成不可挽回的损失。功能性、健壮性、安全性、可移植性是无人系统实际应用首要考虑的因素，对于某些安全攸关场景下的无人系统，尤其对系统的安全性和

质量有较高的要求。而现有的无人系统设计实现方法往往无法对其软件质量进行保证。

通过应用形式化方法对系统构建和运行时进行检验验证在近几年也受到了很多研究人员的关注^[5, 6]。形式化方法既可以判断任务能否完成（可综合性分析），也能被用于运行时安全攸关性质的监测（运行时验证）。因此，基于形式化方法的自动程序综合，可以在理解用户意图的基础上保证控制程序的逻辑可行性，有利于生成完备、正确的无人控制系统。

另一方面，以航天器飞行软件开发（FSW）为例，受限于硬件无法访问、接口不明确、需求快速变化等问题，许多轻量级飞行器软件开发具有较高挑战性。这类无人系统对控制框架的可复用性、可移植性提出了更高的要求。通过使用软件定义的方式将硬件接口虚拟化，同时提出规范化的 FSW 框架，有助于设计底层无关的无人系统并且实现不同飞行任务下的控制程序重用。

总结而言，基于形式化程序综合的软件定义无人系统方法对于高效开发正确、安全、可移植、可复用的无人系统具有重要的意义，同时也是对已有理论知识的有效验证。

1.2 相关研究介绍

1.2.1 程序综合

程序综合（program synthesis），即程序生成，是指在底层编程语言中自动找到满足以某种规范形式表达的用户意图的程序的任务。自 20 世纪 50 年代人工智能问世以来，这个问题一直被视为计算机科学的圣杯。尽管存在一些固有的挑战，如用户意图的模糊性和程序的巨大搜索空间，程序综合领域已经发展出许多不同的技术，使程序综合能够应用于不同的实际领域。经过长久以来科研人员的研究总结，程序综合问题的研究难点主要包括三个方面，分别是：用户意图表达（user intent），可能程序搜索空间（search space）以及目标程序搜索策略（search technique）^[7]。

(1) 用户意图表达：用户意图表达作为程序综合任务的输入，是对目标程序功能需求的直接描述，即期望规约，其规范性和质量直接影响了程序综合器的正确性和效率。规约具有许多种表现形式，包括逻辑规约、输入 / 输出例子、traces 跟踪、自然语言以及完整或部分程序。例如，微软团队提出的 FlashFill 就是一个典型的基于输入 / 输出示例得到的一个可移植的程序综合工具^[8]。

(2) 搜索空间：任何一种非平凡编程语言中的程序数量都会随着程序规模的

增长而呈指数级增长，长期以来，如此庞大的候选程序数量使得这项任务变得难以处理。因此，为了一定程度上缩小程序搜索空间，很多已有研究会选取某些特定领域进行程序综合，并且对于这些领域使用特定的规约进行描述。DeepCoder 就使用了这种思想，针对算法竞赛类问题代码生成任务，定义了一个具有少数基本原语函数的域特定语言（DSL）^[9]，通过组合原语在缩小的搜索空间中实现指定功能。

(3) 搜索策略：概括而言，程序综合有两种搜索策略：演绎式综合（deductive synthesis）和归纳式综合（inductive synthesis）。演绎式综合作为经典的综合方法，它将高级（如逻辑）规约映射到可执行实现，需要提供所需用户意图的完整的形式化规范。这种方法是有效的，而且可以证明是正确的，往往用于控制器程序综合。归纳式综合则接受部分（通常是多模态）规范，通过对候选空间的符号解释来构造一个满足它的程序。相比而言，归纳式合成在需求规约上更灵活，而且最近与人工智能结合的智能归纳式程序合成也备受学术界关注^[10]。

具体到无人系统研究领域，一般而言，所需要生成的控制程序相比于一般的通用软件具有小规模、需求相对明确的特点。因此，结合形式化方法和演绎式综合对无人系统控制程序进行合成能够使无人系统开发更高效、可靠。

1.2.2 反应式综合

作为程序综合领域一个非常重要的研究分支，反应式综合关注从高层次形式化规约（一般以线性时序逻辑的形式）自动生成控制自动机（摩尔型有限状态机等）的综合过程和方法。其反应式特性主要体现在系统和环境之间高度密切的连接，系统运行自动机在环境的变化下根据规约定义做出正确的行为从而实现逻辑控制。基于这个性质，反应式综合一直是无人控制系统领域的一个热门研究内容。

Alonzo Church 最早对反应式综合给出了标准的形式化定义，即：给定一组命题变元组成的规约公式 (φ)，要求构建一个有限状态机 (C) 使得状态机对任意输入都能满足规约约束^[11]。针对这个问题，J. Richard Büchi 和 Michael Rabin 分别从两个角度将这个问题转化为给定目标获胜策略的双人博弈问题^[12] 以及树型自动机判空问题^[13]。随着研究的问题越来越复杂，系统和自动机的规模越来越大，已有研究针对不同类型的分布系统（包括流水线型，环形，异步调用型等等）分别给出了综合方法^[14–16]。

除了对系统类型进行研究，人们也提出了更加适用于反应式综合的时序逻辑语言。相比于早期基于 S1S 语言的规约描述，线性时序逻辑（LTL）的引入^[17] 将自动机构建的复杂度降低到了可行水平。线性时序逻辑加深了形式化方法在反应式

综合领域的应用，GR(1) 博弈就是定义在 LTL 子集（GR(1) 规约）上的一类成熟的反应式综合方法。GR(1) 规约定义了基于假设（Assumptions）和保证（Guarantee）的系统约束，并且能够将综合求解的时间复杂度缩小到指数空间^[18]。

JTLV 和 Slugs 是两个经典的 GR(1) 综合求解工具，能够对给定的 GR(1) 语句判定可综合性并且生成系统获胜策略^[19]。2010 年，Kress-Gazit 等人^[3] 集成了现有综合工具，将线性时序逻辑应用于无人系统任务描述，形成了从自然语言定义到控制器自动生成并且实现简单连续仿真的完整框架 LTLMoP。该开源框架被广泛应用于反应式综合领域研究。

近几年反应式综合的研究热点集中在 GR(1) 规约扩展应用以及综合系统复杂度降低两个方面，代表性工作有由 Shahar Maoz 提出的即时性 GR(1) 规约（根据环境变化即时生成策略）^[20]，以及通过限制自动机规模优化决策过程的有限型综合（Bounded Synthesis）^[21]。

1.2.3 人机物融合系统与软件定义

人机物融合系统（human-cyber-physical systems, HCPS）由信息物理系统（CPS）演变而来，聚焦社会空间、信息空间和物理空间的资源融合场景。自 2006 年在美国自然科学基金会组织的一次 CPS 研讨会中提出以来，人机物融合系统就受到了学术界和工业界广泛的关注，也被应用于基于大数据、云服务等新兴技术的智能系统的构建^[22]。

人机物融合系统相关的研究重点剖析了人类在三元融合场景中扮演的角色以及三元协同交互方式^[23]。人机物融合系统具有广泛的应用场景，包括民用场景下的智能家居、智能制造，乃至军用场景下的运输救援任务、信息化战争建设等等。开展人机物融合系统研究是工业 4.0 时代的重要全球发展战略。

随着硬件越来越成熟，硬件的功能多样性逐渐面临瓶颈，软件发展的迭代速度高于硬件，在这样的环境之下，“软件定义”的概念被越来越多的研究者提及^[24]。在工信部发布的《“十四五”软件和信息技术服务业发展规划》中深化了软件定义对于信息化科技革命和产业变革的重大意义。

作为一种方法，软件定义通过定义硬件的功能，为硬件附能。将硬件虚拟化，抽象化为 API（应用编程接口）的形式，实现了软硬件平台之间的解耦合。软件定义最早被用于网络架构的构建，即软件定义网络（SDN）^[25]，其基本原理是通过定义一组 API 将网络资源抽象到虚拟化系统中，从而在不改动网络设备的条件下实现新型的网络协议、拓扑架构。相比于传统基础架构，软件定义网络大大提高了可扩展性和灵活性。

未来是万物互联的，软件定义存储，软件定义计算，软件定义信息系统，软件定义在工业界和学术界都有重要作用。具体到本课题所讨论的无人系统，通过软件定义的方式，可以将机器人行为执行器和逻辑控制单元相分离，将底层硬件抽象化为 API，有助于生成平台无关、硬件无关的无人控制系统。这对于无人控制系统的可移植性、重用性研究具有重要意义。

1.3 论文研究内容

基于“软件定义无人系统”的思想，为了提高无人控制系统的可靠性和完备性，满足信息化时代下军事和民用场景的“低代码”无人系统开发任务，本课题应用形式化方法设计了一个完整的基于组件的无人控制系统架构。论文基于反应式综合相关理论知识，针对人机物协同场景任务进行信息建模，提出了一套基于规约的无人控制系统生成流程方法以及工具原型。

总体而言，论文基于反应式综合领域经典的 LTLMoP 工具^[3] 以及 NASA 在最新的火星车和无人机上成功部署的 F Prime 框架^[4]，将两个工具在定义良好的接口上紧密地结合，实现基于规约生成一个可执行的底层无关的 F Prime 无人控制系统。该无人系统结合了 LTLMoP 和 F Prime 的优点，既保留了形式化综合方法支撑下的完备性高效性，也具有 F Prime 框架的高度可扩展性、灵活性和可移植性。

具体而言，论文和研究工作内容包括以下三个方面：

(1) 基于 F Prime 框架的无人控制系统架构设计实现。通过设计全局控制模块、逻辑控制模块、动作执行模块并且共同组合形成了一个完整的 F Prime 框架下的上层无人控制系统架构，该系统面向程序自动生成任务，将控制策略（自动机）、系统配置信息（地图临接信息、环境信息等等）作为输入。用户可以通过使用该架构的接口和模版对动作行为进行自定义，同时也可以通过定义端口将多个单机无人控制系统连接通信实现多机协同。另外，该系统还集成了 F Prime 的事件机制用于运行时状态信息可视化。

(2) 基于规约的无人控制系统代码生成与演化方法。通过对任务进行形式化描述，利用 JTLV 工具包求解 GR(1) 博弈过程，从而基于规约综合生成策略（自动机）。本论文在 LTLMoP 工具上进行扩展开发，在规约生成自动机的基础上基于(1) 中提出的架构模型自动生成可运行的无人控制系统，得到了一个从规约描述直接生成相应 F Prime 无人控制系统的可视化工具。该工具提供了较直观的 GUI 用于无人系统“低代码”开发。

(3) 基于 ROS 的无人控制系统仿真实验。论文将基于规约自动生成的无人系统应用于具体的包含无人机和无人车的 Gazebo 环境下，通过实现无人系统中的动

作 API 组件, F Prime 无人系统作为上层控制模块, Gazebo 作为底层仿真平台, 将二者连接起来。为了让仿真环境更加贴近于实际, 论文分别在简单和复杂、单机机器人和多机协同的场景下分别进行了仿真实验。

1.4 论文组织结构

本文的内容分为五章, 前两章主要是对领域相关研究和理论知识背景进行介绍, 并且论证说明了论文思路来源以及课题的研究意义。第三章到第五章是论文的核心内容, 第三章中对 F Prime 下反应式无人控制系统的架构详细介绍, 而第四章提出了基于规约自动生成该无人控制系统的方法。该方法生成的系统被应用于第五章中描述的单机与多机协同仿真场景。论文组织结构如图1-3所示。

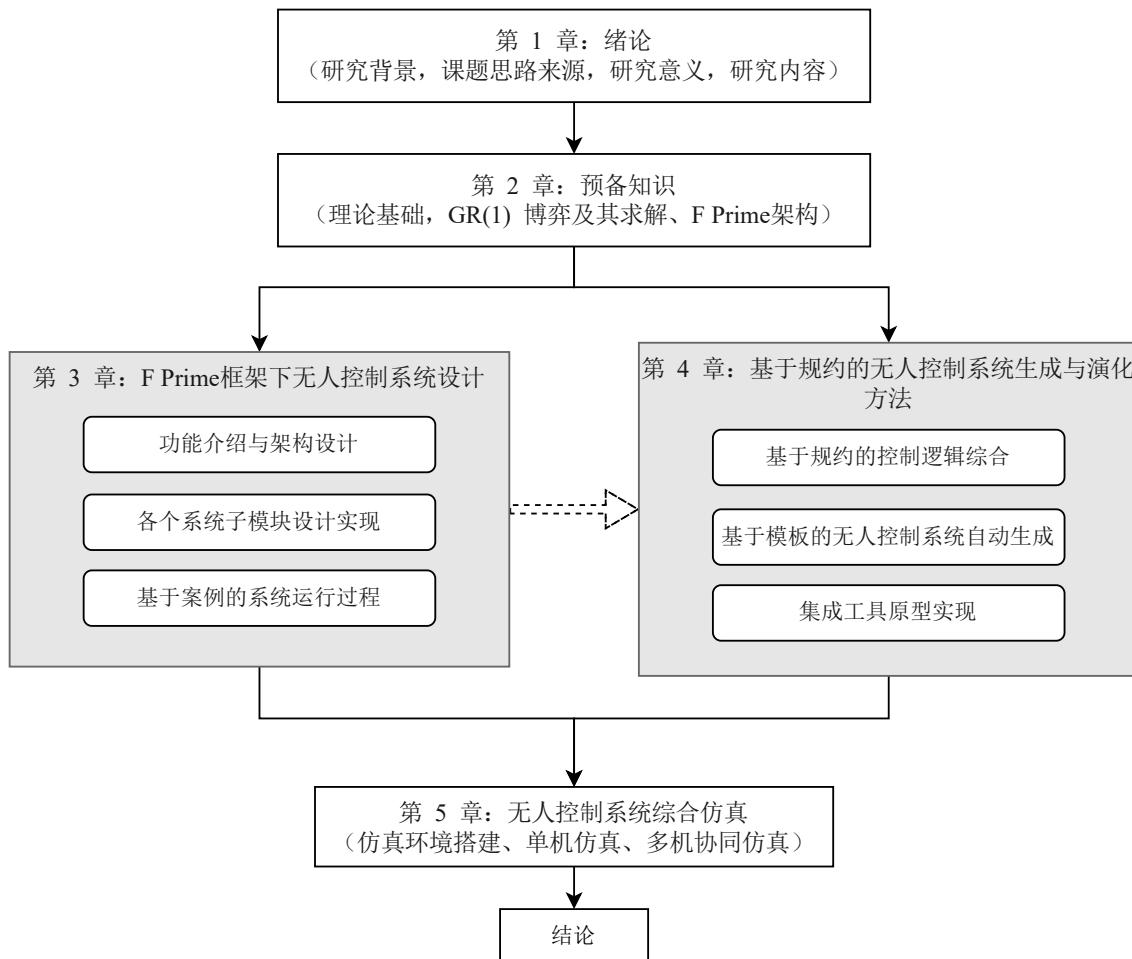


图 1-3 论文组织结构图

第 2 章 预备知识

在第一章中，本文已经对课题来源以及相关研究背景进行了总览性介绍，本章将会对课题在研究实现中使用到的重要算法以及理论知识进行介绍。具体内容包括用于系统逻辑控制生成的 GR(1) 博弈求解知识以及无人控制系统所基于的 F Prime 框架主要功能特性介绍。通过对这些实验开展中用到的相关概念和理论方法进行总结，以期让全文逻辑结构更加完整清晰。

2.1 Generalized Reactive(1) 博弈

在反应式综合领域，线性时序逻辑语言（Linear Temporal Logic, LTL）经常用于系统任务规约描述。相比于一阶逻辑和命题逻辑，LTL 引入了时序算子来反映系统的动态变化性质。这一特性使得 LTL 能够对反应式系统的逻辑行为进行较为完备的定义和描述。LTL 的语法定义：

定义 2-1 (LTL 语法): 给定原子命题集合 AP ，以及一个 LTL 公式 φ ，那么 LTL 的语法可递归定义如下：

$$\varphi ::= p \in AP \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \mid \Theta\varphi \mid \varphi_1 \mathcal{S} \varphi_2$$

在该语法中包含了基本将来时序算子 (\bigcirc, \mathcal{U}) 以及基本过去时序算子 (Θ, \mathcal{S})，它们的语义是定义在一个无穷序列上的。同时，LTL 给出了其他派生时序算子，以下对主要的将来时序算子语义进行解释（过去时序算子与之时序相反）：

- $\pi, i \models \bigcirc\varphi$ iff $\pi, i+1 \models \varphi$
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists k \geq i$ 使得 $\pi, k \models \varphi_2$ 且 $\forall j, i \leq j < k$ 有 $\pi, j \models \varphi_1$
- $\pi, i \models \Diamond\varphi$ iff $\exists k \geq i$ 使得 $\pi, k \models \varphi$
- $\pi, i \models \Box\varphi$ iff $\forall k \geq i$ 都有 $\pi, k \models \varphi$
- $\pi, i \models \varphi_1 \mathcal{W} \varphi_2$ iff $\exists k \geq i$ 使得 $\pi, k \models \varphi_2$ 且 $\forall j, i \leq j < k$ 都有 $\pi, j \models \varphi_1$ ，
- $\pi, i \models \Box\Diamond\varphi$ iff $\forall k \geq i$ 都有 $\exists j \geq k$ 使得 $\pi, j \models \varphi$

为了减少特定无人系统场景下的规约综合复杂度，Piterman 等人提出了定义在 LTL 子集上的 GR(1) 规约进行的反应式综合方法^[2]。本节将重点介绍 GR(1) 规约以及通过求解 GR(1) 博弈算法对规约可综合性进行判定、提取策略的基本原理。

2.1.1 博弈结构与获胜策略

为了实现基于规约的反应式系统综合，Piterman 等人提出了一个系统 - 环境双人博弈结构，从而实现了综合策略求解问题的形式化。系统的目标是无论环境产生什么变化，系统都要在满足规约的条件下进行。该博弈结构是一个八元组：

定义 2-2 (博弈结构): 博弈结构定义为 $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \theta_e, \theta_s, \rho_e, \rho_s, \varphi \rangle$ ：

- $\mathcal{V} = \{v_1, \dots, v_n\}$: 布尔命题集，命题 $s = s_x \cup s_y$ (状态) 是对 \mathcal{V} 的一个指派，即 $s \in \Sigma_{\mathcal{V}}$ 。
- $\mathcal{X} \subseteq \mathcal{V}$: 由环境控制的输入变量集合， $s_x \in \Sigma_{\mathcal{X}}$ 是状态 s 可能输入的指派。
- $\mathcal{Y} = \mathcal{V} \setminus \mathcal{X}$: 由系统控制的输出变量集合， $s_y \in \Sigma_{\mathcal{Y}}$ 是状态 s 可能输出的指派。
- θ_e : 定义在 \mathcal{X} 上的断言，表示初始状态环境的定义。
- θ_s : 定义在 \mathcal{V} 上的断言，表示初始状态系统的定义。
- $\rho_e(\mathcal{V}, \mathcal{X}')$: 定义环境的迁移关系，是在 $\mathcal{V} \cup \mathcal{X}'$ 上的断言。对于状态 s ，给定下一步可能的输入 $s_x \in \Sigma_{\mathcal{X}}$ ，如果 $(s, s_x) \models \rho_e$ ，则称 s_x 为 s 的合法输入。
- $\rho_s(\mathcal{V}, \mathcal{X}', \mathcal{Y}')$: 定义系统的迁移关系，是在 $\mathcal{V} \cup \mathcal{X}' \cup \mathcal{Y}'$ 上的断言。对于状态 s 和下一步可能的输入 $s_x \in \Sigma_{\mathcal{X}}$ ，给定下一步输出 $s_y \in \Sigma_{\mathcal{Y}}$ ，如果 $(s, s_x, s_y) \models \rho_s$ ，则称 s_y 为 s 在接收输入 s_x 后的合法输出。
- φ : 获胜条件，是一个 LTL。

这是一个面向反应式系统综合的通用博弈结构，以下对其博弈运行过程以及系统、环境双方获胜条件、系统获胜策略进行定义：

定义 2-3 (博弈运行过程): 对于 $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \theta_e, \theta_s, \rho_e, \rho_s, \varphi \rangle$ ， G 的一次博弈过程是一个极大状态序列： $\sigma = s_0, s_1, s_2 \dots$ ，该序列需要满足以下条件：

- s_0 是初始状态，包括了对环境和系统的初始设置；
- 对所有的 $j \geq 0$ ，都有 s_{j+1} 是 s_j 的后继。对于状态 s_j ，环境选择输入 s_x ，满足 $(s_j, s_x) \models \rho_e$ ，接着系统选择输出 s_y ，满足 $(s_j, s_x, s_y) \models \rho_s$ ，那么 $s_{j+1} = s_x + s_y$ 就称为 s_j 的后继。

定义 2-4 (系统及环境获胜条件): 对于 $G = \langle \mathcal{V}, \mathcal{X}, \mathcal{Y}, \theta_e, \theta_s, \rho_e, \rho_s, \varphi \rangle$ 上的一次博弈运行过程 $\sigma = s_0, s_1, s_2 \dots$ ，如果 σ 满足下列某一个条件，则称 σ 是系统获胜的：

- σ 是有限的，且对于 σ 的终止状态 s_n ，不存在输入变量的指派 $s_x \in \Sigma_{\mathcal{X}}$ ，使得 $(s_n, s_x) \models \rho_e$ ；

-
- σ 是无限的，且 $\sigma \models \varphi$ 。
- 否则，称 σ 是环境获胜的。

定义 2-5 (系统获胜策略): 系统获胜策略定义为一个偏序函数 $f : \Sigma_{\mathcal{V}} \times \Sigma_{\mathcal{X}} \mapsto \Sigma_{\mathcal{Y}}$ ，该函数用于为当前状态搜索后继状态。如果所有符合策略 f 的运行 ρ 都是系统获胜的，那么称 f 为系统获胜策略。

一般而言，对于通用博弈结构 $G = < \mathcal{V}, \mathcal{X}, \mathcal{Y}, \theta_e, \theta_s, \rho_e, \rho_s, \varphi >$ ，可以构造出相应的 LTL 公式作为规约：

$$\varphi_G = (\theta_e \rightarrow \theta_s) \wedge (\theta_e \rightarrow \square((\exists \rho_e) \rightarrow \rho_s)) \wedge (\theta_e \wedge \square \rho_e \rightarrow \varphi)$$

满足以下定理：

定理 2-1: φ_G 是可综合的当且仅当 G 是系统获胜的^[26]。

2.1.2 GR(1) 博弈与求解

为了降低综合过程的复杂度并且保留 LTL 对反应式系统的描述能力，可以使用 GR(1) 公式定义博弈结构的获胜条件。这样可以将基于命题的 μ 演算^[27] 方法应用于 GR(1) 博弈系统获胜策略获取，实现 GR(1) 博弈结构上的高效计算。

定义 2-6 (GR(1) 规约): GR(1) 规约定义在给定环境变量集合 \mathcal{X} 和系统变量集合 \mathcal{Y} 之下，有以下六种表现形式：

- φ_i^e : 约束环境初始值，是 \mathcal{X} 上的一个断言。
- φ_t^e : 约束环境状态迁移，形如 $\wedge \square B_i$ ， B_i 是 $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X}$ 上的公式。
- φ_g^e : 约束环境目标，形如 $\wedge \square \lozenge B_i$ ， B_i 是 \mathcal{X} 上的公式。
- φ_i^s : 约束系统初始值，是 \mathcal{Y} 上的一个断言。
- φ_t^s : 约束系统状态迁移，形如 $\wedge \square B_i$ ， B_i 是 $\mathcal{X} \cup \mathcal{Y} \cup \bigcirc \mathcal{X} \cup \bigcirc \mathcal{Y}$ 上的公式。
- φ_g^s : 约束系统目标，形如 $\wedge \square \lozenge B_i$ ， B_i 是 \mathcal{Y} 上的公式。

对 GR(1) 博弈的系统获胜策略求解问题可以等价为以下公式的可综合性问题：

$$(\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \rightarrow (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s) \quad \text{公式 (2-1)}$$

通过定义 GR(1) 博弈上的 μ 演算，等价于求解以下 μ 演算公式在博弈结构 G 上的语义^[28]：

$$\begin{aligned} \varphi_{gr} &= \nu Z \left(\bigwedge_{j=1}^n \mu Y \left(\bigvee_{i=1}^m \nu X (J_j^s \wedge \bigcirc Z \vee \bigcirc Y \vee \neg J_i^e \wedge \bigcirc X) \right) \right) \\ &= \nu \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ \vdots \\ Z_n \end{bmatrix} \begin{bmatrix} \mu Y (\bigvee_{i=1}^m \nu X (J_1^s \wedge \bigcirc Z_2 \vee \bigcirc Y \vee \neg J_i^e \wedge \bigcirc X)) \\ \mu Y (\bigvee_{i=1}^m \nu X (J_2^s \wedge \bigcirc Z_3 \vee \bigcirc Y \vee \neg J_i^e \wedge \bigcirc X)) \\ \vdots \\ \vdots \\ \mu Y (\bigvee_{i=1}^m \nu X (J_n^s \wedge \bigcirc Z_1 \vee \bigcirc Y \vee \neg J_i^e \wedge \bigcirc X)) \end{bmatrix} \end{aligned} \quad \text{公式 (2-2)}$$

2.2 组件驱动的分布式控制框架 F Prime

F Prime (F') 是一个组件驱动的软件框架，专注于快速和可验证地开发、部署以轻型飞行器为代表的小型嵌入式系统。该项目由 NASA 的喷气推进实验室 (Jet Propulsion Laboratory) 创建并持续维护，并且已经应用于 ASTERIA、CubeSat 飞行器和火星探测无人车、无人机等实际的 FSW 任务。本章将对 F Prime 架构中的基本概念和理论背景进行解释，并且介绍其完整的工具平台和优秀功能特性。

2.2.1 基本概念与特点介绍

所有的 F Prime 应用都是运行在 F Prime 软件架构 (F Prime architecture) 之下的，遵循该架构的开发规范。F Prime 架构主要包括以下三个概念：组件 (components)，端口 (ports)，拓扑结构 (topology)。

(1) 组件：每一个 F Prime 系统都是由组件组成的。通过引入组件可以将 FSW 任务划分至粒度更小的单元，这有利于各个单元在不同任务中的重用。按照组件定义方式和运行机制可以将组件划分为三种类型：主动型 (Active)，被动型 (Passive) 和周期型 (Queued)：

表 2-1 F Prime 架构中组件的类型划分

组件类型	具有关联线程	具有输入队列
Active	✓	✓
Passive	✗	✗
Queued	✗	✓

在 F Prime 架构中，组件可以理解为面向对象语言中的一个类：定义了一

组数据以及封装了一系列对该组数据的操作。具体而言，组件可以包括命令（Commands）、事件（Events）、通道数据（Channels）和参数（Parameters），这些数据以及相应的操作共同决定了组件在 F Prime 系统中的数据模式。

表 2-2 定义在组件上的输入端口类型

数据类型	功能描述
命令	命令是指示组件执行操作的上行数据项。
事件	代表系统中的单个事件发生的一种下行数据项。
通道	通道，也称为遥测项，是可用于读取和下行的数据值。
参数	参数是组件中可通过命令更新的类型化常量值。

(2) 端口：组件和组件之间的交互是通过唤醒（Invocation）实现的，在 F Prime 架构之下，唤醒行为分为同步（sync）和异步（async）两种类型。同步类型下，在调用者当前线程直接唤醒目标组件；异步类型下，会将唤醒动作添加到目标组件的输入队列中等待异步执行。在 F Prime 架构中，唤醒行为定义在端口（Ports）上，即组件 c_1 对组件 c_2 的唤醒实现方式是：先在 c_1 中向相应端口 p 输出信息，根据输入端口的类型， c_2 会接收来自 p 的输入信息并进行响应。单个组件可以包括一组输入端口和输出端口，其接受唤醒的处理函数 handler 运行方式与输入端口的类型相应关系见下表2-3。

表 2-3 定义在组件上的输入端口类型

输入端口类型	适用组件类型	描述
Sync	active,passive,queued	在调用者线程中同步唤醒
Async	active,queued	在组件输入队列中异步唤醒
Guarded	active,passive,queued	并发访问可变数据，需设置互斥锁

(3) 拓扑结构和部署：一个 F Prime 系统可以用其对应的拓扑结构进行表示，它对系统中的组件组成、端口组成以及连接关系进行定义。F Prime 的可执行程序被称为一个部署，部署中包含了编译缓存（可执行二进制文件）和许多 F Prime 系统（项目），这些系统基于一组组件和一组端口实现特定的完整任务，其中组件定义控制行为，端口定义通信机制。

为了实现高效安全的 FSW 开发任务，F Prime 还提供了为特定体系结构（Linux 和树莓派系统）构建的一组编译链工具、基于类型描述的系统框架生成器、单元测试生成器和用于控制展示系统运行状态的地面数据系统（GDS）。

2.2.2 基于模型的代码框架生成

F Prime 可以基于用户对各个组件、端口、自定义序列化数据类型的描述自动生成 c++ 代码框架，通过补充组件中的各个处理函数实现控制逻辑。为了规范化该描述书写，JPL 实验室的 FSW 课题研究组提出了一个建模语言 FPP(F Prime Prime)，开发者可以通过指定 fpp 描述对系统结构以及组件组成进行定义。

F Prime 提供了一个 Python 编写的基于模型的代码框架生成器（AutoCoder），每一个组件在 c++ 框架下会转化为一个类，而其中定义的事件、命令、参数、通道、端口，都会生成对应的处理函数。开发者只需要在自动生成的代码框架下对输入函数（输入命令和输入端口）进行自定义即可完成对整个系统的实现。在系统实际运行时，每一个组件类将会被实例化进而实现各个组件对控制的接收以及交互操作。

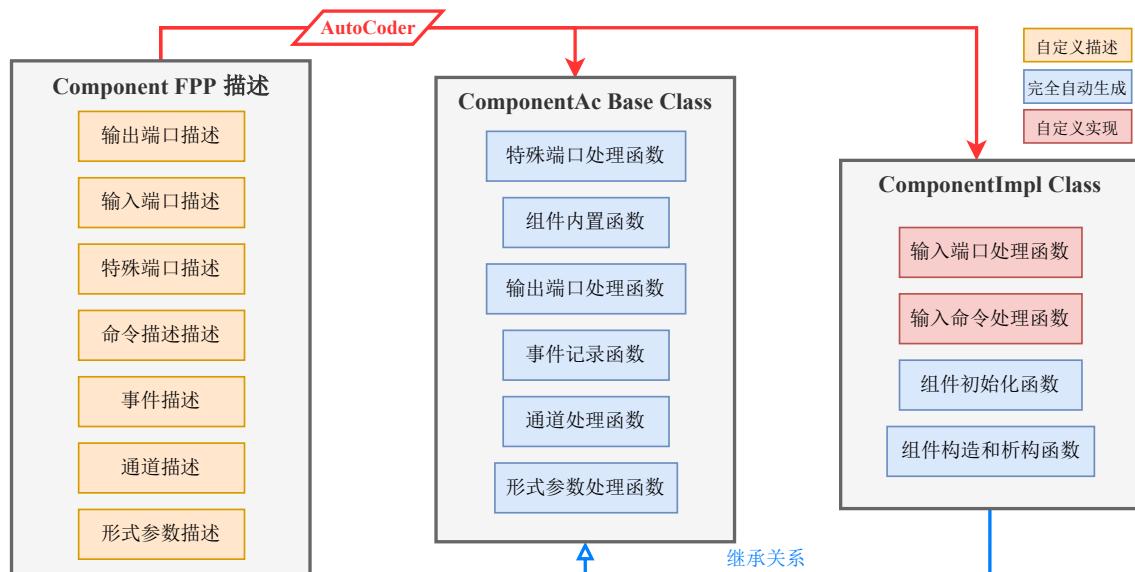


图 2-1 F Prime 框架中组件自动生成与开发过程

同时，F Prime 将单元测试框架生成也集成到了编译链中，可以快速地生成基于 GTest 和 STest 框架的单元测试模板，开发者通过自定义测试集合或者使用 STest 测试库对组件进行多样且丰富的单元测试。这对于无人系统开发的安全性和质量保证具有重要意义。

总结而言，组件驱动的 F Prime 开发框架集成了标准组件库和一系列规范工具，具有高效开发性质、重用性、可移植性、高性能性、适应性、可分析性等特性。

2.3 本章小结

本章主要介绍了论文所需的相关理论知识和方法，在上一章绪论中，已经提出本文主要工作就是将反应式综合和 F Prime 组件驱动框架结合，实现无人控制系统的完整开发以及部署。本章内容就是对以上两个工具（方法）分别进行介绍。

综合方法方面：本章重点对反应式综合问题的博弈结构、获胜策略以及 GR(1) 博弈和求解算法通过形式化定义以及详细说明进行了介绍，基于 GR(1) 博弈的 μ 演算能够实现 GR(1) 规约的高效求解综合。F Prime 相关知识方面：本章对 F Prime 框架的结构、相关概念以及其基于模型的代码框架生成方法进行了概括说明，并且对 F Prime 在本论文所需设计的系统实现中的重要意义进行了分析。

在接下来的内容中，本文将围绕主体工作展开，进一步介绍这些理论和方法在无人控制系统自动生成中的具体应用，并且针对具体问题提供方法介绍和案例解释。

第3章 面向程序自动生成的无人控制系统架构设计

现有的反应式综合方法是通过求解规约得到系统获胜策略，如果策略无法生成就称该规约不可综合。本课题要求实现从高层次的规约 (high-level specification) 中自动生成可直接运行在 F Prime 框架下的无人控制系统，需要设计一个基于 F Prime 框架的无人控制系统架构用于策略的解析以及运行。该无人控制系统以综合策略作为输入，目标就是在运行时能够实时接受环境的变化并且按照策略做出正确的响应。

作为反应式综合和 F Prime 框架的接口，该系统架构直接决定了整个反应式系统生成的正确性和可行性。一方面，架构设计需要标准化，从而有利于后续基于该架构模型的无人控制系统自动生成方法的研究。另一方面，架构设计需要能够和 F Prime 框架中的组件驱动机制以及命令、事件机制高度耦合，充分利用 F Prime 框架在无人系统开发上的特性。

本章将会围绕构建该无人控制系统架构展开，基于前文提到的两个系统架构设计要求，对系统的分层结构设计以及各模块实现细节和交互方式进行详细说明，整个无人控制系统自顶向下的框架如图3-1所示。

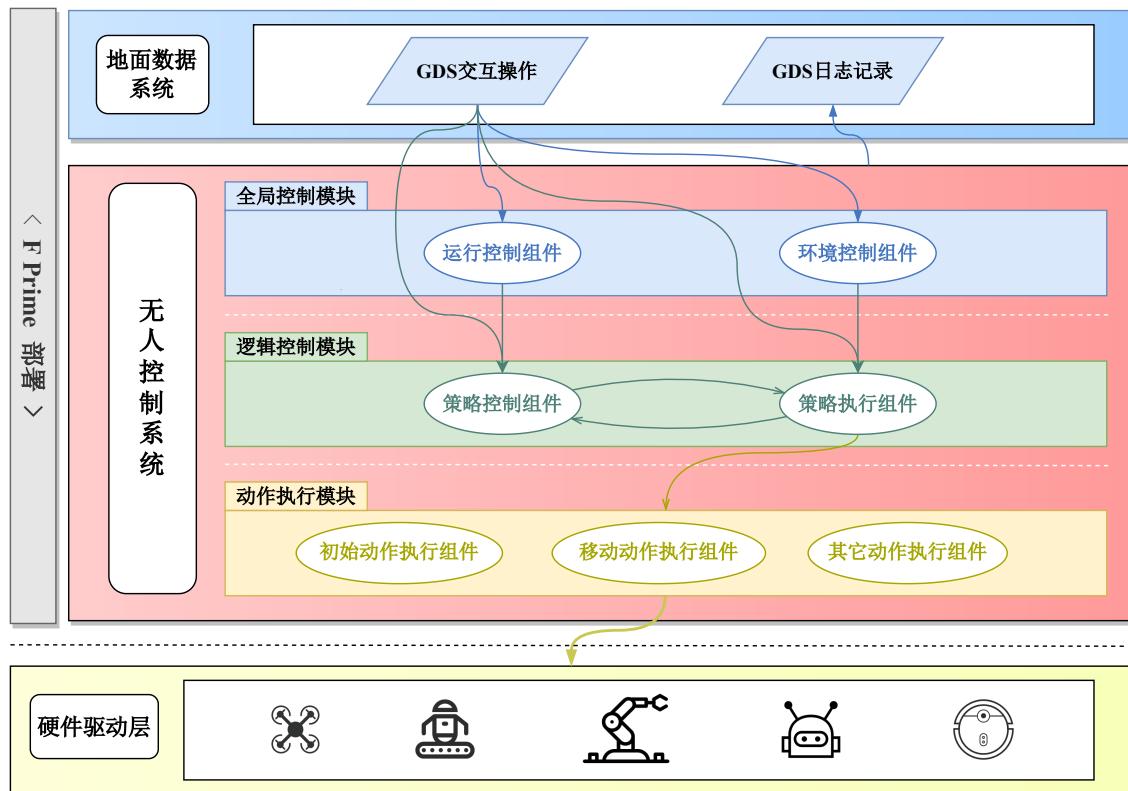


图 3-1 基于 F Prime 框架设计实现的无人控制系统架构完整结构

3.1 分层结构设计和功能介绍

为了实现上述无人控制系统架构，根据具体负责的任务功能不同，将无人控制系统自顶向下划分为三个模块：全局控制模块、逻辑控制模块以及动作执行模块。

(1) 全局控制模块：用于无人控制系统中所有机器人单元的全局控制，包括运行控制组件以及环境控制组件。运行控制组件负责对系统中所有机器人的运行状态进行控制，通过向运行控制组件发送命令从而实现对系统内所有机器人的启动、停止与重置操作。为了实现在具体仿真时对机器人所处环境（感知信息）的改变，同时设计了环境控制组件对全局仿真环境进行配置，该环境感知信息的变化会被所有系统内的机器人接收。

(2) 逻辑控制模块：策略作为无人控制系统的输入，每一个策略对应一个机器人的任务描述，无人控制系统支持多机协同。在系统运行中，策略定义的是单机器人的逻辑行为，其运行由策略控制组件（Controller）和策略执行组件（Executor）共同完成。策略控制组件负责控制单机器人运行状态，在运行策略时会唤醒策略执行组件。策略执行组件能够完成对输入策略自动机的解析，并且在接受唤醒时根据环境感知信息指导状态迁移以及相关动作的执行。逻辑控制模块的持续运行是通过策略控制组件和策略执行组件之间不断相互唤醒实现的。

(3) 动作执行模块：动作执行模块包括一组动作执行组件，这些组件和策略中定义的动作命题相对应，是动作命题的实现也是机器人具体动作的抽象，将无人控制系统与仿真机器人通过行为实现连接。在系统运行过程中，动作执行模块由策略执行组件直接调用，可以对启动动作、判断动作是否完成、结束动作、判断动作是否结束四种相关行为进行定义。基于组件对动作进行定义，一方面增强了动作 API 的可移植性和重用性，另一方面实现了上层控制系统和底层交互行为之间的解耦合，有利于无人控制系统的去平台化以及自定义程度的提升。

在实际运行一个 F Prime 部署中的无人控制系统时，系统提供了对全局运行状态和环境信息修改的命令，以及针对某一个机器人（对应其策略控制器和执行器）的单一运行状态和环境感知信息修改命令。一旦系统被设置启动，将会在网页 GUI 界面显示命令接收情况，当前运行状态情况以及动作执行情况，系统的运行过程将被完整清晰地记录在 F Prime 提供的地面数据系统（Ground Data System, GDS）中。

本文设计的无人控制系统具有高度的可扩展性和规范性，其中多机协同任务

定义以及单机内部组件交互过程都是基于组件之间的端口实现的。通过定义新的端口和组件可以简单高效地实现系统功能扩充，比如可以扩展实现环境信息采集组件代替命令对环境感知信息进行修改，环境采集组件和策略执行组件之间通过端口连接，一旦有环境信息的改变就会唤醒策略执行组件更新当前感知信息从而指导状态迁移。

3.2 F Prime 系统架构的形式化

该无人控制系统的实现基于 F Prime 架构，为了更方便地进行该架构下开发并且对架构整体有更完备清晰的认识，本文对 F Prime 下的系统架构进行了形式化定义。首先对于 F Prime 中的基本行为单元——组件给出以下形式化定义：

定义 3-1 (F Prime 组件): 单个组件的标准数据模式结构由以下四元组给出
 $c = \langle Commands, Events, Channels, Params \rangle$:

- *Commands*: 组件接收的命令集合，组件会对每一个命令进行响应。命令由操作码、助记符和命令的参数列表组成。
- *Events*: 事件集合，代表了组件运行历史记录。事件由名称、格式字符串和描述所发生事件的一组参数组成。
- *Channels*: 通道数据集合，通道数据用于表示当前组件状态。通道由类型、格式说明符和值组成。
- *Params*: 形式参数集合，组件运行时形式参数会绑定到该组件上具体的命令和端口作为参数传入。F Prime 提供了特殊端口用于管理形式参数。

其次，对于基于一组组件和端口完成指定任务的 F Prime 系统（可以直接理解为 F Prime 架构中的一个拓扑结构），可以使用一个三元组对其组成和逻辑行为进行定义：

定义 3-2 (F Prime 系统): 一个 F Prime 系统定义为一个三元组 $S = \langle \mathcal{C}, \mathcal{P}, f \rangle$:

- $\mathcal{C} = \{c_1, c_2, c_3 \dots\}$: 组件集合，是进行程序行为的位置，分为主动型（接受同步和异步唤醒），周期型（接受同步和周期性异步唤醒）和被动型（只接受同步唤醒）三种类型。
- $\mathcal{P} = \{p_1, p_2, p_3 \dots\}$: 端口集合，端口定义了通信数据类型，根据端口唤醒类型分为同步、异步和保护型端口。
- $f : \Sigma_{\mathcal{C}} \times \Sigma_{\mathcal{C}} \mapsto \mathcal{P}$: 连接关系映射，函数输入按顺序分别是唤醒者集合和被唤醒者集合，输出是单个端口。定义了组件之间的唤醒关系，且单个端口支持具有多个唤醒者和多个被唤醒者。

本章接下来的部分将会基于分层结构设计，对 F Prime 框架下无人控制系统的各个模块设计实现方法进行详细阐述。

3.3 主体逻辑控制模块设计实现

主体逻辑控制模块是本系统中的核心功能实现模块，既执行了策略，也定义了 F Prime 无人控制系统的全局拓扑结构。对于每一个系统中的机器人策略，都由一个相应的逻辑控制模块负责，该主体模块的组件和端口连接关系如图3-2所示。

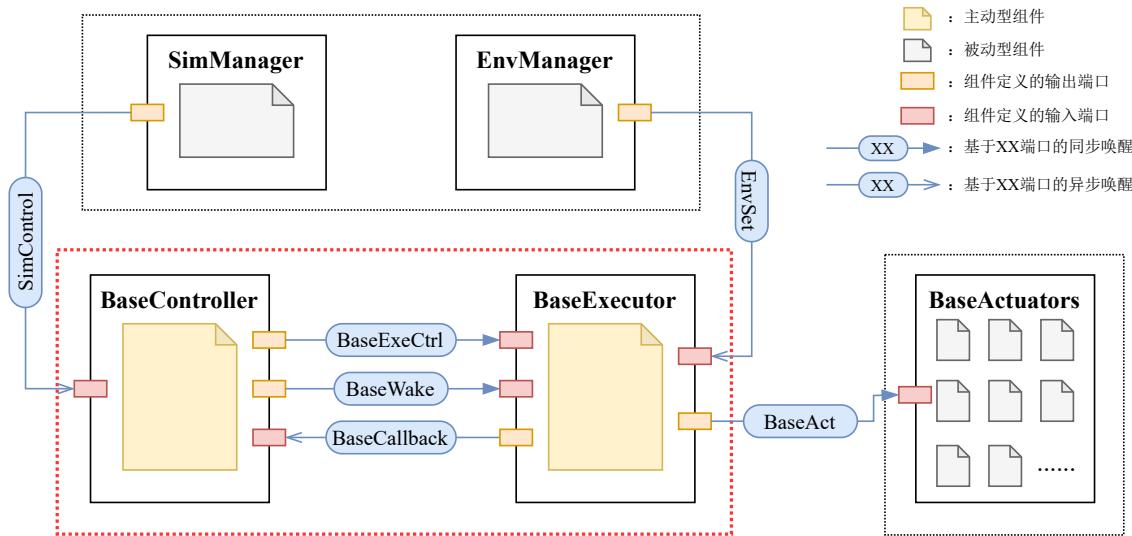


图 3-2 主体逻辑控制模块各组件连接关系

本节将会基于该拓扑结构图，详细介绍主体逻辑控制模块相关端口、组件的实现细节。

3.3.1 策略信息解析库

逻辑控制模块需要将综合生成的策略作为输入进行解析运行，具体而言，该策略可以用一个确定有限自动机（DFA）表示。本论文采用的综合策略和 LTLMoP^[3] 中生成的自动机形式一样，现给出该策略自动机的形式化定义：

定义 3-3 (逻辑控制策略): 一个综合得到的逻辑控制策略是一个四元组：

$$M = \langle \mathcal{S}, \mathcal{X}, \delta, s_0 \rangle:$$

- \mathcal{S} : 有穷状态集，单个状态是由环境感知信息和输出信息共同组成的。
- \mathcal{X} : 有穷的输入命题集合，代表下一状态的环境感知信息。
- δ 定义在 $\mathcal{S} \times \mathcal{X} \rightarrow \mathcal{S}$ 上的转移函数。
- $s_0 \in \mathcal{S}$ ，是初始状态。

其中单个状态由状态标识符（state id）唯一标识，内容还包括目标等级（goal id），对命题集的一组指派和后继状态集。命题集包括当前状态下的感知命题集（Sensors），动作命题集（Actions），自定义命题集（Customs）和位置编码信息。策略的运行就是从初始状态开始，根据当前感知命题信息运行自动机，从后继状态集中选择相应的状态进行状态迁移。图3-3是一个闹钟机器人的策略自动机示例。

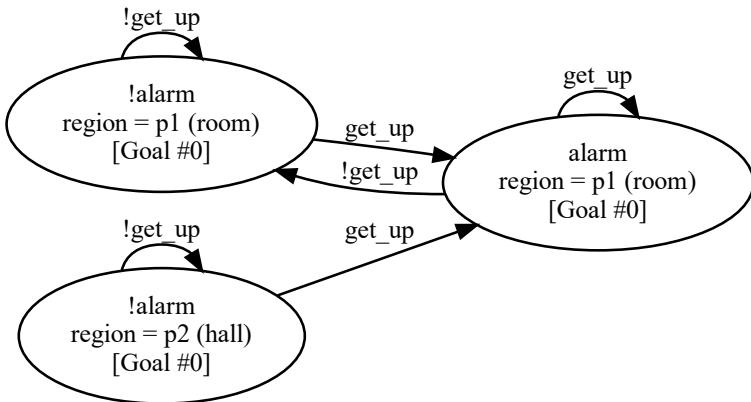


图 3-3 策略自动机状态迁移示例

为了实现逻辑控制模块对该策略自动机的解析，本论文设计实现了一个策略解析库（动态链接库）并且将此解析库通过编译链接给了策略执行组件（策略执行组件在逻辑控制模块中负责解析并且运行策略）。该策略解析库封装实现了对策略信息进行提取并且运行的 `Strategy` 类，以这种方式实现了策略执行组件基于策略的逻辑处理。`Strategy` 类具有良好的定义，并且可以实现以下功能：

- 对策略自动机文件和相关配置文件的正确性检查。
- 运行过程信息输出，包括状态迁移结果以及当前状态信息的即时显示。
- 区域信息解析，指基于地图文件内容解析区域编码建立区域名称映射。
- 策略自动机单步运行，包括后继状态确定、迁移动作获取以及状态更新。

`Strategy` 类在实现中使用一个状态列表记录自动机的所有状态，在策略自动机的运行过程，用状态标识符标识当前运行状态。该运行过程主要分为三个内容：基于当前环境感知信息确定后继状态（算法3-1：`getNextStateID`）、状态迁移动作获取（算法3-2：`runSingleTransition`）以及系统运行状态更新（通过设置当前运行状态标识符并且更新上下文实现）。

为了实现对策略的解析运行，策略执行组件编译时会链接该策略解析动态库，并且通过维护一个全局的 `Strstegy` 类实例完成基于策略的逻辑行为。

Algorithm 3-1: Strategy::getNextStateID // 确定后继状态

Input: 存储环境感知信息的字典 *nextSensorMap*

Output: 后继状态标识符 *next_id*

```

1 foreach suc  $\in$  successors do
2     sensors  $\leftarrow$  getSensorList(next_id);
3     match  $\leftarrow$  0;
4     foreach s  $\in$  sensors do
5         if nextSensorMap[s.name]  $=$  s.value then
6             match  $\leftarrow$  match + 1;
7     if match  $=$  sensorsNum then
8         return suc;
9 return -1;
```

Algorithm 3-2: Strategy::runSingleTransition // 获取动作集合

Input: 当前状态标识符 *cur_id* 和后继状态标识符 *next_id*

Output: 状态迁移所需动作序列 *actions*

```

1 i  $\leftarrow$  0;
2 actions  $\leftarrow$  {};
3 curActions  $\leftarrow$  getActionList(cur_id);
4 nextActions  $\leftarrow$  getActionList(next_id);
5 while i  $<$  actionsNum do
6     if curActions[i]  $\neq$  nextActions[i] then
7         if curActions[i] = 1 then
8             actions.push(Actions[i]);
9         else
10            actions.push(endActions[i]);
11 return actions;
```

3.3.2 基于组件交互的逻辑控制实现

根据拓扑结构图3-2，策略控制组件（以下简称单步控制器）和策略执行组件（以下简称单步执行器）之间通过以下端口进行连接（Base 表示端口模型名）：

- **BaseExeCtrl:** 单步控制器通过此端口向单步执行器同步发送启动、暂停、重置信号，从而实现对本机器人策略运行过程的控制。

- **BaseWake:** 单步控制器通过该端口同步唤醒单步执行器执行状态检查并且根据当前环境信息（由单步执行器维护）进行状态迁移（运行自动机）。

- **BaseCallback:** 在单步执行器完成状态检查以及状态迁移后，会通过此端口以异步的方式唤醒单步控制器再次调用 BaseWake 输出端口，实现周期运行。

BaseExeCtrl 端口具有一个操作符参数（control id），对该参数有以下规定：0 表示启动操作；1 表示暂停操作；2 表示重置操作。对该端口的调用是通过向单步控制器发送命令实现的，当单步控制器接收到不同的命令时，就会以相应的参数调用该端口唤醒单步执行器。时序图3-4对控制器和执行器基于该端口的交互过程进行了详细说明。

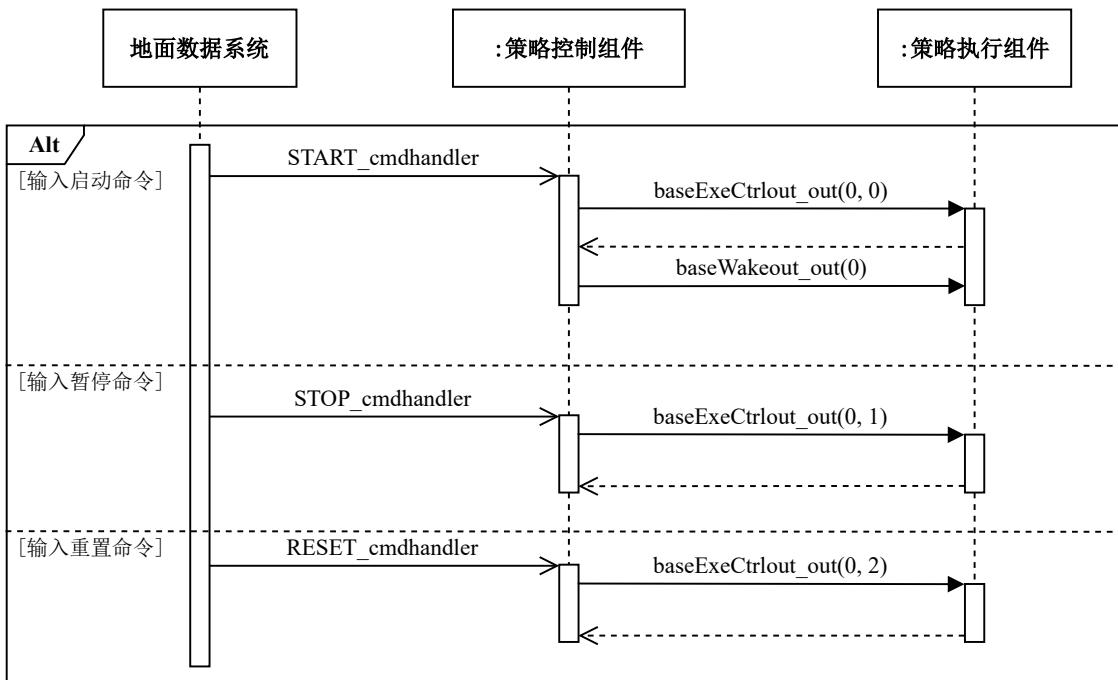


图 3-4 基于 BaseExeCtrl 端口的组件交互时序图

在策略运行的一个周期内，单步控制器会唤醒单步执行器。周期默认设置为 0.5s，每一个周期是运行在一个独立线程上的（异步）。由于组件在运行时只有一个相应线程，对各个端口和命令的处理会加载至其输入队列中依次进行。因此严格来说，对于外部命令和环境改变系统会有一个周期的滞后性。理论上，周期设置越小，系统即时性越强。

在单步执行器被唤醒后，先会对环境改变进行处理（环境变化会直接导致状态迁移），然后会检查系统运行状态是否满足状态迁移条件（具体体现为检查各个

动作是否执行完毕), 最终进行状态更新以及后续动作的执行。同时执行器对于该唤醒端口的处理函数会返回停止信号从而实现暂停控制。其具体流程见算法3-3:

Algorithm 3-3: BaseExecutor::baseWakein_handler

Input: 已经完成初始化的控制策略 *Strategy*

Output: *true* 或 *false* (表示是否中断运行)

Result: 完成单个周期内的状态迁移条件检查和状态迁移动作执行

```

1 if stopFlag then
2   return true;
3 if envChange then
4   backSensorMap  $\leftarrow$  Strategy.curSensorMap;
5   if 待执行动作非空且全部完成 then
6     Strategy.refreshCurState();
7     保存并输出当前状态信息;
8   Actions  $\leftarrow$  Strategy.runSingleTransition(backSensorMap);
9   foreach act  $\in$  Actions do
10    调用动作执行器执行 act;
11 if 待执行动作未全部完成 then
12   foreach act  $\in$  Actions do
13    调用动作执行器检查 act 是否完成;
14 if 待执行动作全部完成 then
15   Strategy.refreshCurState();
16   if Strategy.nextStateID  $\neq$  Strategy.curStateID then
17     保存并输出当前状态信息;
18     Actions  $\leftarrow$  Strategy.runSingleTransition(curSensorMap);
19     foreach act  $\in$  Actions do
20      调用动作执行器执行 act;
21 else
22   if envChange then
23     保存并输出当前状态信息;
24     envChange  $\leftarrow$  false;
25   清空待执行动作;
26 return false;

```

3.4 其它模块功能设计实现

本节将会对全局控制模块和动作执行模块的设计实现进行解释说明，全局控制模块是 GDS 对所有机器人设置运行状态的中间桥梁，而动作执行模块将无人系统中的逻辑控制和实际场景下的机器人硬件平台相连接。

3.4.1 全局运行控制与环境设置

全局控制模块作用于所有机器人单元，通过设计实现全局运行控制组件（SimManager）、端口（SimControl）和全局环境控制组件（EnvManager）、端口（EnvSet），完成所有机器人的统一规划。向全局控制模块中的组件发送命令将会唤醒系统中所有的机器人，其中运行控制组件直接唤醒所有策略控制组件，环境控制组件则会直接唤醒所有的策略执行组件，这两种唤醒都是异步的。

全局运行控制组件接收地面数据系统 GDS 发送的启动、暂停和重置命令之后唤醒策略控制组件，具体流程与图3-4所示基于不同命令的处理方式一致。

在默认仿真运行的方式中，全局环境感知信息是通过用户发送命令进行修改的。在全局环境控制组件中，本系统基于感知命题集定义了一个 F Prime 枚举类型 EnvSensor，该类型用于环境设置命令的感知命题指定，可以使得用户在使用地面数据系统 GDS 发送环境设置命令时更加简单高效。策略执行器在运行中会维护一个实时环境感知信息表，环境变化将会直接修改表中的相应表项，并且在下一次运行周期中完成状态迁移对该环境变化做出即时的动作反应。

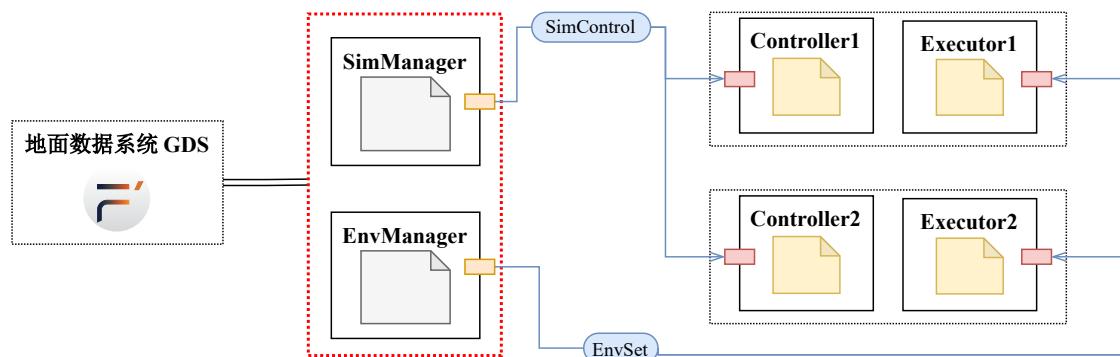


图 3-5 多机器人的全局控制结构

3.4.2 软件定义动作执行模块

F Prime 基于组件的框架理念设计之初就是为了满足 FSW 嵌入式开发任务之间的重用性，而这一点也在组件的实现上有很好的体现。F Prime 组件只要重新设

计端口就能实现跨平台、跨任务的重用，具有很好的可移植性、可复用性和可扩展性。

基于本论文提出的“软件定义无人系统”的思想，在实现时可以为每一个策略提取的动作设计一个动作执行组件（往往是对基本动作的实现，不包括逻辑控制），针对不同的任务可以自定义不同的动作执行组件。一方面，这实现了上层逻辑控制和底层硬件动作相分离。动作执行组件是对硬件层行为控制的抽象，通过定义端口对动作执行组件唤醒，在动作执行组件中实现行为从而将上层逻辑控制和底层硬件环境（仿真环境、实际场景）连接。

另一方面，这能够大幅度提高无人系统开发的效率。对于某些基于相同或者相似机器人模型的任务描述，其基本动作行为是基本一致的，因此之前的任务开发所设计的动作执行组件可以重新应用于新任务中。比如机器人的移动动作、拍照动作等等。

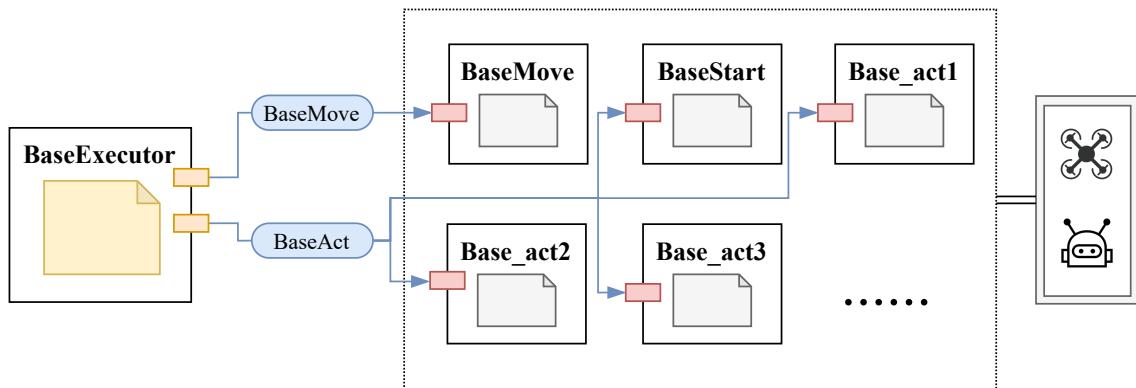


图 3-6 动作执行模块各组件连接关系

当无人控制系统运行时，逻辑控制模块中的策略执行组件会基于 BaseAct 端口对各个动作执行组件进行同步唤醒，特别地，对移动执行组件的唤醒是通过 BaseMove 端口进行的（还需要传递目标地点名称作为参数）。

对于每一个策略输出的动作，动作执行组件支持以下四种操作：启动本动作、检查本动作是否完成、结束本动作、检查本动作是否结束，并且该同步端口提供返回值反馈动作是否完成，或者动作是否结束。通过指定 BaseAct (BaseMove) 端口的操作符参数 (flag) 实现策略执行器在不同情况下对动作执行组件的操作。因此，为了让系统能够对每一个动作更清晰地进行识别、处理，动作执行组件中需要自定义这四种操作的具体实现（有些类型的动作只需要定义部分操作）。以下对不同类型的动作执行器所需实现的操作和实现流程进行讨论。

表 3-1 不同类型动作的定义方法

动作特点	同步不可中断动作	异步可中断动作	异步不可中断动作
一次性执行	✓	✗	✗
周期执行	✗	✓	✓
启动	✓	✓	✓
检查启动完成	✗	✓	✓
结束	✗	✓	✗
检查结束完成	✗	✓	✗

在实际无人系统场景中，由于动作处理机制的多样化，机器人的动作也存在不同的类型，本论文设计的动作执行组件模型基本对所有类型的机器人动作都提供了定义接口，不同类型的动作只需要参考表3-1对相应操作完成自定义实现即可。类型分类中的“同步”是指在策略执行器单个周期内完成动作所有行为；“异步”是指周期性唤醒该动作执行器，在独有的执行线程中执行动作，策略执行器线程负责唤醒执行线程以及检查执行结果；“可中断”是指动作可以接受停止信号。其中，具体各个类型的动作执行组件实现代码框架参见如下图3-7。

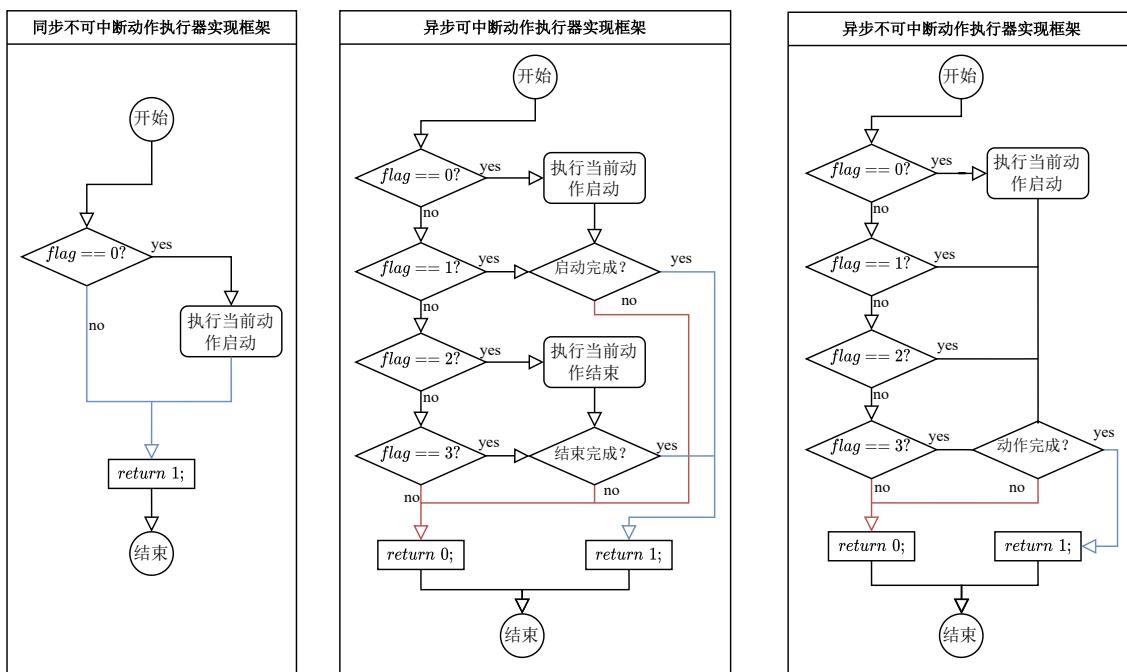


图 3-7 动作执行模块各类型组件实现模板流程图

由图3-6可知，无人控制系统架构除了通用动作执行组件（BaseAct）之外，对每一个机器人还都预设了两个标准动作执行组件：移动执行器（BaseMove）和初

始化执行器（BaseStart）。按照上表3-1可以对这两个动作执行器进行行为定义以及类型划分：

- **初始化执行器：**是机器人启动的一系列动作，在运行无人控制之前，必须先完成启动（无人机起飞至初始高度），因此此动作属于同步不可中断动作，只需要定义其启动行为即可（flag:0）。
- **移动执行器：**指定目标地点，完成机器人的移动动作。实际场景下，如果环境变化导致目标区域改变，那么机器人需要中断上一个移动命令，向新区域移动，因此此动作属于异步可中断动作，需要对所有操作定义行为。

而对于其它的通用动作执行组件，首先，实现时尽量选择异步方式（周期执行）定义不可中断动作和可中断动作。尤其对于耗时较长的不可中断动作（例如从高空降落），应该尽量选择异步方式处理。因为同步方式执行不可中断动作会导致策略执行组件阻塞，等待其动作完成，在整个过程中，策略执行器不会对外部环境变化以及全局运行状态变化做出响应并且无法做出其它动作。

其次，相比于异步不可中断动作，异步可中断动作包含对结束动作的行为实现，一般适用于可中断且持续有限时间的动作（例如移动，无人机爬升某段高度）或者持续无限时间的动作（例如发警报、录像等等）。异步可中断执行器可以定义结束该动作的行为，例如相应的停止移动、停止爬升、结束警报、结束录像等等。与之相对的，异步不可中断动作一般适用于不可中断且需要持续一段时间的动作，比如卸货、上传数据操作。

3.5 本章小结

本章基于 F Prime 框架设计实现了对程序综合策略进行解析运行的无人控制系统架构，具体而言，本章先总后分，在对无人控制系统分层结构介绍之后，就各个具体的基于组件的模块实现细节以及动作执行模块自定义方法进行了展示说明。

该无人控制系统架构的建立搭建起了规约综合和 F Prime 部署之间的桥梁，也为后续完整系统自动生成方法的设计奠定了基础。

第4章 基于规约的无人控制系统代码生成与演化方法

基于规约的反应式综合是程序合成领域中一个传统且经典的理论方法，在GR(1)博弈提出之后，其综合复杂度得到降低从而大大扩大了应用场景。其中基于GR(1)博弈的系统综合一个重要的应用领域就是机器人控制程序的自动合成^[3, 29]。

控制器作为机器人系统中直接决定机器人行为的核心部件，其编写实现是无人系统开发任务最关键的任务。总体而言，控制器综合就是在给定环境和机器人行为建模的基础上，对规约进行求解生成控制程序的过程。其中，规约是对该无人系统任务环境约束和机器人需求的形式化描述。

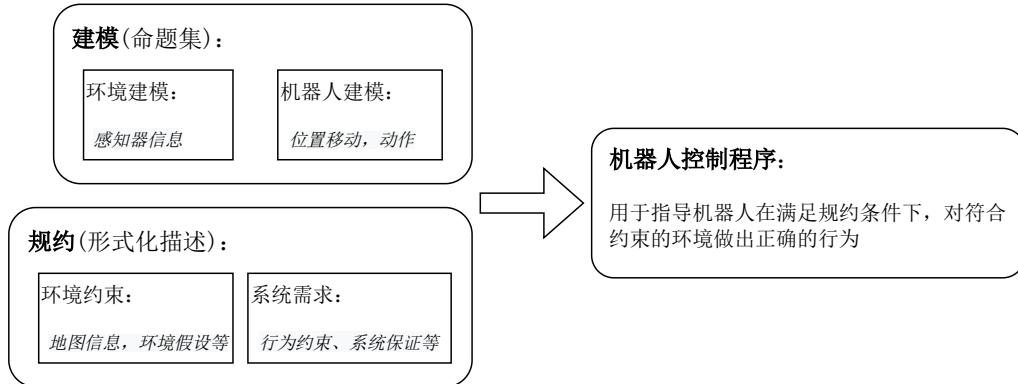


图 4-1 机器人控制程序综合基本流程

在上一章本文建立了 F Prime 架构下的无人控制系统架构模型，该系统可以对综合生成的控制程序策略进行解析运行。为了实现无人控制系统的完全自动生成，本章将会从以下两个阶段进行展开介绍：

- 基于 GR(1) 博弈的反应式综合原理，为单机器人生成对应的控制策略。
- 基于无人控制系统架构模型和得到的策略自动生成独立的单机器人无人控制系统。对于多机器人场景，则需要由开发者自定义各个单机器人控制系统之间的连接。

4.1 基于规约的单机器人控制逻辑综合

对于单机器人的控制程序综合问题可以转化为如图4-2所示基于规约的反应式综合问题。机器人对环境的感知信息抽象为输入命题，动作的执行抽象为输出命题，规约则由线性时序逻辑语言书写，控制程序在给定的输入（感知命题集的一组指派）上，要求在满足规约的条件下生成正确的输出（动作命题集的一组指派）。在本文中，将规约限制于 LTL 的子集 GR(1) 公式之上，实现博弈结构上系统获胜

策略的高效求解从而完成控制程序综合。

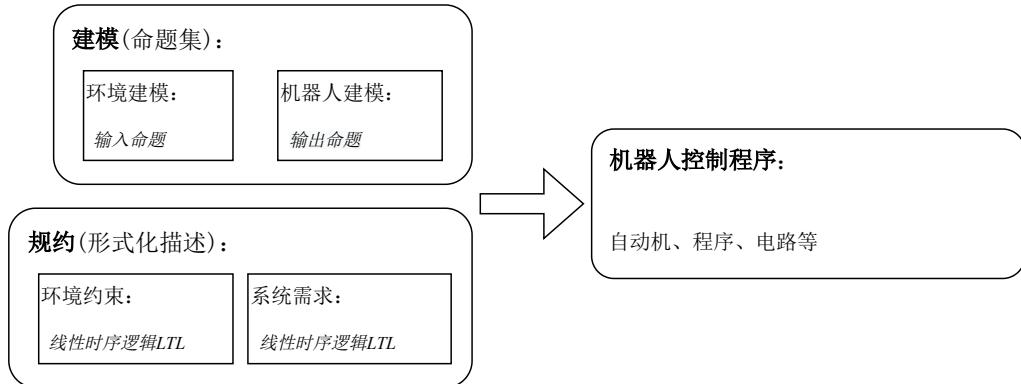


图 4-2 单机器人系统反应式综合

针对上面提出的单机器人控制逻辑综合问题，本文使用了 LTLMoP 工具进行建模以及规约生成，并且利用所提供的 JTLV 套件对 GR(1) 规约进行求解生成策略。具体流程见图4-3：

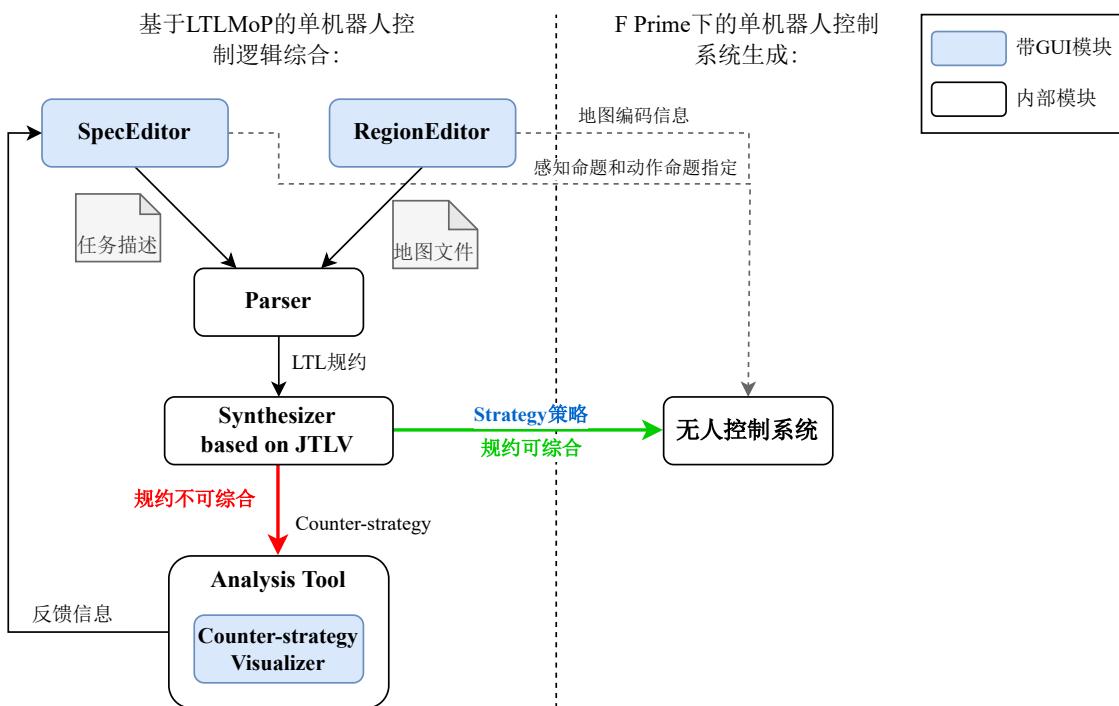


图 4-3 单机器人系统控制逻辑综合过程

4.1.1 问题建模

根据图4-2，需要对机器人任务问题进行抽象，即通过指定输入和输出命题对环境和机器人建模。以一个救援战斗空中巡逻机（RESCAP）的任务作为案例：

例 4-1：图4-4所示的作战地图将工作空间划分为了四个巡逻区域，并且在四个区域之间设置了禁飞区，救援战斗空中巡逻机不会进入禁飞区。启动之后，无人机从 *left* 区域出发，在四个区域上巡逻，主要执行两个任务：发现敌军战机就会停止并且发起攻击，敌机消灭后继续巡逻。发现待救援伤员就会拍照等待处理，救援结束后继续巡逻。

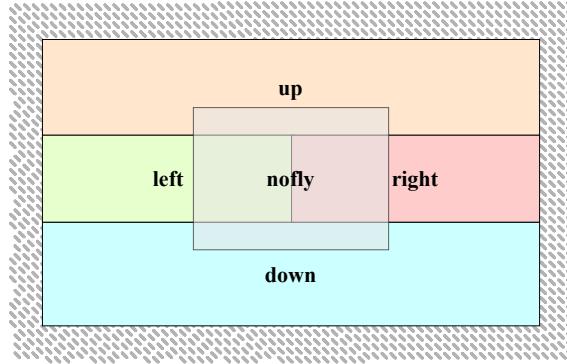


图 4-4 救援战斗空中巡逻任务地图

在进行该案例的反应式综合之前，需要对综合任务对应的 GR(1) 博弈结构进行问题建模。基于定义2-2中对博弈结构的形式化描述，问题建模主要包括环境变量集合 \mathcal{X} 和系统变量集合 \mathcal{Y} （输出命题集）的确定。

$$\mathcal{X} = Sensors$$

$$\mathcal{Y} = Actions \cup Customs \cup Regions$$

所有的环境变量和系统变量都是命题变量，其赋值只可能为真或假。其中，各个集合的含义如下：

- **Sensors:** 感知信息命题变量集合。机器人通过传感器获取外部环境状态。例如，“发现火情”、“发现垃圾”都属于感知信息命题。
- **Actions:** 动作命题变量集合。每一个动作命题都对应了机器人的一个实际动作。例如，“捡起”、“发出警报”都属于动作命题。
- **Customs:** 自定义系统状态集合。主要用于表示机器人处于某个动作已经被执行的状态，可以使得对机器人动作的规约描述更加真实丰富。系统状态命题需要定义激活动作和取消动作。例如，服务机器人“装货”动作会激活其自定义状态命题“装载状态”，而“卸货”会取消该状态。
- **Regions:** 地图区域命题集合。 $Regions = \{r_1, r_2, \dots, r_n\}$ 且 $r_i \cap r_j = \emptyset$ ，区域要求互不相交而且是凸图形。

对于在本搜救巡逻任务，原始地图4-4中 nofly 区域和其他区域存在重叠，显然不能直接用于综合建模。因此，在进行地图信息建模前需要对地图进行分解，使用 MP5 算法将各个区域划分为互不重叠的凸图形^[30]：

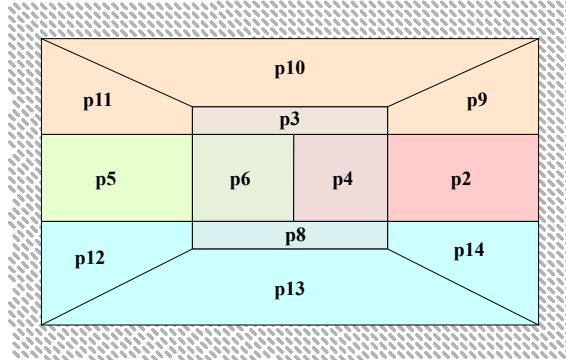


图 4-5 分解后的救援战斗空中巡逻任务地图

最终可以对于案例4-1进行问题建模如下：

表 4-1 案例4-1的问题建模

命题变量集合类型	命题变量集合组成
Sensors:	{enemy, ground_casualty}
Actions:	{attack, attack_over, take_photo}
Customs:	{attack_mode}
Regions:	{p2, p3, p4, p5, p6, p8, p9, p10, p11, p12, p13, p14}

4.1.2 规约设计与生成

在对无人系统任务进行问题建模之后，还需要设计规约对系统和环境需求进行描述。规约主要包括：基于地图生成的区域拓扑信息规约以及用户定义的规约。LTLMoP 提供了一种自然语言描述语法用于自动生成 LTL 规约公式，使得用户能够从更加高层次的描述中完成规约设计。以下是对本案例下自定义的任务自然语言描述。

- Env starts with false;
- you start in **left** and !nofly with false;
- Do **attack** if and only if you are sensing **enemy** and you are not activating **attack_mode**;
- If you are activating **attack** then stay there;

- **attack_mode** is set on **attack** and reset on **attack_over**;
- Do **attack_over** if and only if you are not sensing **enemy** and you are activating **attack_mode**;
- Do **take_photo** if and only if you are sensing **ground_casualty**;
- If you are activating **take_photo** or you were activating **take_photo** then stay there;
- Group **areas** is **left, up, down, right**;
- If you are not activating **attack_mode** and you are not activating **take_photo** then visit all **areas**;

根据前文中对 GR(1) 规约的定义2-6，规约可以划分为环境假设 φ^e 以及系统保证 φ^s 两种类型。

首先，环境假设的刻画包括以下三个方面 $\varphi^e = \varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e$:

- 对于 φ_i^e ，是对初始状态下环境感知信息系统命题的指派，一般设置为 false。在本案例中，初始环境变量全部设置为 false。
- 对于 φ_t^e ，是对环境变化的假设，即对某些状态下后继环境变化的约束。在本案例中，没有对环境变化提出假设。
- 对于 φ_g^e ，是对环境目标的约束，即未来一定会发生的环境状态。在本案例中，不论是 **enemy** 还是 **ground_casualty**，在未来都可能取真或假。

- $\varphi_i^e : \neg enemy \wedge \neg ground_casualty$
- $\varphi_t^e : \square true$
- $\varphi_g^e : \square \diamondsuit true$

然后，系统保证的刻画也包括三个方面 $\varphi^s = \varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s$:

- 对于 φ_i^s ，是对初始状态下输出命题的指派，包括机器人初始位置以及初始动作和状态。在本案例中，无人机从 **left** 出发，且不能位于禁飞区，其它动作初始都为 false。
- 对于 φ_t^s ，是系统动作保证，即对某些状态下环境发生改变后所采取动作的约束。在本案例中，对行为的定义以及约束都可以转化为 φ_t^s 。比如 **attack** 命令的执行条件、不能进入禁飞区、区域拓扑信息约束等等。特别地，对于任务描述中的该系统保证：“如果 **attack** 动作被激活，那么机器人待在原地”。待在原地的动作可以用基于区域编码的逻辑公式表示（本案例有 12 个区域，

需要 4 位编码): $\varphi_{stay} = (s.bit0 \leftrightarrow \bigcirc s.bit0) \wedge (s.bit1 \leftrightarrow \bigcirc s.bit1) \wedge (s.bit2 \leftrightarrow \bigcirc s.bit2) \wedge (s.bit3 \leftrightarrow \bigcirc s.bit3)$ 。

- 对于 φ_g^s , 是对行为目标的约束, 即未来一定会发生动作。在本案例中, 系统目标是指无人机默认会对所有区域进行巡逻。

- $\varphi_i^s : \neg attack \wedge \neg attack_over \wedge \neg take_photo \wedge \neg attack_mode p5 \wedge \neg p3 \wedge \neg p4 \wedge \neg p6 \wedge \neg p8$
- $\varphi_t^s : \square(\neg \bigcirc p3 \wedge \neg \bigcirc p4 \wedge \neg \bigcirc p6 \wedge \neg \bigcirc p8)$
 $\quad \square(\bigcirc enemy \wedge \bigcirc attack_mode \leftrightarrow \bigcirc attack)$
 $\quad \square(\bigcirc attack \rightarrow \varphi_{stay})$
 $\quad \square((attack \wedge \neg attack_over) \rightarrow \bigcirc attack_mode)$
 $\quad \square((attack_over \rightarrow \neg \bigcirc attack_mode)$
 $\quad \square((attack_mode \wedge \neg attack_over) \rightarrow \bigcirc attack_mode)$
 $\quad \square((\neg attack_mode \wedge \neg attack) \rightarrow \neg \bigcirc attack_mode)$
 $\quad \square((\neg \bigcirc enemy \wedge \bigcirc attack_mode) \leftrightarrow \bigcirc attack_over)$
 $\quad \square(\bigcirc ground_casualty \leftrightarrow \bigcirc take_photo)$
 $\quad \square((\bigcirc take_photo \vee take_photo) \rightarrow \varphi_{stay})$
 $\quad \square(p5 \rightarrow (p5 \vee p6 \vee p11 \vee p12))$
剩余的地图拓扑信息省略 (和上条格式一致)
- $\varphi_t^s : \square \diamond ((\neg attack_mode \wedge \neg take_photo) \rightarrow (p5 \vee p6))$
 $\quad \square \diamond ((\neg attack_mode \wedge \neg take_photo) \rightarrow (p3 \vee p9 \vee p10 \vee p11))$
 $\quad \square \diamond ((\neg attack_mode \wedge \neg take_photo) \rightarrow (p2 \vee p4))$
 $\quad \square \diamond ((\neg attack_mode \wedge \neg take_photo) \rightarrow (p8 \vee p12 \vee p13 \vee p14))$

4.1.3 基于 JTLV 套件的 GR(1) 博弈策略综合

对于反应式系统逻辑控制策略的生成可以转化为对以下 GR(1) 规约的可综合性求解: $(\varphi_i^e \wedge \varphi_t^e \wedge \varphi_g^e) \rightarrow (\varphi_i^s \wedge \varphi_t^s \wedge \varphi_g^s)$;

该公式求解可以进一步等价转化为 μ 演算公式2-2中对 φ_{gr} 的求解, 图4-6中是基于 JTLV 套件对 φ_{gr} 进行求解的代码实现。JTLV^[31] 是由 Pnueli 等人提出的一个基于 Java 实现的计算机形式化验证辅助工具, 为算法验证应用程序提供了最先进的集成开发环境。JTLV 对多种形式化规约 (包括 GR(1) 规约)、自动机及验证算法进行了实现, 自提出以来就被广泛应用于形式化验证以及综合领域^[32, 33]。

```

public BDD calculate_win() {
    BDD Z = TRUE;
    for (Fix fZ = new Fix(); fZ.advance(Z);) {
        mem.clear();
        for (int j = 1; j <= sys.numJ(); j++) {
            BDD Y = FALSE;
            for (Fix fY = new Fix(); fY.advance(Y);) {
                BDD start = sys.Ji(j).and(cox(Z)).or(cox(Y));
                Y = FALSE;
                for (int i = 1; i <= env.numJ(); i++) {
                    BDD X = Z;
                    for (Fix fX = new Fix(); fX.advance(X);)
                        X = start.or(env.Ji(i).not().and(cox(X)));
                    mem.addX(j, i, X); // store values of X
                    Y = Y.or(X);
                }
                mem.addY(j, Y); // store values of Y
            }
            Z = Y;
        }
    }
    return Z;
}

```

图 4-6 JTLV 中对 GR(1) 博弈对应规约 φ_{gr} 的求解实现

最终，可以从该求解过程中生成自动机作为控制程序策略。在这个求解过程中，如果规约是不可综合的将无法得到控制逻辑策略。比如假设缺少了初始位置“ $\neg nofly$ ”的声明，当无人机初始位于 left 和 nofly 的交界区域（p6）时就会直接不满足系统保证。从这个角度上来说，也证明了反应式综合可以对规约描述进行可综合性检查，从而对系统的正确性和安全性作出一定的保证。图4-7展示了本案例最终生成的策略自动机部分状态。

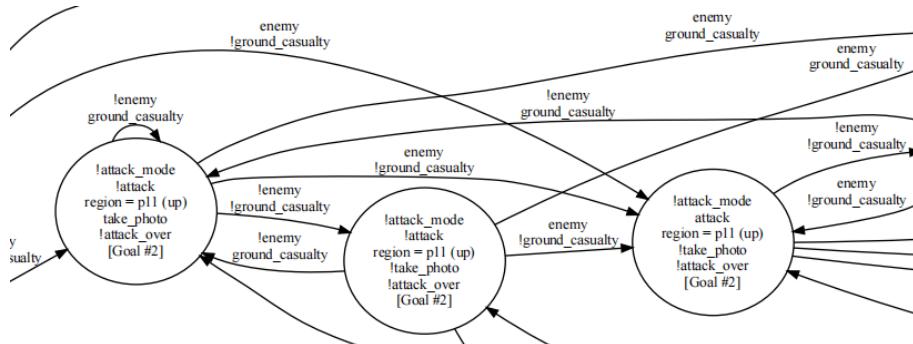


图 4-7 救援战斗空中巡逻机综合策略自动机部分状态图

4.2 基于架构模型的无人控制系统代码自动生成

对于每一个机器人，在使用反应式综合方法生成逻辑策略之后，还需要自动生成解析运行该策略的单机器人无人控制系统。基于在第3章中提出的无人控制系 统架构，本文设计实现了适用于指定机器人任务的无人控制系统实例生成方法。该单机器人无人控制系统的代码生成包括对各个组件的 FPP 描述文件、CPP(HPP) 组件定义文件以及 CMake 编译文件完成实现。

对于单机器人场景，以上方法可以实现上层无人控制系统的完全自动生成，用户只需要自定义动作执行模块即可获取基于任务描述的可执行部署。对于多机器人场景，除了指定动作执行模块行为，用户还需要在 F Prime 中自定义机器人之间的通信机制，该通信机制通过设计连接两个机器人的控制逻辑模块之间的端口实现。本节将会对单机器人和多机器人场景下无人控制系统的自动生成过程进行详细介绍。

4.2.1 单机器人场景下无人控制系统生成

本文基于无人控制系统架构模型设计了如图4-8所示的无人控制系统生成程序 FPBuilder，该程序以问题建模中的感知命题集（Sensors）、动作命题集（Actions）和自定义动作状态集（Customs）作为输入，可以在指定的 F Prime 部署目录中生成单机器人无人控制系统中的各个模块所需 FPP 描述文件、CPP 文件、HPP 文件。

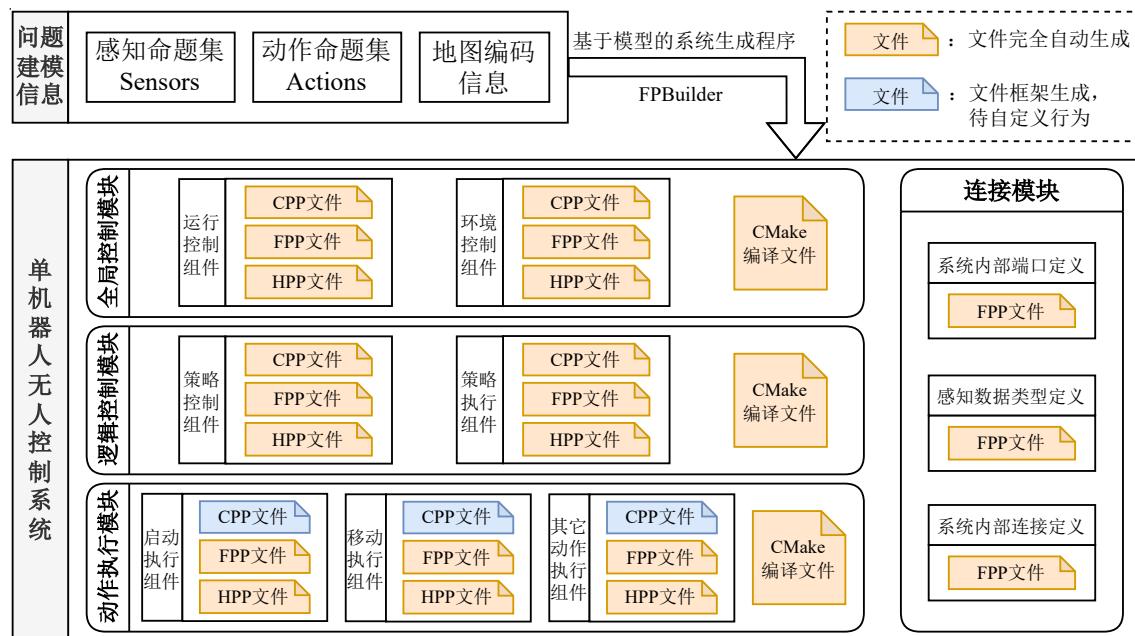


图 4-8 单机器人无人控制系统代码生成器 FPBuilder

首先，FPBuilder 根据任务的名称更新了系统部分模块和组件的名称，以案例4-1名称 ResCap 为例，其文件结构将更新为：

- 全局控制模块：包括运行控制组件（SimManager）和环境控制组件（EnvManager），此模块负责所有机器人的控制，名称是固定的。
- 逻辑控制模块（ResCapMission）：包括策略控制组件（ResCapController）和策略执行组件（ResCapExecutor）。
- 动作执行模块（ResCapActuators）：包括所有动作命题所对应的动作执行组件：ResCap_attack、ResCap_attack_over、ResCap_take_photo。

同时，FPBuilder 可以自动生成各个模块的编译文件（CMakeLists.txt）。F Prime 系统的部署运行在一个主执行程序上，各个组件组成的模块在编译时创建相应静态链接库，主执行程序运行时会对每一个模块中的组件进行实例化并且运行。为了让无人控制系统中的各个模块添加到 F Prime 部署中，需要将这些模块通过 CMake 文件加入 F Prime 部署的编译流中。F Prime 提供了添加编译的接口方法，本文直接基于该方法对于每个模块自动生成相应的编译配置文件。

然后，对于各个模块中的组件，FPBuilder 根据问题建模输入对其 FPP 描述文件、CPP 文件和 HPP 文件分别进行自动生成。F Prime 中组件的定义在 c++ 框架下会实现为 c++ 类，该类继承自 F Prime 中的基本组件类（主动型组件类继承自 ActiveComponentBase），具有特有的端口处理函数、命令处理函数、端口调用函数以及事件记录函数（由 FPP 描述文件指定）。因此，要使得无人控制系统中的各个模块组件添加至 F Prime 部署中，需要对这三种文件都完成实现。

- FPP 描述文件：定义了组件的输入端口、输出端口、支持的命令类型、事件类型和通道数据类型。对于不同组件，基于其相应 FPP 模型完成自动生成。
- HPP 描述文件：包含对于组件类成员变量和相关函数的声明，包括基于 FPP 自动生成的处理函数以及组件自定义的函数和成员变量（策略执行组件中会自定义一个 Strategy 类实例作为成员变量）。对于不同组件，基于其相应 HPP 模型完成自动生成。
- CPP 描述文件：包含对 HPP 文件中函数的实现，定义了组件的行为。对于不同组件，基于其相应 CPP 模型完成自动生成。

最后，FPBuilder 会更新 F Prime 部署中的连接模块。一方面，组件的顺利执行需要完成在 F Prime 部署中完成组件实例的信息注册（更新 instance.fpp），对于本无人控制系统中的策略控制组件和策略执行组件，由于它们属于主动型组件，还需要在信息注册中指明组件的输入队列和栈规模。另一方面，还需要更新 F Prime 部署中对于无人控制系统中组件基于端口的连接关系（更新 topology.fpp）。

FPBuilder 实现了在 instance.fpp 和 topology.fpp 指定位置的数据自动更新加载。对于本案例，最终生成的部分组件端口连接示例如下：

- 全局控制模块和逻辑控制模块之间的连接：

envManager.envSetout → resCapExecutor.envSetin

simManager.simControlout → resCapController.simControlin

- 逻辑控制模块中策略控制组件和策略执行组件之间的连接：

resCapController.resCapWakeout → resCapExecutor.resCapWakein

resCapController.resCapExeCtrlout → resCapExecutor.resCapExeCtrlin

resCapExecutor.resCapCallbackout → resCapController.resCapCallbackin

- 逻辑控制模块和动作执行模块之间的部分连接：

resCapExecutor.resCapStartout → resCapStart.resCapStartin

resCapExecutor.resCapMoveout → resCapMove.resCapMovein

resCapExecutor.resCap_attack_out → resCap_attack.resCap_attack_in

综上，FPBuilder 生成程序可以完成对单机器人无人控制系统自动生成。

4.2.2 多机协同场景下无人控制系统实现

相比于单机器人场景，多机器人协同任务需要对机器人系统之间的交互机制进行定义。在实际的机器人协同场景下，机器人 A 对机器人 B 的唤醒或者行为控制往往通过发送某个信号实现，机器人 B 感知到该信号之后就会做出相应的动作或者进入相应的运行状态。在无人控制系统中的协同机制实现包括两个过程：协同通信命题规约设计以及协同行为实现。

本论文将常见的协同任务分成两类：基于简单唤醒的协同（只唤醒不传递信息）和基于信息传递的协同（带信息传递的唤醒）。接下来本章将会对不同类型的协同任务分别基于案例说明实现方法，首先考虑以下一个简单的协同案例：

例 4-2： 巡逻侦查机 A 和报警机器人 B 交互完成侦查敌军任务。A 在地图中进行巡逻，B 始终待在指挥所。一旦 A 探测到敌人入侵（enemy），就会停在原地并且唤醒 B，B 会发出警报。

在该任务场景下，机器人 B 会接收来自 A 的交互信号（假设该信号用感知命题“A 发现敌人”（a_find_enemy）表示）。B 对该命题的相关规约表示为：

- $\square(\bigcirc a_find_enemy \leftrightarrow \bigcirc alarm)$

为了实现 A 对 B 的调用，还需要在无人控制系统中添加一个如图4-9所示的交互端口从而完成协同行为的实现。首先需要实现 A 对该端口的调用：一旦 A 检测到 `enemy` 赋值为真，就对交互端口进行唤醒；然后需要实现 B 对该交互端口的处理：一旦 B 接收到了来自该交互端口的唤醒，就会修改当前感知环境中对应的通信命题赋值 ($a \ find \ enemy = true$)。

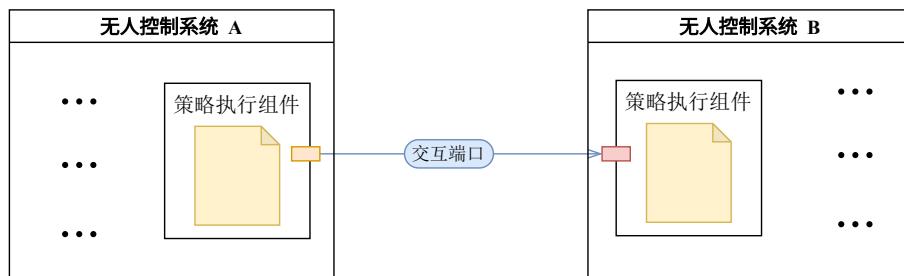


图 4-9 案例4-2的交互机制实现

案例4-2中定义的交互信号是最基本的唤醒信号， B 对该交互信号的响应可以直接表示在规约之中。但是，有些场景下，机器人之间的交互还需要在唤醒的同时互相传递信息（比如位置信息等等），而这个行为的实现显然是不能在 B 规约直接定义的，比如以下案例：

例 4-3: 地雷探测无人机 R_1 和排雷车 R_2 交互完成地雷的探测与排除任务。 R_1 在地图中进行巡逻，一旦 R_1 探测到地雷（landmine），就会停在原地并且向 R_2 发送出发命令， R_2 会移动到该位置进行排雷。

在该场景下，“A 发现地雷”(find_landmine) 可以视为协同通信命题， R_2 在该接受该命题时，需要去 R_1 的位置，规约表示为：

- $\square \diamond (find_landmine \rightarrow (Region\ of\ R_1))$

在不知道 R_1 的位置情况下，该规约是无法完成书写的。这种场景下的协同实现主要有两种角度：扩展动作命题集和扩展感知命题集。

(1) 扩展动作命题集：设计实现一个新的动作命题，对于本案例可以引入 any region 动作，该动作表示机器人向任何位置移动。其规约描述如下：

- $\square(\bigcirc \text{find_landmine} \leftrightarrow \bigcirc \text{any_region})$

通过这一方式，可以基于 F Prime 中的端口实现位置信息的传递，该交互端口以 R_1 位置信息作为参数， R_1 唤醒端口时传入当前区域名， R_2 处理唤醒时调用 Move 动作执行组件向该区域前进。

(2) 扩展感知命题集：对所有 R_1 可能出现的位置设计新的感知命题。假设本案例中包括两个区域： $Regions = \{p1, p2\}$ ，对应扩展两个感知命题： $\{\text{enemy_in_p1}, \text{enemy_in_p2}\}$ 。其规约描述如下：

- $\square \diamond (\text{enemy_in_p1} \rightarrow p_1)$
- $\square \diamond (\text{enemy_in_p2} \rightarrow p_2)$

该交互端口还是以 R_1 位置信息作为参数， R_1 唤醒端口时传入当前区域名， R_2 处理唤醒时只需要根据当前区域名的不同修改对应的感知命题赋值即可。

综上，通过设计协同通信命题及其规约，并且基于 F Prime 端口实现协同行为就可以将单机器人无人控制系统连接起来，完成各种类型的协同任务。

4.3 无人控制系统代码生成集成工具原型实现

LTLMoP 为反应式综合过程提供了一个可视化的操作平台，用户在该平台中定义相关命题以及规约描述，就可以对反应式系统可综合性进行分析并且获取自动生成策略。为了将无人控制系统代码自动生成的过程集成到 LTLMoP 工具中，本论文基于 LTLMoP 原有的 GUI 界面和功能进行了扩展开发。

4.3.1 工具原型功能

本论文设计的集成工具原型主要在 LTLMoP 上扩展实现了以下功能：

1. 设置 F Prime 部署文件夹根目录。如果用户指定的目录不是 F Prime 部署目录，就会引发设置失败错误。
2. 清理 F Prime 部署文件夹下的所有无人控制系统相关模块。
3. 以覆盖模式生成单机器人无人控制系统，适用于单机器人场景。在完成规约编译后可执行。
4. 以附加模式生成多机协同系统生成。部署中对连接模块的生成是增量式的，不会清除已有的无人控制系统相关配置。

5. 展示原图以及分解后的地图。该地图主要用于之后仿真环境地图的搭建。

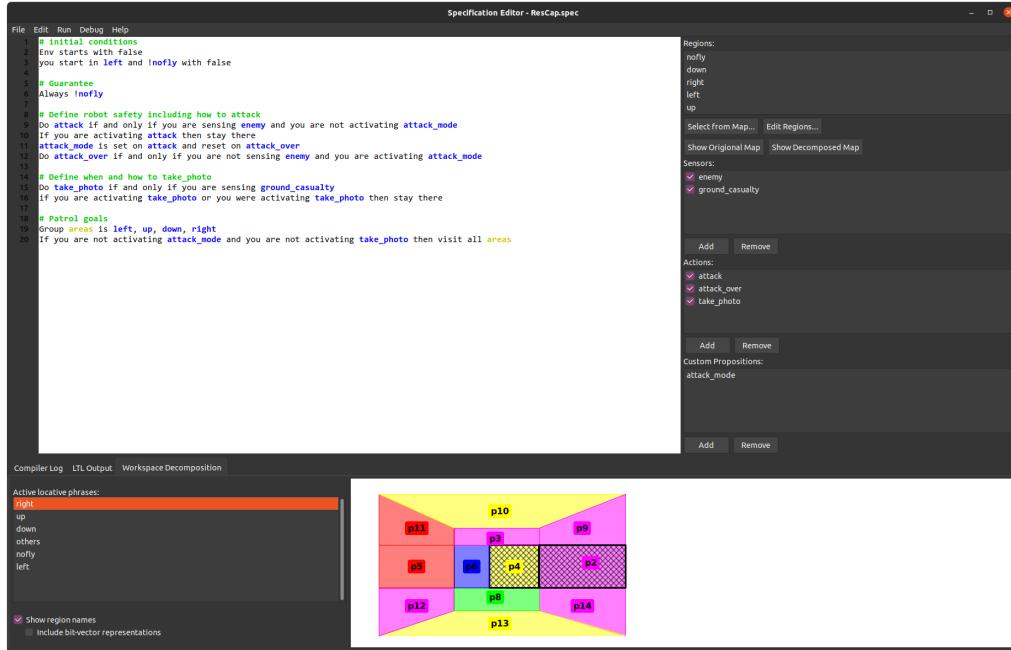


图 4-10 基于 LTLMoP 扩展开发的集成工具界面

4.3.2 工具原型使用

基于案例4-1使用该工具进行规约生成以及无人控制系统生成，最终生成的系统能够在 F Prime 下编译成功并且正确执行。

Events				
Event Time	Event Id	Event Name	Event Severity	Event Description
2022-05-23T04:17:52.20Z	0x001	cmdDisp.OpCodeDispatched	COMMAND	resCapController.START dispatched to port 15
2022-05-23T04:17:52.20Z	0x0f0	resCapExecutor.INIT_SUCC	ACTIVITY_HI	[ResCap] Strategy initializes successfully. Specification is located at [Ref/ResCapMission/Config/ResCap.spec]
2022-05-23T04:17:52.20Z	0x0f3	resCapExecutor.START_RECV	ACTIVITY_HI	[ResCap] Simulation Starts ...
2022-05-23T04:18:00.20Z	0x0e0	resCapStart.STARTFINISHED	ACTIVITY_HI	[ResCap] Starting actions have finished.
2022-05-23T04:18:00.20Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #0 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p5/left.
2022-05-23T04:18:05.20Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p11.
2022-05-23T04:18:05.20Z	0x0f2	cmdDisp.OpCodeCompleted	COMMAND	resCapController.START completed
2022-05-23T04:18:05.20Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #1 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p11/up.
2022-05-23T04:18:10.21Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p5.
2022-05-23T04:18:10.21Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #10 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p5/left.
2022-05-23T04:18:15.21Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p12.
2022-05-23T04:18:15.21Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #19 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p12/down.
2022-05-23T04:18:20.21Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p13.
2022-05-23T04:18:20.21Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #27 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p13/down.
2022-05-23T04:18:25.21Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p14.
2022-05-23T04:18:25.21Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #36 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p14/down.
2022-05-23T04:18:30.21Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p2.
2022-05-23T04:18:30.21Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #44 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p2/right.
2022-05-23T04:18:35.21Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p9.
2022-05-23T04:18:35.21Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #62 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p9/up.
2022-05-23T04:18:40.21Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p10.
2022-05-23T04:18:40.21Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #61 => @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p10/up.
2022-05-23T04:18:45.21Z	0x0f0	resCapMove.MOVEFINISHED	ACTIVITY_HI	[ResCap] Arrived at p11.

图 4-11 案例4-2无人控制系统的运行示例

在工具使用后，为了更直观地体现系统运行过程，对系统中的动作执行模块进行了实现。这里的实现只是为了展示系统运行效果，并没有考虑和仿真环境相连。因此只对移动命令和初始动作命令进行了定义，实现方式简化为等待一段时间之后再完成动作，其它动作的执行都是不进行处理直接完成。最终运行结果见上图4-11。

4.4 本章小结

本章对基于规约的无人控制系统代码生成与演化方法进行了基于具体案例的深入探讨与详细介绍，方法划分为基于规约的单机器人控制逻辑综合以及基于架构模型的无人控制系统自动生成。并且，本章对单机器人无人控制系统生成以及多机协同控制系统演化的具体方法都进行了解释。

最终，本章还对论文基于 LTLMoP 设计的无人控制系统自动生成工具原型进行了功能介绍和基于案例的结果展示。

第 5 章 基于 ROS 的无人控制系统综合仿真

为了验证本论文提出的无人控制系统生成方法的有效性和正确性，同时，也为了体现该无人控制系统的“底层无关”的软件定义特性，本章给出单机器人场景和多机器人场景下的仿真实验结果。

为了让仿真实验尽可能贴近真实情景，论文选择了基于 ROS 的底层仿真环境（模拟器使用 Gazebo 以及 Rviz），对机器人的各个行为完成了实现。

5.1 ROS 机器人操作系统

ROS^[34] (robot operating system) 是一个用于编写机器人软件程序的开源软件架构，虽然它不是严格意义上的操作系统，但是它为机器人软件开发提供了一个具有丰富服务的平台，其中包含了一系列硬件控制抽象、程序库、第三方工具软件和约定协议。ROS 的提出极大程度提高了机器人软件开发效率，如今，ROS 在学术界和工业界都已经建立起了广泛完整的生态。

通过将本论文提出的无人控制系统直接基于 ROS 进行实验，更能让其实际应用性得到验证，也有利于后续将无人控制系统部署在实际机器人上。

5.1.1 ROS 通信机制与文件结构

一个 ROS 系统是由许多 ROS 节点组成的，每一个 ROS 节点对应一个执行具体任务的进程，开发者可以使用不同的编程语言编写生成节点的可执行文件，ROS 节点可以分布式运行在不同的主机上。当 ROS 启动时，主节点（Master Node）会被初始化。主节点负责跟踪哪些节点已注册以及建立节点之间的通信。下图简单展示了主节点和其它自定义节点之间的关系：

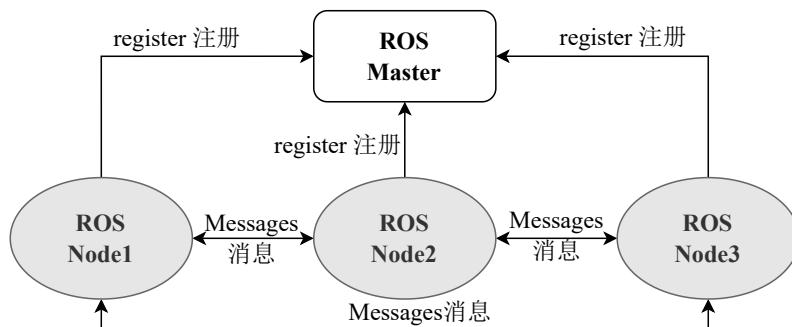


图 5-1 ROS 系统中主节点和其它节点之间的关系

ROS 系统通信机制的核心就是发布订阅机制。节点之间通过话题传输数据，

话题的实现基于发布、订阅模型。话题和某一类数据类型绑定（通过 msg 文件定义），发布者发布话题内容，订阅者则会获取话题内容。其中，一个 Topic 主题上可以拥有多个发布者和订阅者，同时一个节点也可以发布或订阅多个主题。

ROS 文件系统中，功能包是基本组织结构，一个典型的 ROS 功能包文件结构如图所示，其中基本内容包含了记录基本信息的功能包清单、节点实现源码、消息和服务的类型定义。

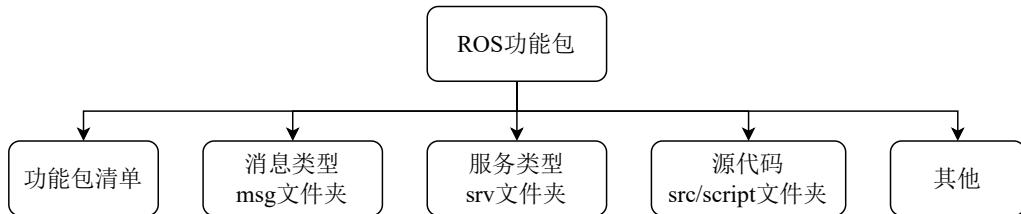


图 5-2 ROS 功能包组成结构

5.1.2 基于 ROS 的仿真环境

本论文中实验使用到的基于 ROS 的仿真模拟器包括 Gazebo^[35] 和 RViz^[36]。其中 Gazebo 是三维仿真平台，旨在搭建一个高保真度的物理仿真环境。而 Rviz 作为一个三维可视化工具，强调把运行系统中的数据进行可视化显示。

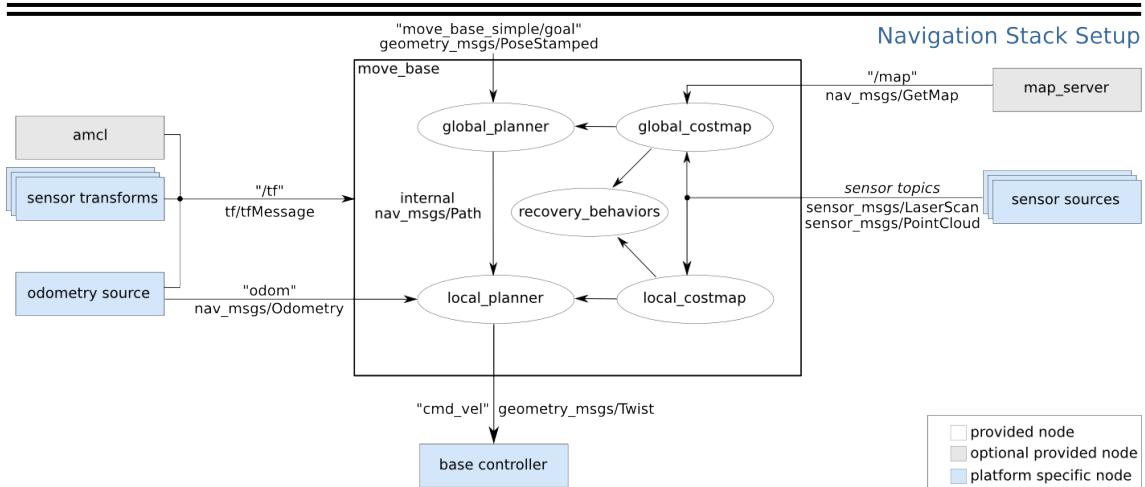
Rviz 是对已有数据的显示。Rviz 提供了很多插件，这些插件可以显示以话题、参数形式被 Rviz 订阅获取的图像、模型、路径等信息。一方面，Rviz 可以对可监测信息进行可视化展示，另一方面，开发者可以基于 Rviz 对机器人行为进行控制。比如机器人 SLAM 建图以及导航过程就是在 Rviz 中实现控制。

相比于 Rviz，Gazebo 更强调仿真，使用 Gazebo 仿真机器人运动功能，可以同时完成相关传感器数据的仿真生成。Gazebo 和 Rviz 的结合使用往往可以完成复杂的仿真实验任务。

5.1.3 ROS 导航功能包

ROS 导航功能包 (ROS Navigation Stack) 是 ROS 提供的一个开源功能包，主要功能就是实现机器人的导航与路径规划。该导航功能包以里程计、传感器信息和目标位姿作为输入，在完成规划以及一系列数据转换之后可以输出机器人到达目标状态所需要的安全速度指令。

ROS 导航功能包为移动机器人在地图中实现避障导航功能提供了定义良好的接口，只需要实现这些接口就能将导航算法应用与自定义的机器人平台上。图5-3展示了 ROS navigation stack 在机器人上设置部署的整体实现思路：

图 5-3 ROS Navigation Stack 整体架构^[37]

以下对该架构中的输入数据进行简要总结：

- **odometry (里程计信息)**: 里程计对机器人的运动距离和速度进行估计，在 ROS 导航功能包中用于局部规划器行为规划以及基于估计位姿信息定位。
- **sensor (传感器信息)**: 一般来自于 IMU、激光雷达和深度相机，用于定位和避障。一般而言，使用的传感器种类越丰富，越有利于提高定位和避障效果。
- **tf 变换信息**: 是一个用于建立多个参考系之间转换关系的功能包。在这里包括 amcl 得到的机器人参考系和地图参考系之间的关系，以及基于传感器属性的传感器与机器人参考系之间的关系。
- **map (地图信息)**: 通过 SLAM 实时或者预先构建。在本论文中高精度地图会预先通过 gmapping 获取，直接以 yaml 文件的形式提供给机器人。

定位方面，除了基本的里程计定位，自适应蒙特卡洛定位（AMCL）^[37] 是 ROS Navigation Stack 中指定的定位算法，它是一种基于概率的定位算法。通过在全局地图中撒粒子（可以理解为机器人的可能位姿），按照评价标准对粒子进行过滤使其在运动过程中集中在位置可能性高的地方。自适应体现在根据粒子的平均分数或者粒子是否收敛来对粒子数量进行调整。AMCL 在导航功能包中主要用于输出 $map \rightarrow odom$ 的 tf 变换，弥补里程计的漂移误差从而获取更准确的位姿估计。

路径规划方面，基于 move_base 模块，该模块整合输入信息，接受目标位置指导导航任务完成，主要包含全局规划、局部规划以及恢复行为。全局规划基于 A* 和 dijkstra 规划好全局路径，局部优化则基于动态窗口法（DWA）^[38] 读取局部代价地图在尽量符合全局最优路径条件下实现实时避障。

当路径规划失败时，机器人就会执行 ROS 导航包中定义的恢复行为。默认情况下，机器人会通过旋转底盘一周尝试清除障碍物。如果能重新规划则继续导航，

如果还是不能则终止导航任务。

5.2 单机控制系统仿真案例

本论文的仿真实验选择了无人系统场景中最常见的无人机和无人车作为机器模型。如图5-4所示，无人机模型选择了由 Johannes 等人设计的四旋翼飞行器 hector quadrotor^[39]，无人车则选择了已经完成 ROS 导航包配置的 turtlebot3^[40]。

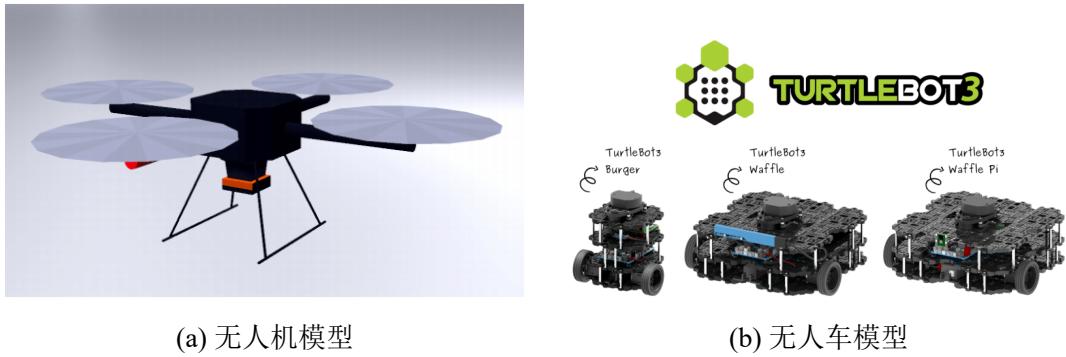


图 5-4 仿真实验的机器人模型

首先，本节将对单无人控制系统的仿真实现进行介绍总结，具体内容是对前文提到的搜救巡逻案例4-1所生成的无人控制系统，基于 ROS 话题订阅与发布机制实现其中的动作执行模块。

5.2.1 仿真环境搭建

在运行该无人控制系统之前，需要完成仿真环境初始化，包括地图模型的构建以及相关 launch 文件的实现与启动。

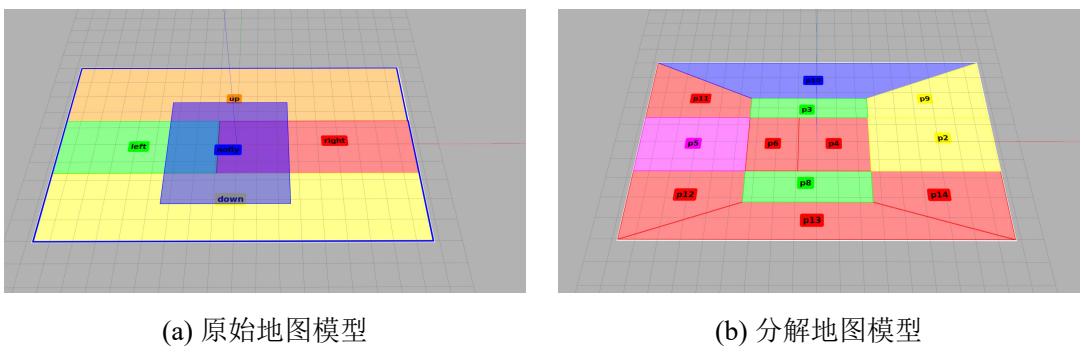


图 5-5 仿真实验的地图模型展示

对于单个运行场景，一般包括如图5-5所示两个地图：用于对区域信息进行配置的分解地图（decomposed map），以及原始地图（base map）。这两个地图的底图

都可以通过工具原型获取。在本案例场景下，为了简化，对每一个分解区域选取一个代表点，区域的到达与否取决于是否到达其代表点附近。在分解地图中对每一个区域的位置信息完成配置，最终仿真是在原始地图上的。

在本案例中主要的 launch 文件就是初始化 launch 文件，该文件负责将地图模型以及机器人模型分别导入 gazebo 仿真环境中，其中机器人模型的位置与规约初始位置描述一致 ($left \wedge \neg nofly$)。根据反应式综合生成的自动机，本案例中的无人机初始位于 p5 区域。最终得到的基础仿真环境见下图：

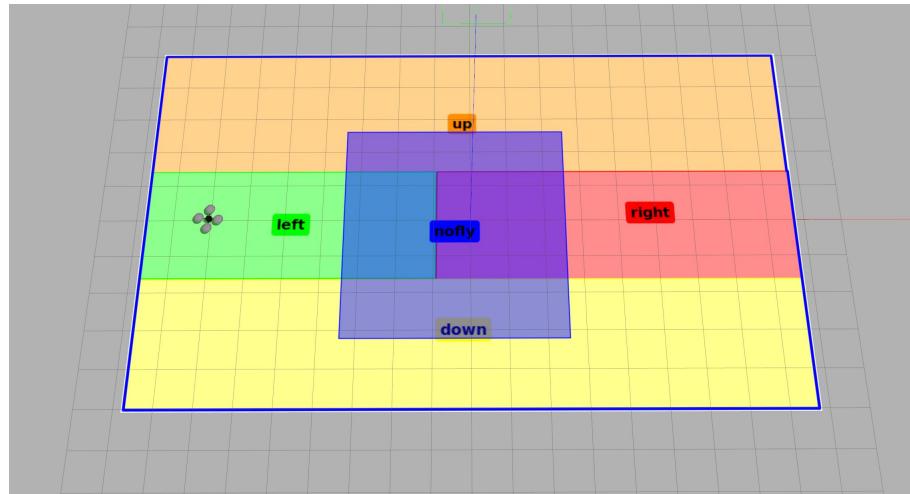


图 5-6 案例4-1下 ResCap 任务初始仿真环境加载

5.2.2 动作执行模块实现

在基于 ROS 的无人控制系统仿真中，动作执行模块是上层系统与仿真环境连接的桥梁。在仿真环境构建完毕之后，环境中预定义的话题和服务可以实时展示机器人状态以及控制机器人行为，例如控制机器人移动的 `/cmd_vel` 话题。

ResCap 任务规约描述中定义了以下动作行为，现分别对其实现思路进行简单介绍：

- **Start**: 属于同步不可中断动作，在执行任务之前无人机需要先完成起飞，通过发布 `/takeoff` 话题完成该动作。
- **Move**: 移动属于异步可中断动作，根据给定区域名获取速度向量并发布 `/cmd_vel` 话题进行移动。移动是一个持续动作，移动检查是通过订阅 `/ground_truth_to_tf/pose` 来判断是否已经移动至指定区域。在仿真运行过程中，可以适应环境变化修改目标区域。
- **attack**: 进攻属于异步不可中断动作，是对进攻状态的激活，主要行为是降低飞行高度进入进攻状态，通过发布 `/cmd_vel` 话题实现。

- **attack_over**: 进攻结束属于异步不可中断动作，是退出进攻状态，主要行为是提高飞行高度继续巡逻，通过发布 /cmd_vel 话题实现。
- **take_photo**: 对伤员拍照是一个即时动作，在本案例实现中不考虑上传时延，因此认为拍照动作一启动就完成，属于同步不可中断动作。通过请求 /image_saver/save 服务完成照相机图片的保存上传。

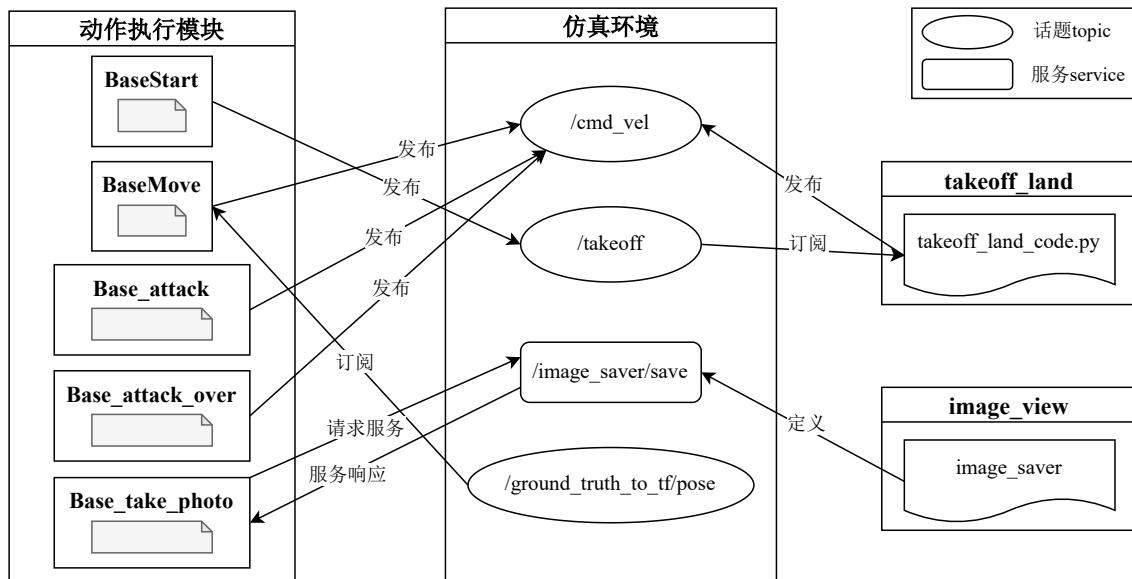


图 5-7 案例4-1无人控制系统动作执行模块基于 ROS 的通信实现

5.2.3 仿真结果与分析

在本实验过程中，用户通过 F Prime 地面数据系统发送命令对环境感知信息进行修改。经过测试，无人机在仿真环境中的行为和案例任务描述一致。

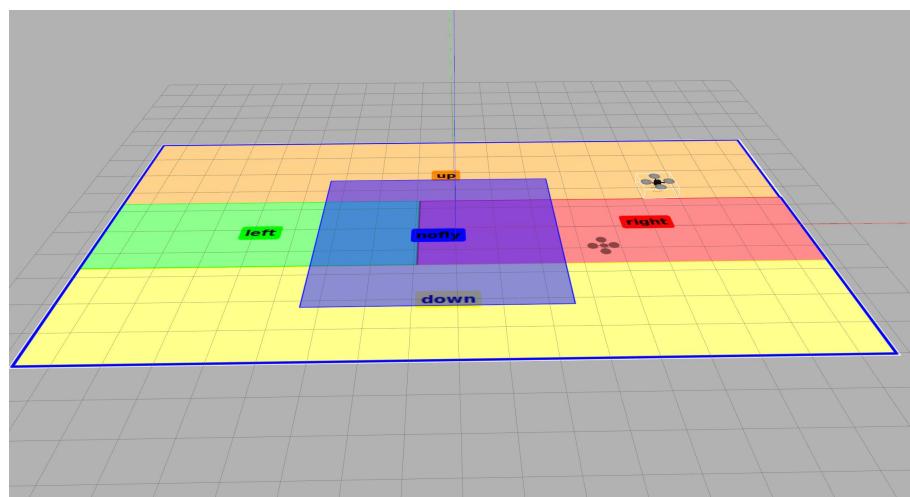


图 5-8 ResCap 发现敌机进入攻击状态

在环境不变的情况下，无人机会对可飞行区域进行巡逻。当发现敌机时，相对应的 Gazebo 环境中无人机会停止并且进入降低高度进入攻击状态（见图5-8）。发现伤员则会停止并且将下方图片保存至指定目录。下图是整个测试过程中 F Prime 的事件记录，运行时事件和仿真环境的行为是同步的。

Event Time	Event Id	Event Name	Event Severity	Event Description
2022-05-20T13:57:43.800Z	0x0f0	resCapExecutor.INIT_SUCC	ACTIVITY__LO	[ResCap] Strategy initializes successfully. Specification is located at: [Ref/ResCapMission/Config/ResCap.spec]
2022-05-20T13:57:43.801Z	0x0f3	resCapExecutor.START_RECV	ACTIVITY_HI	[ResCap] Simulation Starts --
2022-05-20T13:57:51.801Z	0x460	resCapStart.STARTFINISHED	ACTIVITY__LO	[ResCap] Starting actions have finished.
2022-05-20T13:57:51.801Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #0 === @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p0(left).
2022-05-20T13:57:56.802Z	0x460	resCapMove.MOVEFINISHED	ACTIVITY__LO	[ResCap] Arrived at p11.
2022-05-20T13:57:56.802Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #1 === @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p11(up).
2022-05-20T13:58:01.802Z	0x460	resCapMove.MOVEFINISHED	ACTIVITY__LO	[ResCap] Arrived at p5.
2022-05-20T13:58:01.802Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #10 === @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p5(left).
2022-05-20T13:58:06.221Z	0x0f2	resCapExecutor.ENV_UPDATE	ACTIVITY_HI	>> [ResCap] Environment Sensor Updates: enemy = 1
2022-05-20T13:58:06.803Z	0x460	resCapMove.MOVEFINISHED	ACTIVITY__LO	[ResCap] Arrived at p12.
2022-05-20T13:58:06.803Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #19 === @Sensor: enemy[0] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p12(down).
2022-05-20T13:58:06.803Z	0x3000	resCap_attack.ACCTIONFINISHED	ACTIVITY__LO	[ResCap] Action: attack has finished.
2022-05-20T13:58:11.804Z	0x0f1	resCapMove.STAYFINISHED	ACTIVITY_LO	[ResCap] Still Stay at p12.
2022-05-20T13:58:11.804Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #29 === @Sensor: enemy[1] ground_casualty[0]; @Action: attack[1] attack_over[0] take_photo[0]; @Customs: attack_mode[0]; @Region: p12(down).
2022-05-20T13:58:11.804Z	0x3000	resCap_attack.ACCTIONFINISHED	ACTIVITY__LO	[ResCap] Action: attack has finished.
2022-05-20T13:58:16.804Z	0x0f1	resCapMove.STAYFINISHED	ACTIVITY__LO	[ResCap] Still Stay at p12.
2022-05-20T13:58:16.804Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #33 === @Sensor: enemy[1] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[1]; @Region: p12(down).
2022-05-20T13:58:21.805Z	0x0f1	resCapMove.STAYFINISHED	ACTIVITY__LO	[ResCap] Still Stay at p12.
2022-05-20T13:58:21.805Z	0x0f1	resCapExecutor.SHOW_TRANSITION	ACTIVITY_HI	[ResCap] State #33 === @Sensor: enemy[1] ground_casualty[0]; @Action: attack[0] attack_over[0] take_photo[0]; @Customs: attack_mode[1]; @Region: p12(down).
2022-05-20T13:58:21.805Z	0x0f4	resCapExecutor.STOP_RECV	ACTIVITY_HI	[ResCap] Simulation Stopped --

图 5-9 F Prime 事件相关记录

综上，本文设计的单机器人无人控制系统在可以实现基于规约的机器人行为逻辑控制，并且该系统能应用于贴近真实的仿真环境。

5.3 多机协同控制系统仿真实例

为了验证本文提出的基于端口协同的多机无人控制系统的正确性，同时也为了测试系统在复杂环境中的鲁棒性，本节基于 ResCap 搜救巡逻机设计了一个包含信息传递的人机物协同场景。在无人机的基础上引入了无人车，二者共同完成一个完整的巡逻搜救任务。

5.3.1 案例说明

在案例4-1基础上对救援车任务进行定义：

例 5-1：基于案例 4-1 的地图和搜救巡逻战斗机 ResCap，引入救援车 ResCar。ResCar 初始位于 left 区域，当 ResCap 发现伤员，不仅会停在原地拍照上传，还会唤醒 ResCar。救援车 ResCar 会选取一条最优路径移动至 ResCap 所在位置转移伤员，伤员会被转移至 left 区域。一旦伤员转移完毕，ResCap 会继续巡逻。

为了实现 ResCap 和 ResCar 之间的协同,首先定义协同通信命题“`find_casualty`”,然后通过扩展动作命题 (“`any_region`”) 完成 ResCar 和 ResCap 协同的规约设计,其中 ResCar 的关于移动部分规约描述和逻辑命题定义如下:

- Do `any_region` if and only if you are sensing `find_casualty`;
- $\Box(\Diamond \text{find_casualty} \leftrightarrow \Diamond \text{any_region})$

5.3.2 协同系统实现

在本案例中, ResCap 和 ResCar 之间的协同是相互的。ResCap 对 ResCar 的唤醒基于一个单参数传递的端口, 参数是 ResCap 当前的位置信息, 在 ResCar 中对于该输入端口处理函数实现基于 ROS Navigation Stack 的导航移动。ResCar 对 ResCap 的唤醒则基于一个简单唤醒端口, 在 ResCap 中对于该输入端口处理函数负责修改 ResCap 中的 `ground_casualty` 感知信息命题赋值。

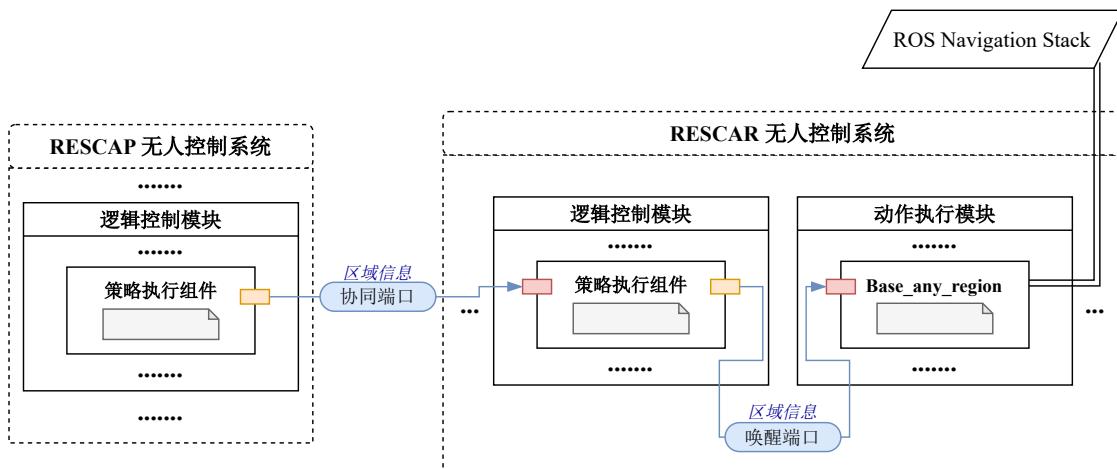


图 5-10 多机无人控制系统协同实现架构

其中,救援车导航功能是基于 turtlebot3 navigation 功能包完成的,只要提供预先使用 gmapping 绘制好的高精度地图,就能使用该导航包实现移动避障的路径规划。在本案例中目标信息通过 move_base 提供的话题接口 “/move_base_simple/goal” 发布。

5.3.3 仿真结果与分析

经过实验测试,当搜救巡逻机 ResCap 发现伤员时,救援车 ResCar 会即时寻找出一条通往 ResCap 所在位置的最优路径(见仿真演示截图5-11)。同时,伤员

转移后，ResCar 会返回 left 区域，而 ResCap 会继续巡逻。实验证明了该多机协同无人控制系统能够合理有效地进行逻辑控制。也证明了本文提出的无人控制系统在多机器人场景下的适用性以及正确性。

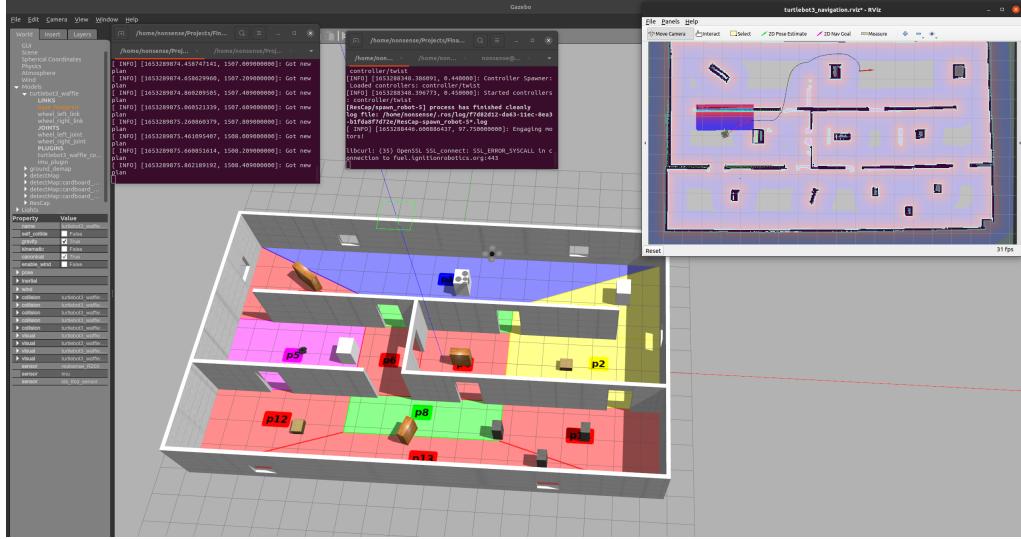


图 5-11 ResCar 导航移动至 ResCap 区域转移伤员

5.4 本章小结

本章实现了对前文提出的无人控制系统生成方法的仿真应用，从仿真环境搭建、到基于单机器人和多机器人协同场景下的仿真实现，本章验证了无人控制系统在反应式系统综合、尤其是人机物协同任务综合中的有效性、可行性以及具有较高的应用价值。

第 6 章 总结与展望

本章对论文的主体研究工作进行总结分析，并且就论文以及无人系统反应式综合领域的相关后续研究进行展望。

6.1 主体工作总结

人机物融合系统是当今计算机科学应用领域的一个重要研究方向，其包含的庞大物力资源和计算构件决定了人机物融合系统相较于经典无人系统在规模和交互方式方面都具有更高的复杂度。本论文基于人机物协同场景，使用反应式综合和基于模型的软件框架自动生成方法简化了无人控制系统的开发过程。同时，系统符合组件驱动的 F Prime 架构，能够将复杂的人机物融合行为划分至粒度更小、相对独立的组件单元中。具体而言，论文的具体工作主要包括以下三个方面：

1) 设计实现了基于 F Prime 框架的无人控制系统架构。该系统作为 LTLMoP 反应式综合以及 F Prime 框架的接口，定义了以策略自动机作为输入的机器人逻辑控制处理过程。论文充分利用了 F Prime 基于端口的组件间通信机制，将无人控制系统分层设计，并且针对不同的人机物融合场景，分别实现了主逻辑控制模块和全局控制模块的通用模型，以及动作执行模块的各种类型实现框架。

2) 提出了基于规约的无人控制系统代码生成与演化方法。论文将反应式综合 (GR(1) 博弈求解) 和基于模型的软件框架自动生成方法相结合，可以从高层次的形式化规约描述直接生成单机器人无人控制系统。同时，论文基于具体的案例详细提出了不同类型的多机协同实现方法。反应式综合用于单机器人逻辑控制，多机协同机制则由 F Prime 架构下的端口实现定义。

3) 完成了无人控制系统自动生成的软件原型开发以及基于 ROS 的仿真验证。为了简化用户开发流程，论文基于 LTLMoP 扩展开发了 F Prime 无人控制系统生成相关功能实现，同时，论文使用主流的仿真环境 Gazebo 以及 RViz 对具有代表性的单机行为以及协同场景成功进行了仿真实验，验证了论文工作的设计应用价值以及正确性。

在研究过程中，论文创新性地将经典的反应式综合理论和优秀的无人飞行器开源框架结合在一起，最终得到的反应式系统既具备理论完备性和安全性（规约可综合是对系统安全性的一定保证），又充分结合了基于高内聚、低耦合的软件工程开发思想以及可重用、可扩展的框架优势，规范化了人机物协同场景下的反应式系统综合应用。

6.2 未来研究展望

根据本文研究内容和具体介绍，在本文的工作以及本领域先关工作都还有很多亟待突破的研究方向，具体来说：对于本文的研究工作，还有以下三个方面可以进一步优化：

1) 协同实现的自动生成。对于多机协同场景，本文提出的生成方法没有涉及到系统协同行为的自动完成，协同行为往往还是由用户在 F Prime 架构下进行自定义补充。后续工作可以围绕将协同行为和协同通信命题结合起来，在规约中直接实现协同行为的定义。这一部分的研究对 GR(1) 博弈求解提出了更高的理论认识要求，需要对求解过程进行适当扩充以满足协同实现。

2) 工具原型开发。后续工作可以基于现有的工具原型，让 F Prime 相关端口和行为组件的开发过程也在工具中完成实现，并且对不同的行为组件类型，直接提供开发模板和已有的组件库。这对于低代码开发具有重要实际意义。

3) 仿真环境与实际机器人相连。本文提出的方法和模型在贴近于实际的仿真环境 Gazebo 中完成了实验验证，但没有落实到真实物理环境中的无人机和无人车上。无人控制系统的实机部署也是一个值得研究的未来发展方向。

对于反应式综合研究领域，基于本课题在研究中的思考提出以下两个方面的未来研究展望：

1) 规约演化方法的优化。现有的反应式综合一个最大的挑战就是用户规约的完备性，规约很难使得无人系统对复杂多变的系统环境都能做出正确的举动，某些环境的变化可能会导致与预期相反的结果。可以考虑提出具有一般性的意外事件处理机制以及规约重综合方法。

2) 基于协同的实时 GR(1) 综合研究，已有的反应式综合都是在运行前就给出了规约的综合策略，对于现实世界中的无人系统，场景中的机器人和环境往往是在不断变化的，尤其是新机器人的加入与旧机器人的退出，如何实时地对基于协同通信的 GR(1) 综合进行实现也是一个具有挑战性的研究方向。

致 谢

落笔于此，论文已至尾声，四年的大学生活也即将迎来句点。再回首，这一路跋涉至此，有很多人、很多事共同成全了现在的我。

首先，感谢我的家人。父亲、母亲、姐姐、我，很普通的一个四口之家，却是我人生中最宝贵的财富。感谢他们的一路陪伴和默默支持，他们是我前进的动力，也是我永远的牵挂。

感谢我的指导导师董威教授。董老师是软件工程领域的顶尖专家，从大三有幸进入董老师课题组以来，我深切体会到了董老师在治学上的严谨性以及对领域前沿研究的敏锐性，同时也感受到了董老师身为导师对学员的体贴关怀。董老师在我升学路上也给予了莫大的帮助，在此谨向董老师致以诚挚的谢意和崇高的敬意。

感谢我的“三任”室友。能够认识他们这群有趣的灵魂是我大学生活的一大幸事，从他们身上我学到了很多东西，也获得了很多帮助。祝大家在奔赴各自前程的道路上风雨无阻，披星戴月。

感谢各位师兄师姐对我的指导帮助，课题组的史浩、杨栋师兄等都给我的毕设工作提过建议和实现思路，每次组会交流都能让我收获新知识和新想法。

感谢计算机学院学院一队这个大集体和彭国栋教导员等领导干部们，能够成为这个集体当中的一员我非常开心，感谢大家的共同努力和队干部的辛勤付出，我会永远铭记这个独特的集体。

感谢所有相遇。大学四年，认识了很多兢兢业业的任课老师，结识了很多朋友、同学，每一段遇见共同构成了我大学四年独一无二的风景。

感谢音乐和羽毛球。贯穿大学四年的音乐和大四拾起的羽毛球为我的生活注入了活力。感谢让我邂逅乐队和羽毛球的她，她让我想要成为更优秀的自己。

感谢图书馆昏黄灯光下的坚持，感谢深夜屏幕微光前的执着，感谢每一份孤独，感谢每一份感动，感谢每一份温暖。

人生就是白纸，时光流逝，白纸变成了故事。我会将这四年的故事珍藏于心，永远对明天保持期待，永远对曾经心存感恩，永远年轻，永远热泪盈眶。

参考文献

- [1] Bousdekis A, Apostolou D, Mentzas G. A human cyber physical system framework for operator 4.0–artificial intelligence symbiosis [J]. Manufacturing letters. 2020, 25: 10–15.
- [2] Piterman N, Pnueli A, Sa’ar Y. Synthesis of reactive (1) designs [C]. International Workshop on Verification, Model Checking, and Abstract Interpretation. 2006: 364–380.
- [3] Finucane C, Jing G, Kress-Gazit H. LTLMoP: Experimenting with language, temporal logic and robot control [C]. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. 2010: 1988–1993.
- [4] Bocchino R, Canham T, Watney G, et al. F Prime: an open-source framework for small-scale flight software systems [J]. 2018.
- [5] Phan D, Yang J, Grosu R, et al. Collision avoidance for mobile robots with limited sensing and limited information about moving obstacles [J]. Formal Methods in System Design. 2017, 51 (1): 62–86.
- [6] LeSage J R, Longoria R G. Mission feasibility assessment for mobile robotic systems operating in stochastic environments [J]. Journal of Dynamic Systems, Measurement, and Control. 2015, 137 (3).
- [7] Gulwani S, Polozov O, Singh R, et al. Program synthesis [J]. Foundations and Trends® in Programming Languages. 2017, 4 (1-2): 1–119.
- [8] Gulwani S. Automating string processing in spreadsheets using input-output examples [J]. ACM Sigplan Notices. 2011, 46 (1): 317–330.
- [9] Balog M, Gaunt A L, Brockschmidt M, et al. Deepcoder: Learning to write programs [J]. arXiv preprint arXiv:1611.01989. 2016.
- [10] Nye M, Solar-Lezama A, Tenenbaum J, et al. Learning compositional rules via neural program synthesis [J]. Advances in Neural Information Processing Systems. 2020, 33: 10832–10842.
- [11] Church A. Logic, arithmetic, and automata [J]. Journal of Symbolic Logic. 1964, 29 (4).
- [12] Büchi J R, Landweber L H. Solving sequential conditions by finite-state strategies [C]. The Collected Works of J. Richard Büchi. 1990: 525–541.

- [13] Rabin M O. Automata on infinite objects and Church's problem [M]. American Mathematical Soc., 1972.
- [14] Kupermann O, Varfi M. Synthesizing distributed systems [C]. Proceedings 16th Annual IEEE Symposium on Logic in Computer Science. 2001: 389–398.
- [15] Pershin I, Pervukhin D, Ilyushin Y, et al. Design of distributed systems of hydrolithosphere processes management. A synthesis of distributed management systems [C]. IOP Conference Series: Earth and Environmental Science. 2017: 032029.
- [16] Finkbeiner B, Gieseking M, Olderog E-R. Adam: Causality-based synthesis of distributed systems [C]. International Conference on Computer Aided Verification. 2015: 433–439.
- [17] Pnueli A, Rosner R. On the synthesis of an asynchronous reactive module [C]. International Colloquium on Automata, Languages, and Programming. 1989: 652–671.
- [18] Amram G, Maoz S, Pistiner O. GR (1)*: GR (1) specifications extended with existential guarantees [C]. International Symposium on Formal Methods. 2019: 83–100.
- [19] Ehlers R, Raman V. Slugs: Extensible gr (1) synthesis [C]. International Conference on Computer Aided Verification. 2016: 333–339.
- [20] Maoz S, Shevrin I. Just-in-time reactive synthesis [C]. Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. 2020: 635–646.
- [21] Oura R, Ushio T. Learning-based Bounded Synthesis for Semi-MDPs with LTL Specifications [J]. IEEE Control Systems Letters. 2022.
- [22] 姚锡凡, 练肇通, 杨屹, et al. 智慧制造——面向未来互联网的人机物协同制造新模式 [J]. 计算机集成制造系统. 2014, 20 (6): 9.
- [23] Wittenberg C. Human-CPS Interaction-requirements and human-machine interaction methods for the Industry 4.0 [J]. IFAC-PapersOnLine. 2016, 49 (19): 420–425.
- [24] Huang G, Mei H, Cao D G. 从软件研究者的视角认识“软件定义” [J]. 2015.
- [25] Hu F, Hao Q, Bao K. A survey on software-defined network and openflow: From concept to implementation [J]. IEEE Communications Surveys & Tutorials. 2014, 16 (4): 2181–2206.
- [26] Bloem R, Jobstmann B, Piterman N, et al. Synthesis of reactive (1) designs [J]. Journal of Computer and System Sciences. 2012, 78 (3): 911–938.

- [27] Kozen D. Results on the Propositional Mu-Calculus. *Theoretical Computer Science*, v. 27, n. 3 [J]. 1983.
- [28] Kesten Y, Piterman N, Pnueli A. Bridging the gap between fair simulation and trace inclusion [J]. *Information and Computation*. 2005, 200 (1): 35–61.
- [29] Kress-Gazit H, Fainekos G E, Pappas G J. Temporal-logic-based reactive mission and motion planning [J]. *IEEE transactions on robotics*. 2009, 25 (6): 1370–1381.
- [30] Fernández J, Tóth B, Cánovas L, et al. A practical algorithm for decomposing polygonal domains into convex polygons by diagonals [J]. *Top.* 2008, 16 (2): 367–387.
- [31] Pnueli A, Sa’ar Y, Zuck L D. JTLV: A framework for developing verification algorithms [C]. International Conference on Computer Aided Verification. 2010: 171–174.
- [32] Liu J, Ozay N, Topcu U, et al. Synthesis of reactive switching protocols from temporal logic specifications [J]. *IEEE Transactions on Automatic Control*. 2013, 58 (7): 1771–1785.
- [33] Maoz S, Ringert J O. Spectra: a specification language for reactive systems [J]. *Software and Systems Modeling*. 2021, 20 (5): 1553–1586.
- [34] Stanford Artificial Intelligence Laboratory et al. Robotic Operating System [EB/OL]. 2018. <https://www.ros.org>[accessed May 20, 2022].
- [35] Koenig N, Howard A. Design and use paradigms for gazebo, an open-source multi-robot simulator [C]. 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566): 2149–2154.
- [36] Kam H R, Lee S-H, Park T, et al. Rviz: a toolkit for real domain data visualization [J]. *Telecommunication Systems*. 2015, 60 (2): 337–345.
- [37] Gill J S. Navigation tutorials: Robotsetup [EB/OL]. 2018. <http://wiki.ros.org/navigation/Tutorials/RobotSetup>[accessed May 20, 2022].
- [38] Fox D, Burgard W, Thrun S. The dynamic window approach to collision avoidance [J]. *IEEE Robotics & Automation Magazine*. 1997, 4 (1): 23–33.
- [39] Meyer J, Sendobry A, Kohlbrecher S, et al. Comprehensive Simulation of Quadrotor UAVs using ROS and Gazebo [C]. 3rd Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR). 2012: to appear.
- [40] Amsters R, Slaets P. Turtlebot 3 as a robotics education platform [C]. International Conference on Robotics in Education (RiE). 2019: 170–181.