Behavior Trees and State Machines in Robotics Applications

Razan Ghzouli, Thorsten Berger, Einar Broch Johnsen, Andrzej Wasowski and Swaib Dragule

Abstract—自主机器人通过组合多个技能来形成日益复杂的行为，称为任务。尽管技能通常在相对较低的抽象层级上进行编程，其协调部分在体系结构上是分离的，并且通常采用更高层级的语言或框架来表达。多年来，state machines 一直是行为建模的首选语言，但近年来，behavior trees 在机器人研究者中获得了越来越多的关注。behavior trees 最初是为建模电子游戏中的自主行为者而设计的，提供了一种可扩展的基于树的任务表示方式，并被认为支持模块化设计与代码复用。尽管已经有多个 behavior trees 实现被使用，但关于它们在现实世界中的使用情况与适用范围仍知之甚少。由 behavior trees 提供的概念与传统语言（如状态机）之间的关系如何？behavior trees 和 state machines 中的概念在实际应用中是如何使用的？

本文对 behavior trees 中的关键语言概念进行了研究，这些概念由内部或外部 领域特定语言（DSLs）库提供实现，并分析了它们在由 Robot Operating System (ROS) 支持的开源机器人应用中的使用情况。我们分析了行为树的 DSLs，并将它们与机器人行为建模的标准语言——state machines 进行了比较。我们识别出这两种行为建模语言的 DSL，并深入分析了其中五种。我们从开源代码库中挖掘使用上述 DSLs的机器人应用，并对其使用情况进行了分析。我们识别出 behavior trees 与 state machines 在语言设计方面的相似性，以及它们为满足机器人领域需求所提供的语言概念。我们观察到在开源项目中，行为树 DSL 的使用正在快速增长。我们还观察到在模型结构与代码复用层面，行为树模型与状态机模型在挖掘出的开源项目中表现出类似的使用模式。我们将所有提取出的模型作为数据集贡献出来，希望能够激发社区进一步使用和开发 behavior trees 、相关工具与分析技术。

Index Terms—behavior trees, state machines, robotics applications, usage patterns, empirical study

1 Introduction

机器人时代正在到来！它们可以在不适合人类进入的环境中执行任务，例如在危险区域进行灭火，或在受污染的医院中进行消毒。机器人能够处理日益复杂的任务，从拾取与放置操作到在动态环境中导航时执行复杂服务。机器人通过组合多个技能来形成复杂的行为，这些行为被称为任务 [1], [2]。尽管技能通常在相对较低的抽象层级上进行编程（例如传感器和执行器的控制器），但将这些技能协调成完整任务的过程要么在与技能实现紧密耦合的低层次上进行编程，要么采用行为模型进行更高层次的表达。随着机器人系统复杂性的提升，任务的高层表示在提升软件质量与可维护性方面变得愈发重要 [3], [4], [5]。

State machines 是描述机器人任务中行为模型最常见的表示形式之一 [1], [6], [7]。近年来，behavior trees 作为表达此类高层协调的方式，正受到机器人研究者的关注。这两种模型都用于描述具有有限决策能力的预定义任务。behavior trees 最初被发明用于建模电子游戏中自主非玩家角色的行为。与自主机器人类似，这些角色也具有反应性，并在复杂环境中做出决策 [8], [9], [10]。

许多研究者观察到，behavior trees 在建模反应式行为时具备易复用性、模块化和灵活性等优势 [6], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20], [21]。这些研究主要关注行为树模型的理论层面和概念验证。机器人社区中也正出现越来越多为机器人系统开发软件的领域特定语言（DSL）和模型驱动工具 [22], [23], [24]，但我们仍需更深入理解与改进这些解决方案，以提升其实用性 [23]。尽管已有多种 DSL 被用于支持行为树和状态机模型的实现，但我们仍缺乏从软件工程视角出发的研究，来理解这些实现以及它们在现实项目中的使用方式。

本研究弥补了这一知识空白，我们通过分析行为模型 DSL 的实际实现以及其在实践中的使用情况来探讨其关键概念。我们将机器人领域中新兴语言 behavior trees 的 DSL 与机器人研究者传统选择的 state machines 进行了比较。本文考察了 behavior trees 与 state machines 在五种 DSL（由库提供的内部或外部 DSL）中的实现概念，并进行了对比。我们的比较聚焦于行为树 DSL 及其所提供的语言概念。我们进一步分析了这些语言概念在开源 ROS 项目中用户的实际使用方式。我们的目标是获取这两种行为建模语言在实际应用中使用情况的经验数据。

Goal and Research Questions

我们对行为树与状态机语言及其在开源机器人应用中的使用进行了探索性研究。尽管研究以定性分析为主，我们也提供了关于挖掘模型的定量数据，以及在实践中观察到的不同概念和现象的出现频率。我们提出了以下研究问题：

RQ1. 在语言实现（库）中，*behavior trees* 与 *state machines* 所提供的建模概念有哪些？

我们识别了五种支持 ROS [25] 的 behavior trees 和 state machines DSLs（库）——ROS 是一个用于开发机器人应用的中间件和框架，拥有目前最大的机器人库生态（ROS packages）——这些库都在积极维护且有良好文档。随后，我们识别并分析了这些 DSL 所提供的建模概念。我们的目标是理解在实际中可用的行为树与状态机建模概念，并比较它们之间的异同。

RQ2. 这些语言（库）在实践中是如何被工程实现的？

对于已识别的 行为树和 状态机 DSLs，我们研究了其关键的设计原则与语言实现。通过分析这些语言的实现，我们捕捉了机器人社区中使用的技术与实践。通过研究这些起源于机器人研究者实践的语言，我们能够更好地理解机器人领域所需的设计概念，并探讨潜在的改进方向。

RQ3. 行为树与状态机模型在机器人项目中是如何使用的？

我们在 GitHub 上挖掘了使用已识别行为树和状态机 DSLs 的开源代码库，并分析了它们在机器人应用中的使用情况。我们检查了这些语言的使用趋势，以反映其受欢迎程度。我们

从挖掘出的项目中提取了一个样本，并分析了这些项目中行为树和状态机概念的使用情况与模型结构。最后，我们结合可视化和代码层级的分析方法研究了项目中的复用机制。综上，我们报告了关于在开源 ROS 机器人项目中 behavior trees 与 state machines 使用情况的实证结果，包括 DSLs 的流行度、模型结构、已识别概念的使用频率以及行为树与状态机 DSLs 中的实际复用状态。

Journal Extension

本文扩展了在 SLE 2020 [26] 上发表的关于 behavior trees 的研究。在本文中，我们将研究范围扩展至 state machines 。我们提取了状态机建模概念，并将其与前期工作中提取的行为树概念进行了对比。我们更新了 开源 行为树模型的集合，并额外挖掘了使用状态机 DSLs 的 开源 机器人项目。我们改进了项目筛选流程（过滤）并通过脚本实现了自动化处理。新的筛选流程应用于所有挖掘出的项目。最后，我们定性与定量地分析了一组与先前行为树模型数量相匹配的状态机模型随机样本，以理解所研究的行为建模语言在开源项目中的使用方式。因此，与前一篇论文相比，我们的关注点已从仅研究 behavior trees 转向包括与 state machines 在建模概念与机器人应用层面的比较。

Results

我们对 behavior tree 与 state machine DSLs 所提供建模概念的分析（**RQ1**）显示，在语言设计与提供的概念方面存在相似性。这些语言是开放的，支持领域特定的模式，反映了机器人领域的通用需求。开放性是所研究的 DSLs 中的共同特征，它们不强制固定的模型结构，而是允许甚至期望具体项目按需扩展 DSL。另一个观察结果是，所有所研究的 DSLs 都提供了对常见控制流模式的建构支持。尽管行为树与状态机 DSLs 对这些模式的支持范围有所不同，但能够满足特定领域用户需求的语言设计本身就是一种良好的实践。

我们对 DSL 设计的分析（**RQ2**）发现，提供用于模型构建与监视的可视化工具能更好地理解模型并促进代码复用。在提供图形表示与基于 GUI 编辑器的 DSLs 中，项目更常使用内建语言构件，例如 behavior trees 中的 Decorators 和 state machines 中的并发容器（在 Section 4 中介绍）。而在无 GUI 的 DSLs 中，这些构件更多通过代码体现而不是通过模型。这与 GUI 缺失的 DSLs 通常被视为内部 DSL 密切相关，而带 GUI 的 DSL 是外部 DSL。外部 DSL 强制使用其语言构件（如 decorators），而内部 DSL 则更容易偏离并使用普通编程语言构造。尽管这在实际中是务实的做法，但长期来看，内部 DSL 会削弱行为模型的可维护性与可分析性。另一个观察结果是，所有 DSLs 都遵循 *models-at-runtime* 范式 [27], [28]：模型协调技能、动作与任务，而这些则由更低层次（如 ROS 组件）实现。

我们对使用行为树与状态机 DSLs 的机器人项目所采样模型的分析（**RQ3**）揭示了这些语言在实践中的使用情况。首先，behavior trees 的使用在 开源机器人项目中正迅速增长。其次，从结构角度来看，开发者对 DSL 的使用是相似的。开发者保持模型结构相对简单；在多数 采样 项目中，无论是 behavior trees 还是 state machines ，模型都较浅且规模适中。根据我们对 采样 模型的分析经验，保持行为模型的简洁性有助于其可理解性。最后，我们在采样模型中观察到三种代码复用模式，这些复用模式的使用方式存在相似性。为了复用一个技能（动作），主要的复用模式是跨模型引用；为了复用一个任务（由多个动作组成的子树或状态机），主要的复用模式是克隆-持有（clone-and-own）。

Perspectives

通过本文，我们希望能激发两个研究社区——软件语言工程与软件建模——对机器人行为语言的兴趣。我们希望对当前实践状态的观察能促进改进。我们也希望本研究能够启发行为树与状态机语言的设计者重新审视，或至少正当化其设计选择。此外，从彼此的工具中借鉴改进点似乎是有益的，因为其中一些工具是基于模型驱动设计理念和良好的语言设计原则构建的。

最后，我们贡献了一份 开源 行为模型的数据集，以激励社区使用并进一步发展这些语言、相关工具与分析技术。附带的**在线附录 [29]** 包含了这些模型、我们的挖掘与分析脚本，以及更多细节。

2 Background

在机器人社区中，有多种控制结构被用于协调智能体的行为，包括 behavior trees 、 state machines 、 teleo-reactive 架构、subsumption 架构、顺序行为组合、流程图以及决策树 [6], [7], [30]。这些控制结构各有其优势与不足 [6], [30]。

其中许多控制结构以领域特定语言（DSLs） [31] 的形式提供给开发者使用。对于移动机器人来说，其中许多 DSL 甚至面向终端用户，提供可视化语法，正如我们在前期研究中观察到的 [32]。我们发现，大多数此类 DSL 仍处于相对较低的抽象层次，具有命令式编程的风格，类似于编程语言。有趣的是，我们还观察到，许多 DSL 是通过裁剪一种真实编程语言，并为其保留的语言概念（如表达式、声明和语句，包括机器人特定库函数调用）添加可视语法而实现的。这种可视语法通常采用 Scratch 或 Blockly 实现，提供一种简单的基于积木的编程界面。这些面向终端用户的 DSL 主要面向具备一定技术背景的用户，而其他 DSL，尤其是我们后续研究的行为树与状态机 DSL，更明确地面向开发人员。

2.1 Behavior Trees

近年来，behavior trees 已成为一种流行的行为建模语言，用于任务的规范与其控制流程的协调。已有研究表明，behavior trees 能够泛化多种控制架构 [15], [33]。传统上，state machines 因其结构简单和易于理解的控制切换机制而广泛用于机器人系统，但当任务复杂性上升时，其维护变得困难 [7], [14]。而 behavior trees 被认为具有模块化和灵活性等优势，能够克服这一问题 [6], [14], [15]。使用 behavior trees 所定义的任务可映射为 state machines ，反之亦然 [15], [17]。相比之下，本文首次对其在实际机器人项目中的使用进行了研究。

Applications of Behavior Trees

behavior trees 非常适合表达智能体的运行时行为，因此在电子游戏与机器人领域都有广泛应用。许多知名游戏（例如《Halo》 [9]）采用了 behavior trees 。然而，behavior trees 在游戏领域中的实现方式与机器人领域中的实现存在差异。例如，可能是目前世界上使用最广泛的行为树 DSL——*Unreal Engine 4 (UE4) 行为树*，更强调事件驱动的编程范式，而不是机器人中更为关键的时间触发控制机制。

在机器人社区中，对 behavior trees 的关注正持续增长。Hierarchical state-machines 曾是 ROS 导航系统中主要的任务编排机制，并被多种语言支持 [34]。随着 ROS 升级至新版 ROS 2，ROS 2 导航系统中的主要定制机制被更换为 behavior trees [35]。虽然 ROS 2 仍支持分层状态机语言 [36]，但它们已不再作为主要的定制机制。这进一步证明了 behavior trees 受欢迎程度的提升。IROS 2019，作为机器人领域的关键研究会议之一，举办了一个专门关于机器人中 behavior trees 的研讨会[1]。此外，RobMoSys——机器人领域领先的模型驱动社区之一[2]，也开展了多个项目用于建立 behavior trees 的最

1. https://behavior-trees-iros-workshop.github.io/
2. https://robmosys.eu/

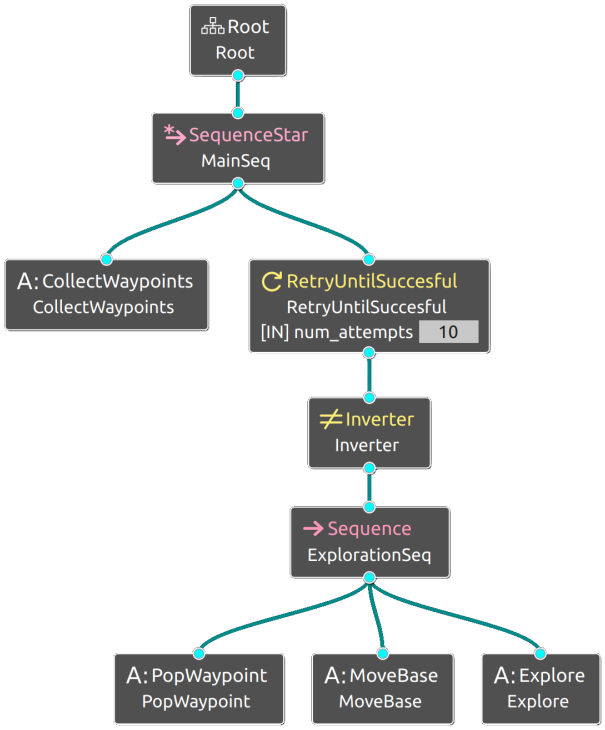

Fig. 1: 一个健康与安全机器人巡检员的行为树，来自 GitHub 项目 `kmi-robots/hans-ros-supervisor`，在 `Behavior-Tree.CPP` 的 Groot 编辑与动画工具中展示。

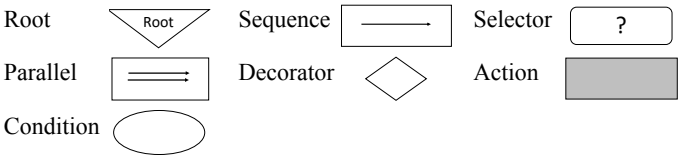

Fig. 2: 行为树中的节点类型（可视语法）

佳实践和相关工具集（例如 CARVE[3] 和 MOOD2Be[4]）。欧盟项目 Co4Robots[5] 基于行为树概念开发了一种多机器人任务规范 DSL [20], [37]。最后，behavior trees 也被用于自动驾驶系统中。Autoware [38]，一个领先的开源自动驾驶平台，采用 behavior trees 来协调其支持的预定义驾驶场景[6]。CARLA [39]，一个用于自动驾驶研究的高水平开源仿真器，使用我们本文研究的行为树 DSL 之一——`PyTrees`，来定义非自车（non-ego vehicle）的行为 [40]。

Behavior-Trees Example

如 Figure 1 所示，展示了来自 Knowledge Media Institute[7] 的一款健康与安全检查机器人行为树模型示例。该机器人在某一区域执行探索任务。主要操作位于最下方 `ExplorationSeq` 子树中，依次执行获取下一个航点、移动移动底盘至航点、探索该区域的操作。如果获取新航点失败（即栈为空），第一个任务将失败，但通过 `Inverter` 节点转换为成功，从而表示整个运动序列已完成。否则，机器人将重复执行相同操作（下一个航点、移动、探索）最多十次，前提是栈未空。整个计算过程位于一个循环中，该循环在获取新航点与执行探索序列（`MainSeq`）之间交替，直到所有子任务均成功完成。

3. https://carve-robmosys.github.io/
4. https://robmosys.eu/mood2be
5. http://www.co4robots.eu
6. https://autowarefoundation.github.io/autoware.universe/main/planning/behavior_path_planner/#behavior-tree
7. http://kmi.open.ac.uk/

Behavior-Tree Concepts

一般而言，behavior trees 可视为图形模型，其结构为有向树，拥有专用根节点、称为*控制流节点*的非叶节点，以及称为*执行节点*的叶节点。行为树的执行由根节点开始发出称为 *tick* 的信号，按照控制流节点的语义遍历整棵树。Tick 以一定频率触发 [6], [41]。节点接收到 tick 后会执行任务，该任务可为控制流任务，若是叶节点被 tick，则执行具体的机器人任务（即技能）。这些叶节点可分为动作（例如 Fig. 1 中的 `MoveBase`）和条件节点（用于测试命题，例如机器人是否位于基地），用于控制任务执行流程。被 tick 的节点会向其父节点返回以下状态之一：(1) *成功*：任务成功完成，(2) *失败*：任务执行失败，(3) *运行中*：任务仍在执行中。

上述执行语义具有一定的独特性，是 behavior trees 区别于其他行为建模语言的核心特征。我们将在后文中针对具体的行为树 DSL 展开讨论。需要指出的是，在行为树中，当前的执行状态并不会被显式表示。而在大多数其他语言中，例如状态机或流程图，状态是通过控制流元素（如转换）将当前执行点推进来更新的。例如在流程图中，一个动作执行完后由控制流导向下一个动作。值得注意的是，行为树中的节点同样是由控制流元素控制的动作。换言之，行为树本质上与许多命令式程序相似——回想我们之前研究的面向终端用户的机器人控制 DSL [32]，它们大多是带有可视语法的命令式编程语言子集，基于 Scratch 或 Blockly 表示。行为树通过在短时间间隔内重新执行整棵树，使得动作可以被重复执行。直观上，可以将更具反应性的控制逻辑放在树的左上方，由每次 tick（即控制循环）定期触发，而更具计划性的技能（动作）则位于树的底部或右侧。向其他语言（包括状态机和流程图）添加这种反应控制能力是困难的，即使是层次状态机也难以实现。换句话说，这种特殊的执行语义有助于实现模块化，并可统一其他控制架构，例如 subsumption 架构 [15]。

使用 behavior trees 的优势在于它们可以通过一小组但可扩展的控制流节点表达任务协调逻辑。大多数行为树语言提供以下控制流节点：sequence（顺序）、selector（选择）、decorator（修饰器）以及 parallel（并行）。如 Fig. 1 所示示例中，包含了两个 sequence 节点（`MainSeq` 和 `ExplorationSeq`）与两个 decorator 节点（`Inverter` 和 `RetryUntilSuccesful`）。直观上，sequence 节点会对所有子节点发出 tick，并要求所有子节点都成功才能自身成功，而 selector 节点只需其中一个子节点成功即可。Decorator 节点支持更复杂的控制流程，如 for 和 while 循环，且具备可扩展性，开发者可以自定义实现新的 decorator 节点。Parallel 节点的命名实际上具有一定误导性。它们是 sequence 和 selector 节点的泛化形式，允许设置自定义策略，例如通过设定需要成功的最小或最大子节点数量（cardinality）来控制行为。在已有文献中，关于 parallel 节点是否像其名称所暗示的那样并行执行子节点，尚无统一观点 [6], [10], [30], [41], [42], [43]。

如 Fig. 2 所示，行为树主要节点类型的可视表示在机器人与游戏中的研究中被广泛使用 [6], [41], [43]。我们在 Fig. 1 中展示了一种流行的行为树 DSL ——`Behavior-Tree.CPP` 所使用的可视语法，它也是我们本文研究的 DSL 之一。尽管不同文献与实际语言实现中节点的图形形状可能存在差异，但每种节点类型的内部符号通常保持一致，例如 selector 节点始终使用“?”作为标识。

2.2 State Machines

状态机可能是目前最流行且研究最充分的系统行为建模语言。它不仅用于建模，还常用于分析（如模型检查）和控制真实系统的软件合成。有关状态机的研究在非机器人领域已有大量文献。因此，与上文对行为树的详细应用讨论不同，本文不会展开讨论状态机的应用，而是通过一个示例将行为树映射为熟知的状态机语法，并重新概述状态机的核心概念。

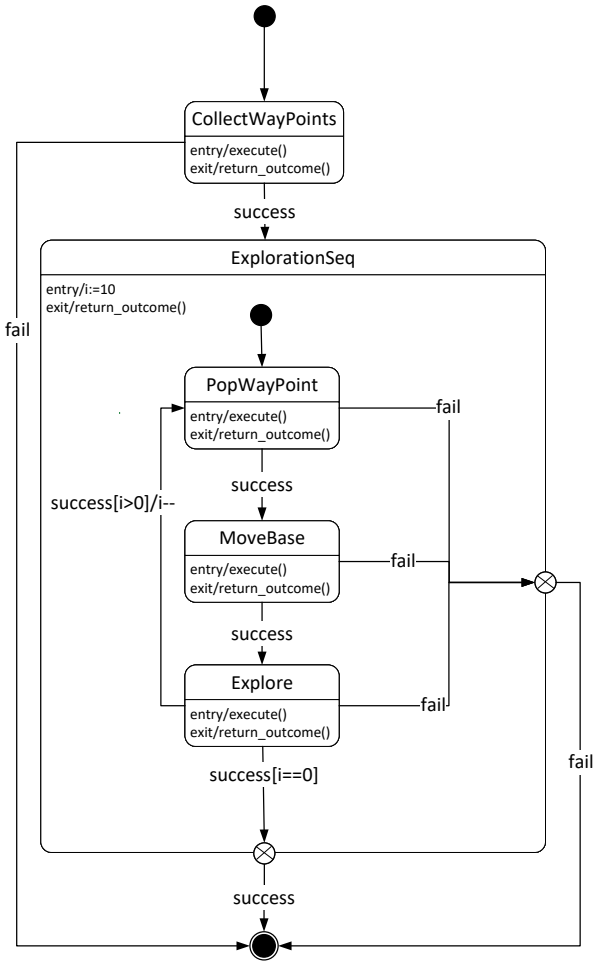

Fig. 3: 使用 UML 语法表示的状态机，与 Fig. 1 中的行为树模型描述的是相同的任务。

State-Machines Example

Figure 3 展示了与 Fig. 1 中健康与安全检查机器人相同任务的状态机表示，采用了统一建模语言（UML）的语法（UML 语法见 Fig. 4）。behavior tree 模型中的 `ExplorationSeq` 被置于嵌套（也称为复合）状态 `Exploration_seq` 中。在 `Exploration_seq` 的入口处，将变量 `i` 设置为尝试次数上限(此处为十次)。当 `Explore` 状态执行成功且尝试次数达到上限时（触发事件为 `success`，守卫条件为 `i==0`），模型将从 `Exploration_seq` 退出。若 `Explore` 状态触发 `success` 事件且 `i>0`，则在从 `Explore` 的转换中执行自减操作 `i--`。

State-Machine Concepts

state machines 与分层状态机（hierarchical state-machines ）已在多个领域应用数十年，其语法与语义存在细微变体 [44], [45]。尽管语法存在差异，UML 的状态图（state diagram）可能是软件建模社区中最常见的状态机可视语法（见 Fig. 4）。

状态机模型是由三类基本元素构成的有向图：状态（state）、转换（transition）与动作（action）。根据 UML 语义 [44], [46], [47]，状态表示某类行为模式，机器人所执行的行为体现在动作中。从计算语义角度来看，UML 状态机支持 Mealy 和 Moore 两类执行语义 [48], [49]。当动作与转换关联时（action-on-transition），类似 Mealy 机，动作在转换过程中执行；当动作与状态关联时（action-on-state），类似 Moore 机，动作在进入或退出状态时执行 [50]。大多数状态机实现采用事件驱动方式。

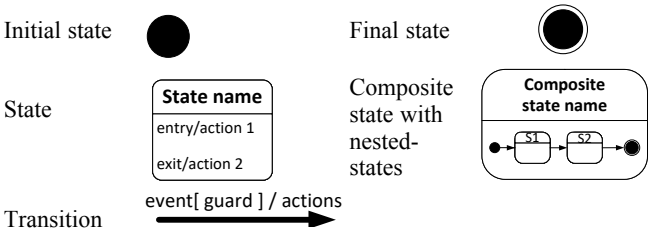

Fig. 4: 根据 UML 规范，state machines 的节点（状态）类型与转换（可视化语法）

Hierarchical state-machines ，也称为状态图（statecharts）[51]，是 state machines 的扩展，支持嵌套（层次结构）与并发执行 [51], [52]。因此，hierarchical state-machines 在保持状态机基本特性的同时，允许将行为划分为子状态或复合状态，从而增强模型模块化并减少所需的转换数量。

与机器人中的 behavior trees 类似，state machines 也以图形模型的形式表示机器人代理的行为。Colledanchise 与 Ögren [15] 指出，在机器人领域，behavior trees 可泛化 state machines 。在本文接下来的内容中，我们将聚焦于机器人领域中的实现方式；这些机器人相关的 state machines 实现将与 UML 语法略有差异。

3 Methodology

本节将介绍我们用于识别 behavior trees 与 state machines 的相关 DSL 及其所提供的建模概念（RQ1）、分析其实现方式（RQ2），以及识别与分析使用这些 DSLs 的开源机器人项目的方法（RQ3）。

3.1 Identifying Languages and Concepts (RQ1)

为了识别行为树 DSL，我们在 GitHub 上以“behavior tree robotics”、“behavior tree robot”等关键词搜索机器人相关项目。搜索结果中包含多个采用不同行为树 DSL 的项目，我们识别出其中导入的 DSL，并重点关注 Python 和 C++ 两种最常用于机器人开发的编程语言。此外，我们还查阅了机器人行为树相关文献以识别更多 DSL。

对于状态机 DSL 的搜索，在 GitHub 上得到的结果数量巨大，因此我们转而使用 ROS[8] wiki 平台来寻找状态机 DSLs。该平台是开发者发布其开源机器人语言与工具的常用平台。通过在 ROS wiki 上而非 GitHub 或 Google 搜索，我们确保识别出的语言支持 ROS，并在后续步骤中排除了非机器人项目。

为了确保所识别的 behavior trees 与 state machines DSL（以库形式提供）在真实机器人应用中的相关性，我们采用以下排除标准来筛选：（1）缺乏文档；（2）项目不再维护（最后一次提交早于 2019 年）；（3）不支持 ROS（特别针对通过 GitHub 收集的 behavior tree DSLs 进行检查）；（4）未在挖掘的项目中被使用。

随后，为了理解所包含 DSLs 的主要特性与建模概念，我们识别其核心语言构件并分析这些构件之间的关系。我们通过探索性文献调研 [6], [30], [43] 和雪球式引用收集行为树概念。对于已识别的 库，我们查阅其文档 [53], [54]，并编写脚本执行小型测试以更深入地理解其语义。

对状态机概念的收集采用类似方法，基于相关文献 [55], [56], [57], [58] 以及库的官方文档 [59], [60] 进行分析。在此过程中，我们特别关注行为树的概念，以及状态机是否直接支持类似概念，抑或需通过间接方式表达。我们采用迭代分析方法，确保概念在不同 DSLs 中得到恰当体现。我们的比较范围限制在满足筛选条件的 DSL 所提供的建模概念之内。

8. http://wiki.ros.org/

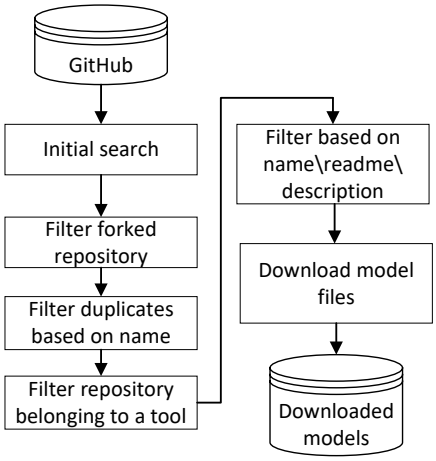

Fig. 5: 用于识别相关代码库的筛选步骤。

3.2 Language Implementations (RQ2)

在识别出支持状态机与行为树建模语言的相关库，并分析其语法与语义之后，我们进一步探究这些库的实现设计。我们的研究结果来源于多种信息源，具体包括 GitHub 上的库实现、相关文档 [53], [54], [59], [60], [61]、教学教程 [62], [63], [64], [65]，以及相关研究文献 [55], [56], [58], [66], [67], [68], [69]。

我们重点考察了库的语言设计与所提供的具体语法，并通过学习教程与文献了解其动态性特征。此处的“动态性”指的是运行时修改模型的能力。最后，我们还通过查阅实现与文档，分析各库所采用的并发建模机制。

3.3 Identifying Language Projects and Analysis (RQ3)

为了理解所识别 DSLs 在机器人项目中的使用情况：(1) 我们在 GitHub 上挖掘使用这些语言进行机器人任务建模的开源代码库；(2) 比较这些 DSLs 在开源项目中的流行趋势，并分析其中一个样本项目集在语言概念使用与模型结构方面的具体表现；(3) 观察 DSLs 用户在实际中是如何实现代码复用的。以下部分详细说明了这些步骤。

Mining GitHub

开源项目在软件工程与机器人研究社区中被广泛用于理解实际应用情境。GitHub 等开源平台为研究人员提供了极大便利，使他们能够识别机器人与软件工程领域中的主流实践 [70], [71], [72]。受到我们前期工作 [26] 以及其他在 GitHub 挖掘领域工作的启发，我们选择使用 GitHub 作为开源机器人项目的挖掘来源。

针对已识别的行为树与状态机 DSL，我们研究了这些语言在机器人项目源码中的使用方式。在 `Behavior-Tree.CPP` 中，术语 *main_tree_to_execute* 表示 XML 源码中的树入口节点；而在 `PyTrees_ros` 、`SMACH` 与 `FlexBe` 中，分别通过 *py_trees_ros*、*smach_ros* 和 *flexbe_core* 进行语言导入。这些术语必须在目标语言源码中使用。我们编写了一个 Python 脚本，利用 GitHub 的代码搜索 API（通过 PyGithub[9]）对这些术语在代码中的简单文本匹配进行挖掘。

随后，我们希望过滤掉属于课程或教程的项目，以更真实地反映这些库在机器人中的实际使用情况。参考 Malavolta 等人提出的机器人软件挖掘指南 [70]，我们在其方法基础上稍作修改以适配我们的研究目标。Figure 5 展示了我们的过滤机制。在 GitHub 中完成开源项目挖掘后（步骤 1），我们通过 Python 脚本排除 fork 项目（步骤 2），从而避免重复模型。我们注意到还有一些重复项目并非 fork，而是 clone 后重新上传的。因此，我们通过脚本提取不同 GitHub 用户中项目名称相同的仓库，手动验证是否为重复项目后将其删除 (步骤 3)。进一步检查发现，项目名中包含关键词 (tool) 的仓库往往是工具类软件，这些工具依赖目标库但并不包含具体的机器人任务实现。考虑到我们研究的是机器人的实现项目而非辅助工具，因此使用脚本排除项目名包含“tool”的仓库 (步骤 4)。我们还排除了以下组织用户的项目，这些要么是目标库作者发布的教程仓库，要么是已知工具项目，其依赖所识别语言但不属于机器人任务实现：BehaviorTree、splintered-reality、team-vigir、ros-planning、ros-infrastructure、carla-simulator。最后，我们通过脚本检查项目名称、README 和描述字段，排除包含以下关键词的仓库：(assignment、course、tutorial、introduction) (步骤 5)，确保排除课程或作业项目。经过这一系列筛选后，我们得到了符合标准的相关仓库列表。

接下来（步骤 6），我们下载包含模型定义的文件。在提取建模概念过程中（见 Section 4），我们识别出每种 DSL 中用于构建模型的特定术语。在 `Behavior-Tree.CPP` 中，*main_tree_to_execute* 表示树构建入口；在 `PyTrees_ros` 中，*add_child* 通常用于添加节点；在 `SMACH` 与 `FlexBe` 中，分别通过 *StateMachine.add* 和 *OperatableStateMachine.add* 添加状态。我们在挖掘的 GitHub 仓库中匹配这些术语所在文件并下载。所有上述步骤均基于 Python 正则表达式[10]、Requests API[11] 以及 GitHub 的代码搜索 API 实现。所有代码可在我们的在线附录中获取 [29]。

Analyzing Models

为理解 behavior trees 与 state machines 在开源机器人项目中的使用情况 (RQ3)，我们从两个角度分析所挖掘的项目：DSL 的流行趋势与模型结构。首先，我们提取每个仓库的创建时间与最后提交时间，以绘制年度活跃项目数量曲线。我们认为一个项目在其创建日至最后提交日之间均为活跃。我们使用全部挖掘项目 (在过滤前)，仅排除以下语言组织/作者发布的项目：BehaviorTree、team-vigir、pschillinger、FlexBE、splintered-reality、ros、ros-visualization。数据收集截止至 2021 年 12 月 31 日。

在模型分析方面，我们关注 behavior trees 与 state machines 在实际机器人项目中的使用方式。我们使用筛选步骤中获得的相关仓库及其下载的模型文件。分析行为树模型时，我们获取了 75 个模型。尽管 2022 年初进行的更新挖掘带来了新模型与项目，但由于原样本已覆盖不同规模与领域，我们保留了原样本。为保证可比性，我们从状态机模型中随机抽取 75 个模型。

由于状态机挖掘获得了成千上万个模型，因此我们按模型大小 (即节点数量，见下文指标) 划分两个样本池：常规规模 [2–6] 与大型规模 [7–66]，然后使用 Python 的 DataFrame 库中的抽样 API[12] 从每个样本池中随机抽取项目。各样本池的模型大小范围依据数据分布确定。总样本量与行为树模型一致，均为 75 个模型。

我们计算了若干结构性指标，用于描述模型结构与建模概念的使用情况。由于我们分析的是两种结构不同的模型 (state machines 为有向循环图，behavior trees 为有向无环树)，无法统一所有指标，因此分别计算通用指标与专用指标，并用 SM 与 BT 缩写标记：

9. https://github.com/PyGithub/PyGithub

10. https://docs.python.org/3/library/re.html
11. https://docs.python-requests.org/en/latest/
12. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sample.html

- 模型规模（BT.size, SM.size）：行为树中为除根节点外的节点总数；状态机中为状态总数。
- 树深度（BT.depth）：从根节点到最深节点的边数 [73]。例如 Fig. 1 中模型的 BT.depth 为 5。
- 平均分支因子（BT.ABF）：每个节点的平均子节点数。以 Fig. 1 为例，BT.ABF 为 1.6。
- 嵌套层级（SM.nesting）：由状态复合关系形成的嵌套层级数 [74], [75], [76]。主状态机为第一层。如 Fig. 7 中，主容器 `Inspect_SM` 为第 1 层，嵌套容器 `Iterator_10_attempts` 为第 2 层，`Exploration_seq` 为第 3 层。
- 节点类型比例（N.pct）：某节点类型在总节点中的占比，衡量 behavior trees 中复合节点与 state machines 中容器构造的使用频率。

上述指标的提取高度依赖于各 DSL 的实现，因此我们为每种语言分别设计提取方法。对于行为树 DSLs，我们参考库教程与一组随机模型样本，观察用户的实现风格，并注意到模型通常深度嵌套于其他代码中。部分指标可自动提取，其他需手动抽取模型。我们通过库文档整理出每种节点类型的函数名，再用 Python 脚本统计文本匹配次数，以提取 BT.size 与 N.pct。由于叶节点实现结构不统一，无法自动计数，BT.depth 与 BT.ABF 也需从人工提取模型中手动统计。

在提取 behavior tree 模型时，若使用外部 DSL（如 `Behavior-Tree.CPP` ），我们可利用随库附带的可视化编辑器 Groot；若为内部 DSL（如 `PyTrees_ros` ），我们需手动分析源码中调用 API 构建树的部分。凡是包含根节点的树结构，我们均视为 behavior tree 模型。

对于状态机 DSLs，我们同样参考库教程与样本项目，观察到每个库的实现模式较为统一，因此可自动提取。借助 Comby API[13]，我们为每个状态机定义语法模式，以匹配其容器的起止，从而隔离模型。随后使用 Python 脚本提取模型，并统计状态与复合状态数量，以计算 SM.size 与 N.pct。如同 behavior trees 中复合节点的计数方式，我们通过函数名匹配提取容器类型。最终我们将模型可视化，以便人工检查，使用 state-machine-cat (SMCat)[14] API 实现。由于 `FlexBe` 与 `SMACH` 自带的可视化器存在项目依赖问题，我们选择将模型提取后用 SMCat 外部可视化。通过 Python 脚本将模型转换为 SMCat 语法，生成 SVG 图像文件。SM.nesting 的提取需人工完成，我们逐一浏览图像并统计嵌套层级。所有脚本与 SVG 文件均可在我们的在线附录中获取 [29]。

Analyzing Reuse

最后我们分析了模型的代码复用情况，因为代码复用是机器人软件工程中的关键问题之一 [1], [5], [78], [79], [80], [81]。复用机制对于实现 behavior trees 与 state machines 在实际应用中的可扩展性与可持续性至关重要，因此我们分析了研究样本中实际复用的现状。本文所指“复用”包括复用已有机器人技能（动作）的代码，而非重新编写；也包括在同一模型或不同模型中复用一段组合任务（由多个技能组成）的代码。

我们结合模型图与源码分析，识别样本项目中的技能级与任务级复用。在模型与项目不同子模型中，任务通常表现为 behavior trees 中的子树，或 state machines 中的复合状态；技能则是 behavior trees 中的叶节点，或 state machines 中绑定到状态的动作。通过观察模型图，可初步识别复用模式，尤其是模型结构相似但技能组合略有不同的任务复用。随后，我们深入源码分析这些复用任务与技能的实现方式，理解其采用了哪些具体的复用机制。

13. https://github.com/comby-tools/comby
14. https://github.com/sverweij/state-machine-cat

TABLE 1: 识别出的行为树与状态机 DSLs。加粗部分为我们分析过的语言。

name	target language	ROS	doc.	last update
`Behavior-Tree.CPP` https://github.com/BehaviorTree/BehaviorTree.CPP	**C++**	**yes**	**[54]**	**2021/05**
PyTrees https://github.com/splintered-reality/py_trees	**Python**	**no**	**[53]**	**2021/05**
PyTrees_ros https://github.com/splintered-reality/py_trees_ros	**Python**	**yes**	**[61]**	**2021/05**
BT++ https://github.com/miccol/ROS-Behavior-Tree	C++	yes	[82]	2018/10
SkiROS2 https://github.com/RVMI/skiros2	Python	yes	[83]	2020/11
pi_trees https://github.com/pirobot/pi_trees	Python	yes	n/a	2017/10
Beetree https://github.com/futureneer/beetree	Python	yes	n/a	2016/03
SMACH https://github.com/ros/executive_smach	**Python**	**yes**	**[59]**	**2020/05**
FlexBe https://github.com/team-vigir/flexbe_behavior_engine	**Python**	**yes**	**[60]**	**2020/12**
SMACC https://github.com/reelrbtx/SMACC	C++	yes	[84]	2021/05
RSM https://github.com/MarcoStb1993/robot_statemachine	C++	yes	[85]	2020/03
Decision making https://github.com/cogniteam/decision_making	C++	yes	n/a	2016/07

4 语言概念（RQ1）

我们的调研共识别出 12 种用于在 ROS 中实现行为树与状态机模型的 DSL。 Table 1 的上半部分列出了所识别出的行为树DSL。我们重点分析了前三行中加粗字体的DSL，原因如下：在与机器人相关的DSL中，这三种在我们检查的时间点（2021 年 6 月 9 日）仍处于活跃开发阶段。它们共同支持用 Python 和 C++ 这两种机器人社区中最流行的编程语言实现的 ROS 系统。ROS 的最新版本 ROS 2 [86] 也得到了这些 DSL 的支持。`PyTrees` 是 Python 社区中主要的行为树DSL，它并不直接面向 ROS，而是面向通用机器人应用。一个流行的扩展 `PyTrees_ros` 为其提供了 ROS 绑定。由于 `PyTrees` 与 `PyTrees_ros` 的差别仅在于是否打包为 ROS 组件，我们决定将 `PyTrees` 纳入语言分析，尽管其本身不直接支持 ROS。虽然我们分析了 `PyTrees` 和 `PyTrees_ros` ，但在本节其余部分中，为简洁起见我们仅使用 `PyTrees_ros` 来表示它们的分析结果，因为我们的发现适用于两者。

我们决定将其余DSL排除在分析之外。BT++ 已被淘汰，其开发者在加入 `Behavior-Tree.CPP` 项目后，BT++ 被 `Behavior-Tree.CPP` 所取代。SkiROS2 是一个用于机器人任务级编程的软件平台，借鉴了 MDE 的设计理念，并以 behavior trees 作为执行引擎。SkiROS2 是 SkiROS [87], [88] 的新版本，而后者目前已被淘汰。SkiROS2 并没有开源项目可供分析，因此被排除。Beetree 和 pi_trees 是不再维护的实验项目，已被废弃。

Table 1 的下半部分列出了我们识别出的状态机DSL。我们分析了前两个 (加粗字体显示), 并排除了其余三个，因为它们符合我们的排除标准 (参见 Sect. 3.1)。SMACC 是一个持续维护的DSL，支持用 C++ 实现的 ROS 系统。Robot Statemachine

TABLE 2: 我们所研究的行为树语言与状态机语言中的关键概念对比

行为树	行为树语言	状态机语言
概念 / 方面	`PyTrees`, `PyTrees_ros`, `Behavior-Tree.CPP`	`SMACH`, `FlexBe`
编程模型	支持同步与异步，基于时间触发与活动驱动。可通过 tick 概念与子树重排序实现一定程度的反应式编程。	异步、事件触发、反应式
简单节点	执行**动作**（任意命令，可瞬时或持续），或评估**条件**（根据值返回成功/失败）。	基本状态，关联动作/条件
返回状态	每次被触发时，节点返回成功、失败或运行中状态。状态报告决定遍历是否推进到下一个节点。	每个状态返回其状态（即结果），结果由用户自定义。执行函数定期检查结果
复合节点	定义每个周期（tick）中的层次遍历与控制流程。以顺序方式组合。节点本身可触发并发代码。	类似容器概念，允许状态嵌套与控制流程构造。
根节点	每次遍历的入口点。只能有一个子节点。每个周期重新进入根节点。	初始状态（第一个加入容器的状态）。
顺序（sequence）	按顺序触发子节点，直到第一个失败为止。若无失败则返回成功。	通过预定义容器支持。
选择（selector）	按顺序触发子节点，直到第一个成功为止。若无成功则返回失败。	无直接支持，可通过容器扩展实现。
并行（parallel）	是 sequence/selector 的泛化形式，带有策略参数，例如指定成功子节点的最小数量。	无直接支持，可使用守卫条件扩展。
跳转（goto）	无通用跳转结构，遍历始终沿树结构进行。	支持跳转。
装饰器节点（decorators）		
inverter	反转子节点的成功/失败状态。	无直接支持，可通过容器扩展实现。
succeed	忽略子节点返回状态，始终返回成功。	无直接支持，可通过容器扩展实现。
repeat	重复触发子节点指定次数，全部成功则自身成功，否则失败。	可通过 `SMACH` 中的 *Iterator container* 实现 repeat-until 逻辑，循环终止条件由用户定义。
retry	若子节点失败则立即重试，最大次数后若仍失败则自身失败，否则成功。	与 repeat 概念相似，可通过修改 *Iterator container* 实现。
动态性	由于实现的动态特性，可在运行时修改模型（如节点重排）。	支持运行时修改模型（添加/移除状态，修改状态实例，重排状态）。
开放性	用户可按需自定义新节点与操作符。	使用容器概念可实现新的状态与控制流程。
并发性	支持 interleaving 与协程；通过 sequence、selector、parallel 节点声明。	支持 interleaving；通过并发容器声明；状态顺序执行，因使用 Python 线程而非真正并行。

(RSM) 是另一个支持 ROS 且带有 GUI 的 C++ DSL。由于在 GitHub 上缺少使用 SMACC 和 RSM 的开源项目，因此我们也将其排除。Decision making 虽然支持 C++ 下的 ROS 系统，但目前已不再维护。我们纳入的两个DSL在检查时（2021 年 6 月 9 日）仍处于活跃维护状态，文档完善，支持 Python 下的 ROS 系统，并在 GitHub 上有多个可用于挖掘的项目。`FlexBe` 支持 ROS2，而 `SMACH` 不支持 [36]。

接下来我们探讨这些已纳入行为树和状态机DSL所提供的语言概念，包括它们的语法与语义。Table 2 总结并比较了行为树和状态机DSL所支持的语言概念。最左列列出了与行为树DSL相关的概念，这些概念要么被包含，要么被显著地排除。最后一列简要说明了对应概念在状态机DSL中的处理方式。本节余下内容将首先讨论 behavior trees ，再讨论 state machines 。

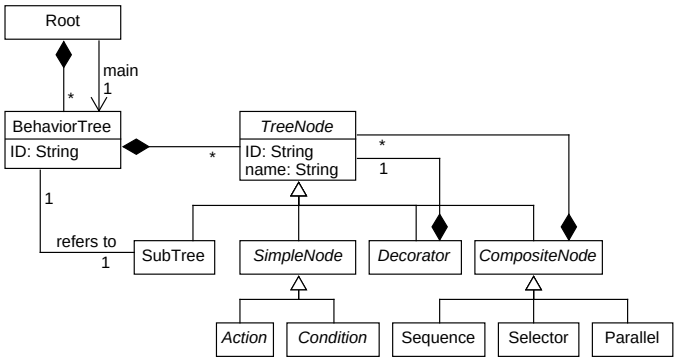

Fig. 6: 从 `Behavior-Tree.CPP` 的 XML 格式中逆向工程得到的 `Behavior-Tree.CPP` 元模型。

4.1 行为树 DSL 中的概念与语义

以下讨论基于对现有文献和 `PyTrees_ros` 与 `Behavior-Tree.CPP` 文档 [53], [54] 中关于行为树语言的广义描述。Table 3 展示了分析中的 DSL 所涉及的基本行为树概念类别。

在分析的DSL中所采用的 behavior trees 变体主要是一种*定时触发、基于活动*的行为建模语言，不同于游戏中的 behavior trees 实现（参见 Sect. 2）。计算过程由具有持续时间的活动组成，主控循环以（通常）固定的时间间隔触发整个模型的执行，就像一个电路一样。每次 tick（或 epoch）都会遍历整棵树，不同类型的节点会引入分支。遍历过程中可以启动新的活动、评估条件、访问状态并执行具有副作用的基本动作。尽管文档中反复声称支持响应式编程，但实际上响应式编程并不是一等公民，[15]不过可以通过足够高频率地执行模型来模拟响应性。

分析的 DSL 支持使用行为设计模式 [89] 中的黑板机制作为全局存储结构，黑板是一个键值对存储区。不支持作用域，

15. 例如，`PyTrees` 的文档称该语言提供了*一种在实现目标的计划性与在关键事件面前保持反应性的良好平衡*；https://py-trees.readthedocs.io/en/devel/background.html

TABLE 3: Behavior tree 概念及其在 Behavior-Tree.CPP 、PyTrees_ros 和 PyTrees 中对应的语言元素

concept	Behavior-Tree.CPP	PyTrees PyTrees_ros
Simple Node	subclasses of ActionNode ConditionNode	behaviour.Behaviour
Composite	subclasses of ControlNode	classes in composites
Sequence	Sequence, SequenceStar ReactiveSequence	composites.Sequence
Selector	Fallback, FallbackStar ReactiveFallback	composites.Selector composites.Chooser
Decorator	subclasses of DecoratorNode	classes in decorators
Parallel	ParallelNode	composites.Parallel

所有键都是全局的。黑板用于模型内部和系统之间的通信，模型与系统会异步地读写黑板内容。

Behavior-Tree.CPP 和 PyTrees_ros 都支持四种基本的控制流节点类别和两种基本的执行节点。为了展示抽象语法，我们在 Fig. 6 中提供了一个从 Behavior-Tree.CPP 的 XML 格式逆向建模得到的元模型。这些概念会在 Table 2 中进一步详细说明。

简单节点

回顾一下，简单节点（即语法树中的叶子节点）分为条件节点和动作节点。动作节点在模型中实现基础计算。我们所分析的DSL的用户需要实现自定义的动作节点——这些节点是符合 Action 接口的类，其中包含在节点被 tick 时执行的 Python 或 C++ 代码。条件节点用于计算布尔谓词的值，并将其转换为成功或失败的结果。

Behavior-Tree.CPP 支持多种类型的同步与异步动作节点，而 PyTrees_ros 主要支持同步类型。 最简单的动作节点是同步的，因此它们会迅速终止，并立即返回成功或失败状态。异步节点也可以返回“运行中（running）”的状态，并通过某种形式的并发机制持续运行。执行引擎会尝试在下一个 epoch 再次触发这些节点。

复合节点

复合节点是行为树中的内部节点，其主要功能是在每一个时间点（每次触发）定义遍历顺序。与需要由用户实现的简单节点不同，DSL 提供了一系列预定义的复合节点。*根节点*是复合节点，它作为每次遍历的入口点，包含另一个节点作为其主体。该节点在每次遍历开始时被重新进入。

我们所研究的 DSL 中，不同复合节点的语义遵循行为树相关文献的定义，如 Sect. 2 所述。*顺序*（*sequence*）节点类似于许多编程语言中的 *forall* 高阶函数。*选择器*（*selector*）节点则类似于 *exist*。我们确认，在这些实现中，所谓的 *并行*（*parallel*）节点其实并不真正并发执行，而是将 sequence 和 selector 一般化为一系列的*策略*，这些策略在各自的 DSL 文档中都有描述。

由于行为树的执行总是对整棵树的遍历，因此不支持直接的跳转（goto）。与此相对，复合节点可以局部地影响遍历方式，与 state machines 的控制流机制形成鲜明对比。在所研究的 DSL 中，控制流的典型变化可以导致任意状态的改变，常常是跨越语法树结构的，这取决于某个节点返回的状态。

在每次复合节点的 tick 之后，包括通过传播影响到的简单节点，都会显式返回一个状态。我们的 DSL 支持的状态集合与行为树文献中定义的状态一致（见 Sect. 2）。这些状态值在遍历过程中会根据复合节点的语义自下而上传播。

装饰器

装饰器在所研究的 DSL 中被实现为一元复合节点（只有一个子节点），用于修饰子树并改变其数据或控制流。Behavior-Tree.CPP 与 PyTrees_ros 提供了种类丰富的装饰器结构。*反转器*（*Inverter*）将子节点的返回状态在成功与失败之间反转。*成功器*（*Succeeder*）不论其子节点返回什么状态，始终返回成功。*重复器*（*Repeat*）节点是有状态的，类似于 for 循环：它在指定的 tick 次数内持续触发子节点。在每次触发时递增内部计数器。当达到设定次数时节点返回成功并重置计数器；如果子节点失败则节点失败并重置计数器。*重试器*（*Retry*）节点类似于 Repeat 节点，其主要目的是让一个不稳定的节点最终成功。像 Repeat 一样，它可以运行一个节点多次，但与 Repeat 不同的是，它只在节点失败时才进行重试，且不等待下一个 epoch，而是立即重试。如果在设定的尝试次数内子节点始终失败，则重试器返回失败。

> Observation 1. 行为树语言中所提供的概念及其语义来源于实际应用需求。我们所研究的行为树DSL汇集了许多基于模式的构造，这些构造据用户和开发者反馈在自主系统的高级控制中被频繁使用。

开放性

在 behavior trees 中，除基于协程的定时触发计算模型外，其开放性与非限定性可能是最有趣的特征。其他研究也在 DSL 变异性上下文中注意到了这一点 [90]。

不同于 Ecore[16] 或 UML，这些语言的元模型*不是固定的* [26]。DSL 提供了复合节点的元类，而将简单节点保留为抽象，或仅提供基本功能（参见 Fig. 6）。语言的用户需要首先扩展元模型，通过实现基础动作节点，然后将这些节点连接组成语法树，可能还需使用外部 XML 文件。这种做法与 stereotype 的概念颇为相似 [46]。当然，Ecore 的用户也可以在运行时扩展元模型类并为其添加新功能，但这种用法被视为进阶技能，并不常见。二者的区别在于程度：几乎没有不创建自定义节点就能使用行为树DSL的方式。

前置条件（用户群体）

行为树 DSL 的开放性特征意味着其建模与调试体验极度类似于建模和语言设计研究社区中的语言导向编程实践。用户需不断与元类打交道，进行组合、遍历等操作。任何有在 Ecore 或类似框架上构建 DSL 经验的人在使用 PyTrees_ros 或 Behavior-Tree.CPP 时都会产生一种 *似曾相识*（*déjà vu*）的感觉。

鉴于许多机器人工程师以及 ROS 用户缺乏计算机科学与软件工程方面的正规训练 [91]，我们对该设计能在社区中受到欢迎感到惊讶。即便在软件工程领域，语言实现与元编程技能仍被视为高级技能。然而，使用行为树DSL正需要这些技能。这对建模社区构成了挑战：如何设计一种行为树 DSL，既能灵活集成进大型复杂系统，又能让普通机器人程序员也易于使用。

> Observation 2. 所研究的行为树DSL的灵活性与可扩展性对机器人开发者提出了语言导向编程的要求。软件语言工程社区可以通过设计一种更易上手但依然灵活的行为树方言为此作出贡献。

16. https://www.eclipse.org/modeling/emf/

关注点分离

行为树DSL 支持在特定机器人系统中构建的特定平台模型(PSM)，以在运行时控制行为。模型的作用是简化并概念化行为的描述。是否能将相同模型复用于其他硬件或类似系统尚不是主要目标。所研究的行为树 DSL 不仅是 PSM，而且往往与系统紧密耦合。自定义节点通常直接引用系统元素并与系统 API 交互。因此，很难将创建的模型与机器人系统分离使用。

虽然 Groot 可以可视化一个独立 XML 文件表示的模型，但仅要可视化 `PyTrees_ros` 模型的语法，就需要一个可运行的 ROS 环境。这可能不仅包括合适的 Python 与 ROS 库的安装，还需要例如机器人的仿真环境，甚至硬件环境。你需要启动整个系统并插入可视化调用才能检查模型!

原则上，使用这两种 DSL 都可以构建与系统完全解耦的模型。只需通过黑板机制实现与系统的所有通信即可。`Behavior-Tree.CPP` 提供了用于该目的的专用 XML 原语，只要系统能够读写黑板，整个行为就可以用 XML 编程。这种分离结构允许模型被独立处理，比如用于可视化、测试、迁移到其他系统等。我们认为这是一种良好的架构实践。然而，在实际模型中我们并未观察到这一点（参见 Sect. 6）。大多数模型将行为规范与实现紧密耦合，使得关注点分离几乎不可能实现。

> Observation 3. 通过所研究的 DSL 实现的行为树模型往往与连接到底层系统的软件粘合代码紧密交织。这使得在系统外操作模型变得困难，阻碍了模型的可视化、测试与复用。

4.2 状态机 DSL 中的概念与语义

在本节中，我们将回顾在状态机语言 State MACHine (`SMACH`) [55] 和 Flexible Behavior Engine (`FlexBe`) [57] 中观察到的语法与语义特征。

这两种 DSL 实现的状态机都是事件触发的，这与多个领域中传统状态机的实现相一致。与所研究的行为树 DSL 不同，状态机 DSL 并没有显式的 tick 或周期性遍历概念。状态仅在模型执行开始时启动一次，只有当控制流回到该状态时才会重新进入。在 `FlexBe` 中，当前状态可以被中断，并根据外部事件触发跳转，从而支持响应式编程。这种中断机制得益于与状态关联的自治等级阈值，允许引入人的决策。而 `SMACH` 中则不具备响应性，当前活动状态会阻塞执行，直到返回结果。

从计算模型角度来看，这两种语言都类似于 Moore 机（状态驱动动作）[92]。用户需将状态实现为 Python 类：`SMACH` 中使用 `State` 类，`FlexBe` 中使用 `EventState` 类。与 UML 中基于 Moore 机的状态机实现不同，这些状态接口中并没有 entry/exit 动作。两者都提供一组方法，其中构造函数 `__init__` 与执行方法 `execute` 是实现状态动作的核心部分。两种 DSL 的构造函数都要求声明状态可返回的结果（作为事件触发条件）。此外，两种语言都包含其他类型的状态接口，如 `SMACH` 的 `ConditionState` 和 `FlexBe` 的 `LockableState`，更多细节见其官方文档 [59], [60]。

为实现状态与系统间的数据流，使用了一种名为 *UserData* 的本地作用域键值对字典。相比于行为树中的全局 *blackboard*，*UserData* 定义了本地的输入输出端口用于访问所需数据，而 *blackboard* 更像是任意节点都可访问的全局存储。有文献 [67] 和在线讨论[17]指出，*blackboard* 的现有实现会在行为树变大时引发命名冲突和意外覆盖的问题。有趣的是，受到状态机语言中作用域端口机制的启发，behavior trees 的开发者已引入作用域 *blackboard* 的新机制以改进此问题。[18] [19]

`SMACH` 与 `FlexBe` 通过设计模式 container 支持分层嵌套，从而允许用户构建 hierarchical state-machines 。[20]Container 本质上是一个 Python 模块，可扩展以支持不同的执行语义。此设计模式在所研究的 DSL 中也用于定义常见的高级控制流结构。我们将 container 分为嵌套容器与控制流容器。`StateMachine(SMACH )`与 `OperatableStateMachine(FlexBe )` 均用于定义状态机模型，也用于嵌套和构建 hierarchical state-machines 。

在 `SMACH` 中，`Concurrence`、`Sequence` 与 `Iterator` 是提供的控制流容器。`Sequence` 类似于行为树中的 Sequence 节点，状态按预定义顺序顺序执行。`Iterator` 是一个 repeat-until 循环，反复执行状态直到达到特定结果。`Concurrence` 则允许多个子状态同时激活（通过线程实现）。和行为树中的 Parallel 节点一样，并不提供真正的并发执行。其并发结果通过一个字典定义，其中键为容器结果，值为其子状态可能的结果。在 `FlexBe` 中，并发行为仅通过 `ConcurrencyContainer` 实现。一个 `SMACH` 中的 iterator 示例如图 Figure 7 所示。

除了常见的控制流结构，`FlexBe` 还通过 `flexbe_state` API 与 `generic_flexbe_states`[21] 状态库提供了常见状态的元类。我们在模型分析中也观察到其实际应用（见 Sect. 6.3）。这些状态的文档在 [57] 的附录 A.1 中有提供。

为频繁出现的控制流模式提供语言结构在所研究的行为树与状态机 DSL 中较为普遍。这可能与机器人任务具有一系列动作或迭代任务的性质有关，促使行为建模语言必须支持这类行为。

在建模语言中将行为设计模式作为语言结构直接提供，有时会被视为不良设计决策，因为这可能增加语言的复杂性。然而，正如 Bosch [93] 所指出，真正增加语言复杂性和用户负担的，是缺乏表达力的语言结构。一种语言应满足其领域的需求。在我们此前的研究 [26] 中，我们在比较 UML 行为建模语言（状态图与活动图）与行为树 DSL 时观察到了所支持结构的显著差异。Whittle 等人也报告了类似的定制需求 [94]。实际中，使用 DSL 替代 UML 建模语言是更常见的做法，因为机器人领域需要更强的表达力 [22]。因此，我们认为 DSL 开发者做出此类设计决策并不令人意外，这似乎正是机器人应用的真实需求。

> Observation 4. 为常见控制流模式提供结构在机器人任务建模中似乎是一种普遍需求。我们所研究的状态机与行为树 DSL 分别通过容器与复合节点概念来满足这一需求。

开放性

开放性是我们所研究语言中的一个共同特征，这与机器人任务的特性密切相关。与 behavior trees 类似，状态机 DSL 并不通过固定的模型和实现对用户施加约束。用户可获得用于容器和状态的元类，并可在此基础上进行扩展。控制流类型的扩展在语言中是可能的，然而在我们分析的项目中（见 Sect. 6），并未观察到类似的自定义，行为树项目中也同样如此。要解释这一现象的原因，需要进一步纳入语言用户进行深入调查。

这种设计在实践中支持了语言的开放性，并使其能够灵活适配机器人中的多样化场景。开放性似乎是由机器人社区对机器人行为的理想控制模型尚无共识所驱动的。由于这一问题在较长时间内仍可能悬而未决，该设计允许用户在构建机器人系统时根据自身需求自由适配语言。

17. 例如，问题曾在 `Behavior-Tree.CPP` 的 GitHub issues 页面中被提出：#18 #41 #44

18. https://www.behaviortree.dev/migrationguide/

19. https://py-trees.readthedocs.io/en/devel/changelog.html#x-2019-11-15-blackboards-v2

20. http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/container.html

21. https://github.com/FlexBE/generic_flexbe_states

5 语言实现（RQ2）

本节重点分析了表格 Table 1 前三行中加粗字体标示的 behavior tree 库，以及表格下半部分前两行中加粗字体标示的 state machine 库。对于这些已识别的库，我们通过检查其实现技术与实践，拓宽了分析范围。

5.1 行为树 DSL：语言设计与架构

从语言设计的角度转向行为树语言的实现方式，我们首先注意到的一个显著特点是，这两种语言主要是作为库进行分发的，而不是作为语言工具链或建模环境。`Behavior-Tree.CPP` 是作为一个 C++ 库实现的，被封装为一个 ROS 组件，能够轻松集成进 ROS 的代码库中 [66]。相比之下，`PyTrees` 是一个纯 Python 库。它的扩展 `PyTrees_ros` 将 `PyTrees` 封装为一个 ROS 包，并添加了 ROS 特定的节点。

语法与可视化

这两种方言都提供了将模型可视化为图的方式，`Behavior-Tree.CPP` 甚至还提供了一个图形编辑器和可视化运行时监视器 Groot（即 Fig. 1 中所展示的 behavior tree 图形化表示工具）。

然而，理解一点非常重要：behavior trees 并不是传统意义上的可视化建模语言。首先，在这两个库中，模型都是在文本编辑器中以 C++ 或 Python 的混合方式构建的。其次，模型是直接在抽象语法层面上构建的，通过实例化和连接抽象语法类型来实现。为了方便，并支持 Groot，`Behavior-Tree.CPP` 提供了一个 XML 格式，可用于在静态文件中编写树的语法。如 Listing 1 所示，展示了与 Figure 1 中任务对应的 XML 文件。该文件会在运行时被解释，并从中动态构建抽象语法树。第三，更为关键的是，节点的类型（以及 `Behavior-Tree.CPP` 中的 XML 文件）并不能代表模型的全部含义。模型的重要部分是嵌入在自定义节点类的 C++/Python 方法中的代码。这部分模型既不能在图形工具中修改，也无法被显示。

最后请注意，`Behavior-Tree.CPP` 是通过 Groot 和类似 XML 的格式实现的外部 DSL，而 `PyTrees_ros` 是一个内部 DSL，因为它没有类似的图形工具。根据我们在分析模型过程中的经验（参见 Sect. 6），我们确认 `Behavior-Tree.CPP` 模型更容易理解，而其图形化编辑器 Groot 的可用性也使得分析 behavior tree 模型的效率高于 `PyTrees_ros` 模型。

Listing 1: 使用 `Behavior-Tree.CPP` XML 表示法显示的与 Figure 1 相同的示例。

```
<root main_tree_to_execute="MainTree" >
    <BehaviorTree ID="MainTree">
     <SequenceStar name="MainSeq">
          <Action ID="CollectWaypoints"/>
          <RetryUntilSuccesful num_attempts="10" >
             <Negation>
                <Sequence name="ExplorationSeq">
                   <Action ID="PopWaypoint"/>
                   <Action ID="MoveBase"/>
                   <Action ID="Explore"/>
                </Sequence>
             </Negation>
          </RetryUntilSuccesful>
          </SequenceStar>
    </BehaviorTree>
</root>
```

并发性

我们所研究的行为树 DSL 并未规定特定的并发模型，不同实现方式也有所差异。例如，`Behavior-Tree.CPP` 引擎使用的是单线程执行引擎。所有节点应立即返回状态。如果某一操作耗时较长，应启动异步逻辑，并在完成后再次被 tick 时通知行为树引擎其运行状态（或已完成）。因此，程序员拥有选择并发机制的完全自由。

> Observation 5. 我们所研究的行为树 DSL 的实现通过依赖底层的 ROS 平台，间接支持交错执行与真正并发。并发模型并未在语言中严格定义，而是主要留给用户自行决定。

内部 *DSL* 还是外部 *DSL*？

我们的 DSL 拥有异常开放的特性。严格来说，`Behavior-Tree.CPP` 是一种外部 DSL，但其实现中暴露了许多动态内部 DSL 的特性。程序员既可以使用 XML（外部、静态）创建模型，也可以在运行时创建新的节点类型或动态修改语法树结构。`PyTrees_ros` 完全是动态 DSL，用户可以像使用内部 DSL 一样自由混合新节点类型与 Python 代码。

解释器还是编译器？

我们的 DSL 模型均为解释执行。一旦抽象语法树构建完成，用户可以调用某方法单次触发模型，或以固定频率持续触发模型。这种方式与其他“运行时模型（models-at-runtime）”的应用非常相似 [27], [28]。`Behavior-Tree.CPP` 使用模板元编程替代了代码生成，在实现自定义树节点时可提供一定的类型安全性，而无需使用任何专用的代码生成工具。该库的使用方式与普通 C++ 库无异。与预期一致，`PyTrees_ros` 并不提供静态类型安全性。

5.2 状态机 DSL：语言设计与架构

`SMACH` 与 `FlexBe` 是两个用 Python 编写的开源软件框架，用于构建和监控状态机（state machines）与层次状态机（hierarchical state-machines）。类似于 `PyTrees_ros`，`SMACH` 可以独立于 ROS 系统使用，同时也通过 `smach_ros` 提供了 ROS 绑定。

从语言设计的角度来看，这两个状态机 DSL 的实现方式存在差异。虽然 `FlexBe` 可直接作为 Python 库使用，但它更常被作为建模环境使用，因为其提供了名为 FlexBE App 的图形用户界面（GUI）。[22] FlexBE App 用于通过图形化方式构建状态机，并在运行时对状态机进行监视和修改。相比之下，`SMACH` 主要作为纯 Python 库使用，并通过 `smach_ros` 包集成进 ROS 系统。它也提供了一个名为 smach_viewer 的 GUI[23]，用于查看和调试已创建的状态机，但不支持构建或修改状态机。虽然 `SMACH` 本身不依赖 ROS，但其 viewer 无法在没有运行中的 ROS 系统的情况下运行。

语法与可视化

`SMACH` 和 `FlexBe` 拥有各自的表示法，与 UML 标准（见 Fig. 4）不同。`SMACH` 使用椭圆形表示状态，双框椭圆表示嵌套状态机，而 `FlexBe` 使用矩形表示状态。`FlexBe` 还使用了类似 UML 的初始态与终止态专用符号，而 `SMACH` 则没有。Figure 7 展示了一个使用 smach_viewer 语法的状态机示例。在 `SMACH` 和 `FlexBe` 中，事件被称为 outcome。`SMACH` 在可视化语法中区分了容器 outcome 与状态 outcome。引发状态转换的 outcome（即触发事件）显示在转换箭头上；而像 Fig. 7 中的 `Interator_10_attempts` 这样的容器 outcome 被明确表示为红色椭圆，看起来像状态。这偏离了典型的状态机语法。

这两种语言在构建状态机模型时的使用方式不同。`FlexBe` 将图形语法与文本语法结合使用。图形语法通过 FlexBE App（`FlexBe` 的图形化 GUI）以拖拽方式构建，随后由解释器生成 Python 代码形式的文本语法。生成的代码表示模型结构，部分代码可供开发者修改以增加灵活性。对于每个状态，系统

22. https://github.com/FlexBE/flexbe_app
23. http://wiki.ros.org/smach_viewer

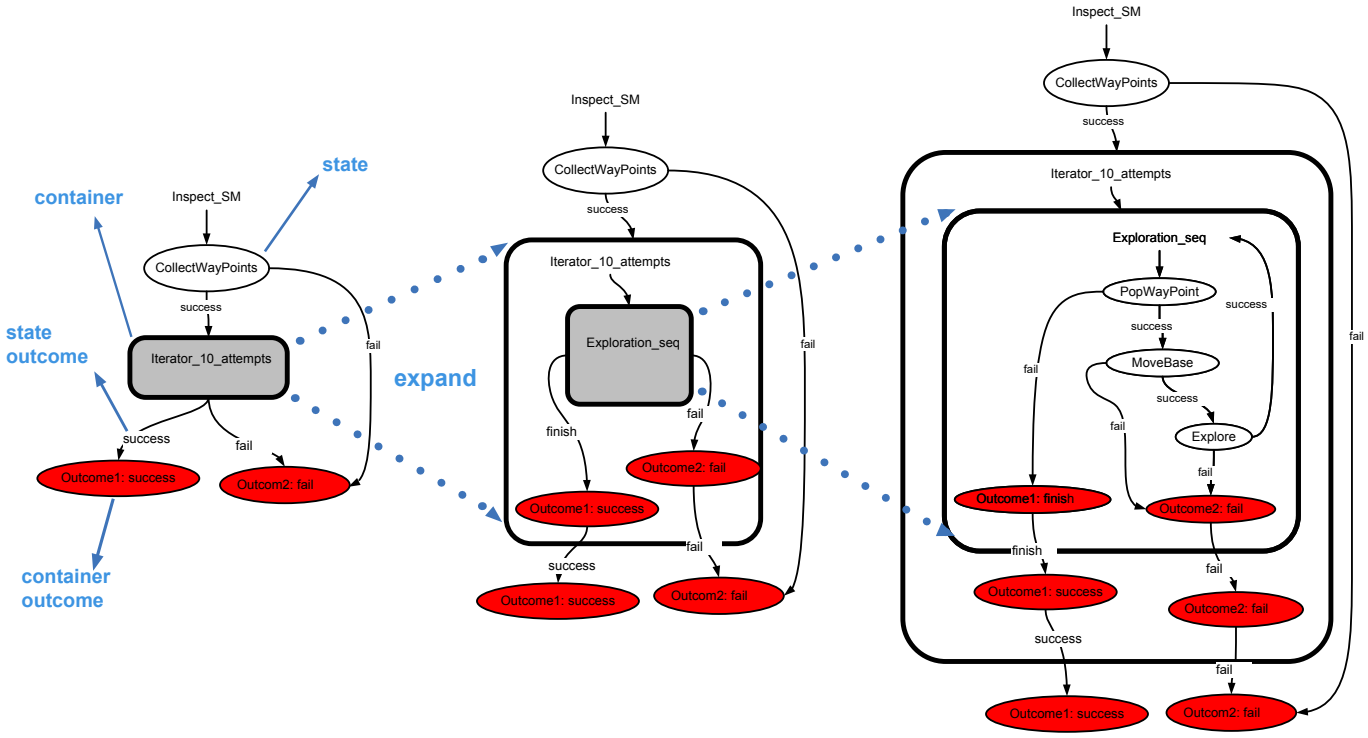

Fig. 7: 使用 smach_viewer 语法表示与 Fig. 1 和 Fig. 3 中相同任务的一个 state machine 示例。左侧为俯视图，中间和右侧图分别展示了展开的迭代器容器和嵌套容器。

会自动生成包含基本函数（如构造函数、执行函数等）的方法框架。开发者需在 Python 中完善这些状态方法。

相比之下，SMACH 用户则直接在文本编辑器中以 Python 语言构造抽象语法模型。

FlexBe 在模型构建过程中使用了模型驱动工程（MDE）技术。通过从模型（半）自动生成代码，体现了模型驱动设计流程。它还支持语法与语义一致性检查，因此在 FlexBe 中也支持角色分离。Behavior-Tree.CPP 也通过提供图形化建模界面和将模型生成 XML 文本语法的机制实现了角色分离 [66]，同时支持关注点分离。不过 Behavior-Tree.CPP 仍需手动重复编码。相比之下，FlexBe 通过生成样板代码（如必要库的导入和默认初始化函数）减少了手动编程工作量。如果将此特性引入行为树语言中，可能会降低编程负担并减少语法错误。然而，由于自定义执行函数仍需手动编写，完全无需编程背景仍不可行。

> Observation 6. 其中一个状态机 DSL 引入了模型驱动工程的理念，有助于减少重复编码与语法错误。如果为行为树 DSL 提供类似支持，可进一步降低编程负担与语法出错率。

并发性

与 behavior trees 类似，状态机 DSL 的实现中并未强制规定特定的并发模型。用户可利用 ROS 平台中提供的任意并发语义。在这两个库中，每个状态机容器在自己的线程中执行，其 *UserData*（本地作用域的键值对字典）不会在多个状态机容器间共享。如需在两个运行中的状态机之间共享数据，必须借助外部同步与通信机制。

所分析的 DSL 提供了并发容器。在两个DSL中，均采用 Python 的 threading 模块，允许多个状态并行激活，各自以独立速率运行。但由于使用的是 Python threading（即交错执行），状态实际是顺序激活而非真正并行。

内部 *DSL* 还是外部 *DSL*？

FlexBe 的开放性与其他采用模型驱动设计的建模语言类似，与 Behavior-Tree.CPP 一样，它被实现为外部 DSL。状态机模型可通过 FlexBE App 的图形化方式构建。在生成代码前，系统会进行语法与部分语义一致性检查，例如是否定义了每个状态所需的 *UserData*。

模型与状态的扩展可在线下进行。一旦模型与状态实例化，在运行时只能对状态机结构进行有限修改，例如添加或删除状态、转换函数等，不能修改状态的具体实现。相比之下，SMACH 类似 PyTrees_ros ，是内部 DSL，用户只需编辑 Python 代码即可轻松扩展模型与状态。

解释器还是编译器？

这两个库都在运行时以解释方式执行状态机模型。FlexBe 允许在运行时对模型进行修改。FlexBe 通过补丁（patch）机制实现对运行中状态机的修改，该机制是对变更的最小摘要表示。补丁通过比较旧代码与新代码之间的差异，并仅重新导入变更内容来生效。为确保修改安全，在应用补丁前会使用 Python threading lock[24] 锁定被修改的状态以防止转换，然后使用内建的 reload() 函数重新加载 Python 模块。

在 FlexBe 引擎运行一致性验证检查之前，不会应用任何修改（参见 Sect. 5.2）。总的来说，FlexBe 的开发者基于 Python 的 reload 支持，构建了一个实用的运行时状态机热更新机制。SMACH 则在模型执行前没有验证步骤。

6 行为树与状态机模型（RQ3）

我们在挖掘中为每种 DSL 找到了数百个 GitHub 项目。以下是我们对这些项目中所使用状态机与行为树DSL的探索性使用情况调查。首先，我们概述了这些语言在所挖掘的开源项

24. https://docs.python.org/3/library/threading.html

目中的流行度差异。其次，我们对部分项目样本进行了分析，内容包括模型结构及其代码在项目之间的复用情况。

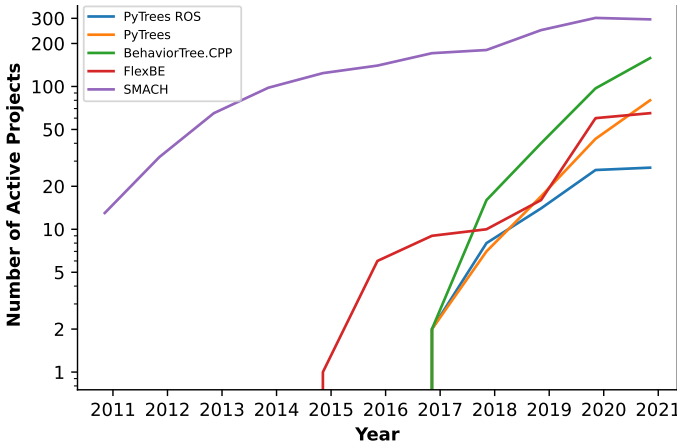

Fig. 8: 我们研究的状态机与行为树 DSL 在 GitHub 开源项目中的使用趋势概览

6.1 语言流行度

初步挖掘共找到了 1,086 个使用状态机DSL的项目，以及 271 个使用行为树DSL的项目（检索关键词见 **??**）。为可视化这些语言（即库）在开源项目中随时间的使用情况，我们统计了每年每个库中的活跃项目数量，并绘制了每个库的趋势线。

Figure 8 展示了上述趋势。2015 年之前，`SMACH` 是五种 DSL 中唯一可用的语言，其使用量在过去五年中稳步增长（每年约增长 1.2 倍）。2015 年，其余四种语言被发布，但直到 2018 年它们的使用才开始显著增加。总体而言，这些语言的使用在过去五年中均呈增长趋势。到 2021 年，`Behavior-Tree.CPP` 和 `PyTrees` 在开源项目中的使用量已达到 2018 年的近十倍。

我们推测，过去三年中这种显著增长可能是由于 ROS 将其作为核心组件集成进导航栈 Navigation 2 [35]。对于 `PyTrees`，其增长可能归因于它是唯一一个稳定维护、构造丰富、面向 behavior trees 的 Python 实现。虽然 `FlexBe` 在开源项目中的使用量仍低于 `SMACH`（2021 年比 `SMACH` 少 78%），但自 2018 年以来，其整体增长速度远高于 `SMACH`。2021 年 `FlexBe` 的使用量是 2018 年的 6.5 倍，而 `SMACH` 仅增长了约 1.6 倍。

6.2 模型特征

在我们多轮过滤后的挖掘结果中，共得到包含 2,507 个状态机模型的 620 个项目，以及包含 658 个行为树模型的 169 个项目。Table 5 展示了每种 DSL 的项目数与模型数量。在每个项目中，只要文件中完整定义了一个模型，就计为一个模型，因此一个项目中可能包含多个模型。我们从中随机抽取了每种语言各 75 个模型。

我们总共分析了 150 个行为树与状态机模型（每种类型各 75 个），这些模型分属 43 个项目（25 个为 behavior trees 项目，18 个为 state machines 项目），详见 Table 4。项目数量不等是由于我们选择按模型数量而非项目数量进行等量抽样所致。

在我们的样本数据集中，共识别出 11 个不同的应用领域，如 Figure 9 所示。其中占主导的应用领域为机器人巡逻、服务机器人和抓取搬运（pick&place）任务。我们将项目分为三类：(1) 研究项目，来自实验室、研究小组或比赛队伍；(2) 企业项目，来自生产机器人解决方案的公司；(3) 未知项目，无法获取足够信息加以分类。Table 4 中列出了这些项目的类型，鼠标悬停在蓝色文本上可查看其所属组织和 GitHub 仓库链接。

TABLE 4: 我们抽样的 GitHub 项目概览：使用行为树与状态机 DSL 来定义机器人行为

project, GitHub link	language	models	Type
sam_march KKalem/sam_march	`PyTrees_ros`	1	research
mobile_robot_project simutisernestas/mobile_robot_project	`PyTrees_ros`	1	unknown
smarc_missions smarc-project/smarc_missions	`PyTrees_ros`	2	research
dyno samiamlabs/dyno	`PyTrees_ros`	2	company
gizmo peterheim1/gizmo	`PyTrees_ros`	8	unknown
roborts_project Taospirit/roborts_project	`PyTrees_ros`	1	unknown
robotics-player braineniac/robotics-player	`PyTrees_ros`	1	unknown
refills_second_review refills-project/refills_second_review	`PyTrees_ros`	1	research
Robotics-Behaviour-Planning jotix16/Robotics-Behaviour-Planning	`PyTrees_ros`	3	unknown
pickplace ipa-rar/pickplace	`B-Tree.CPP`	1	research
stardust julienbayle/stardust	`B-Tree.CPP`	4	research
neuronbot2_multibot skylerpan/neuronbot2_multibot	`B-Tree.CPP`	2	unknown
mecatro-P17 alexandrethm/mecatro-P17	`B-Tree.CPP`	11	unknown
Yarp-SmartSoft-Integration CARVE-ROBMOSYS/Yarp-SmartSoft-Integration	`B-Tree.CPP`	1	research
bundles MiRON-project/bundles	`B-Tree.CPP`	5	research
BTCompiler CARVE-ROBMOSYS/BTCompiler	`B-Tree.CPP`	8	research
BT_ros2 Adlink-ROS/BT_ros2	`B-Tree.CPP`	2	company
vizzy_behavior_trees vislab-tecnico-lisboa/vizzy_behavior_trees	`B-Tree.CPP`	7	research
hans-ros-supervisor kmi-robots/hans-ros-supervisor	`B-Tree.CPP`	1	research
Pilot-URJC MROS-RobMoSys-ITP/Pilot-URJC	`B-Tree.CPP`	2	research
vizzy_playground vislab-tecnico-lisboa/vizzy_playground	`B-Tree.CPP`	6	research
behavior_tree_rosC++ ParthasarathyBana/behavior_tree_rosC++	`B-Tree.CPP`	1	unknown

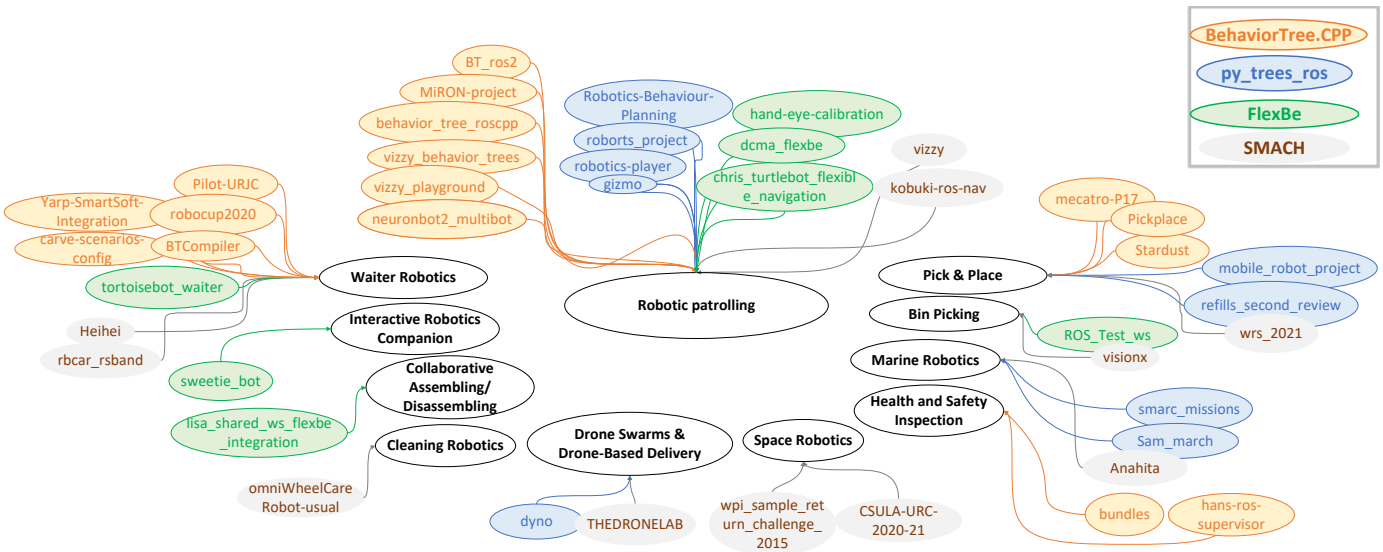

Fig. 9: 被分析项目的应用领域概览

TABLE 4: Sampled projects overview (continued)

project, GitHub link	language	models	Type
MiRON-project ajbandera/MiRON-project	B-Tree.CPP	1	research
robocup2020 IntelligentRoboticsLabs/robocup2020	B-Tree.CPP	2	research
carve-scenarios-config CARVE-ROBMOSYS/carve-scenarios-config	B-Tree.CPP	1	research
ROS_Test_ws QuiN-cy/ROS_Test_ws	FlexBe	6	unknown
lisa_shared_ws_flexbe_integration lawrence-iviani/lisa_shared_ws_flexbe_integration	FlexBe	4	research
sweetie_bot sweetie-bot-project/sweetie_bot	FlexBe	5	company
chris_turtlebot_flexible_navigation CNURobotics/chris_turtlebot_flexible_navigation	FlexBe	3	research
hand-eye-calibration tku-iarc/hand-eye-calibration	FlexBe	2	research
dcma_flexbe andy-Chien/dcma_flexbe	FlexBe	8	unknown
tortoisebot_waiter rigbetellabs/tortoisebot_waiter	FlexBe	2	company
Anahita AUV-IITK/Anahita	SMACH	2	research
wrs_2021 hentaihusinsya/wrs_2021	SMACH	2	unknown
Heihei Luobokeng2021/Heihei	SMACH	2	unknown
THEDRONELAB DeVinci-Innovation-Center/THEDRONELAB	SMACH	2	research
kobuki-ros-nav vibin18/kobuki-ros-nav	SMACH	3	unknown
visionx tsoonjin/visionx	SMACH	3	unknown
wpi_sample_return_challenge_2015 rcxking/wpi_sample_return_challenge_2015	SMACH	3	research
rbcar_rsband darshan-kt/rbcar_rsband	SMACH	4	unknown
vizzy vislab-tecnico-lisboa/vizzy	SMACH	6	research
omniWheelCareRobot-usual Art-robot0/omniWheelCareRobot-usual	SMACH	7	unknown
CSULA-URC-2020-21 CSULA-URC/CSULA-URC-2020-21	SMACH	11	research

我们观察到，在所分析样本中，行为树模型与状态机模型的结构特征存在差异。状态机模型的模型规模呈右偏分布，平均模型规模为 9；行为树模型也呈右偏分布，但其平均模型规模为状态机模型的三倍（总体平均为 26）。值得注意的是，行为树的平均分支因子（BT.ABF）为 3，说明开发者通常保持树结构在可控范围内。行为树模型规模较大的原因可能在于其同时表示执行节点与控制流节点，而状态机模型仅表示状态转换 [15]。另一种解释是状态机模型在规模变大后结构会变得复杂，影响其可理解性 [95], [96]，因而开发者倾向于保持模型较小。对于 behavior trees ，目前文献中尚未有研究明确指出模型的最优大小。但通过我们对样本的分析发现，适中的规模与分支程度更便于模型的理解与导航。这里的“适中”指的是 BT.size 和 BT.ABF 的平均值。

我们还关注了另一个结构特征：嵌套与深度。已有实证研究表明，嵌套层级影响状态机模型的可理解性，因此浅层模型更为可取 [97], [98]。

在我们的样本中，状态机模型平均嵌套层级为 1，而行为树模型平均深度为 5（在不同 DSL 中相似）。在状态机模型中，使用 `FlexBe` 的模型比 `SMACH` 更频繁地实现了 hierarchical state-machines ，其中 14 个层次模型中有 11 个来自 `FlexBe` 项目。我们样本中的开发者似乎都保持了模型的浅层结构，无论是构建状态机还是行为树模型。

不幸的是，尚无研究探讨行为树深度对复杂性的影响，无法解释为何开发者倾向于设计浅层模型。一种可能的解释是出于类似状态机模型中的可理解性考量。但这仍需进一步研究加以证实。

接下来我们分析样本模型中语言结构的使用情况，先从行为树开始。在我们收集的行为树模型中，66% 的节点为叶子节点（总节点数 1,850 中有 1,228 个），34% 为复合节点。Table 6 总结了各库中复合节点的使用情况以及总体占比。

大多数复合节点为 Sequence 类型，占总复合节点的 56%；其次是 Selector 类型，占 21%。Parallel 节点（可视为 Sequence 和 Selector 的泛化）使用较少，仅占 7%。这或许也解释了为何标准编程语言库中很少包含存在量词与全称量词（exists 与 forall）的泛化函数——这些用例本身就很少。行为树DSLs的重入特性允许使用 Parallel 等待多个子树成功后再继续，但实际使用频率并不高。

在 `PyTrees_ros` 模型中，装饰器的使用频率相对较低，仅占复合节点的 6%。这可能是因为相比于使用行为树抽象语法结构，开发者更倾向于在 Python 代码中直接实现变换逻

TABLE 5: 每个库中提取的项目数与模型数

	SMACH	FlexBe	B-Tree.CPP	PyTrees_ros
项目数	560	60	141	28
模型数	2065	442	595	63

TABLE 6: 各行为树库中不同复合节点类型的使用占比，以及总体分布

	复合节点占比			
库	Sequence	Selector	Decorator	Parallel
Behavior-Tree.CPP	57%	19%	19%	6%
PyTrees_ros	53%	28%	6%	13%
总体占比	56%	21%	16%	7%

辑。而在 Behavior-Tree.CPP 中，装饰器的使用频率要高出近三倍（占复合节点的 19%）。这是因为使用行为树（而非 C++）中的数据流装饰器能让其在图形编辑器（Groot）中被可视化与监控。而 PyTrees 中没有类似工具，因此模型的大部分内容可能“泄露”到了代码中。

这说明行为树用户常常需要决定模型的“作用域”——即在领域分析时确定哪些内容纳入模型，哪些用程序实现。这一点明显区分了通用编程语言（如 Python 与 C++）与 DSL。然而据我们的经验，关于如何界定模型作用域与精度层级的能力，在教学与研究文献中鲜有涉及（但存在例外 [31]）。

最后，我们观察到所有模型均未实现自定义节点。开发者主要依赖 behavior trees 的可扩展性，通过新增的自定义操作符（即装饰器）来完成特定行为。通过使用 Behavior-Tree.CPP 和 PyTrees_ros 中现成的装饰器，即可实现例如改变动作/条件的返回状态、调整动作持续时间等定制行为。例如：无需等待即可执行某个动作、重试某个动作 n 次后再放弃，或重复执行某个动作 n 次等。

回到 Fig. 1，装饰器（RetryUntilSuccesful）用于构造一个条件循环，该循环执行子树（ExplorationSeq）最多 10 次，除非任务失败。而失败结果通过（Inverter）转换为成功。开发者无需在脚本中使用 while 循环或类似的控制流结构，也能完成此建模。

> Observation 7. 所研究的行为树 DSL 提供了适合机器人开发者使用的一系列语言结构，但其使用情况在不同语言中因 GUI 支持的差异而有所不同。

对于状态机 DSL，我们聚焦于控制流容器，它们与 behavior trees 中的复合节点概念等价。如 Sect. 4.2 所述，SMACH 提供了多种控制流容器，但与 FlexBe 仅共享 Concurrence 行为。因此，Sequence 与 Iterator 的使用数据仅针对 SMACH 报告。

总体来看，样本中的状态机模型使用控制流结构的频率远低于行为树模型。仅有 12% 的状态机模型使用了某种控制流结构，而所有行为树模型均使用了某种控制流节点。Concurrence 在所有模型中的使用率为 11%，其中大部分来自 FlexBe （8 个模型中有 7 个使用了 FlexBe ）。虽然在 SMACH 中 Concurrence 的使用频率较低，但在 SMACH 项目中却占到了全部控制流结构的 67%。Iterator 是 SMACH 项目中第二常见的控制流结构，占比为 33%。而 Sequence 在我们的样本中未被使用。

尚不清楚为何状态机模型中所提供的控制流结构不如行为树中受欢迎。可能的解释包括状态机的语法本身——即状态表示系统状态、转换表示控制流——已经足够清晰，因此不需要额外引入控制流结构。此外，也可能与 PyTrees_ros 中装饰器的情况类似：在 Python 中直接实现控制逻辑比将其提升为抽象语法结构更为简单。

TABLE 7: 各库在技能层与任务层复用模式的频率分布

库	**模型内引用**		**克隆-复用**		**模型间引用**	
	技能	**任务**	**技能**	**任务**	**技能**	**任务**
SMACH	-	-	20%	13%	35%	8%
FlexBe	1.3%	-	-	16%	32%	-
状态机汇总	**1.3%**	-	**20%**	**29%**	**67%**	**8%**
B-Tree.CPP	25%	13%	3%	39%	37%	3%
PyTrees_ros	5%	5%	13%	9%	19%	-
行为树汇总	**30%**	**18%**	**16%**	**48%**	**56%**	**3%**

6.3 复用

在介绍了我们样本中模型的核心结构特征后，我们转向另一个机器人工程软件 [1], [5], [79], [80], [81], [99] 和控制架构 [4], [30] 中的关键议题——复用。为了促进复用，建模语言应支持将机器人任务或行为划分为模块化组件。支持模块化设计（进而支持可复用组件）的行为建模语言，是克服机器人控制架构挑战、提升软件可维护性与质量的关键因素之一 [4], [5], [30], [99]。

在我们的研究中，复用指的是在同一模型中或跨模型中复用已实现的技能代码（在某些上下文中也称为动作），或复用由多个技能构成的重复任务的代码。我们使用 skill-level 和 task-level 来分别指代这两类复用。在我们样本中观察到的 skill-level 和 task-level 复用中，出现了三种模式：*模型内引用*（*intra-model referencing*）、*克隆-复用*（*clone-and-own*） [100] 和 *模型间引用*（*inter-model referencing*）。

*模型内引用*主要出现在行为树模型中。在任务层面，其实现方式是将重复的行为建为一个子树，并在模型的多个分支中通过引用该子树实现复用，并传递不同的参数（通常通过向黑板写入新值）。在技能层面，一个叶子节点被定义为主模型执行文件中的函数，再通过传参进行复用。

行为树模型广泛采用了*模型内引用*用于技能层面（30% 的行为树模型），而在任务层面仅 18% 的模型使用了这种模式。该模式在状态机模型中很少见，仅一个模型通过引用复用了技能，少数在任务层面复用。Figure 10 展示了我们收集的一个行为树模型片段，用于养老院服务机器人。图中红框展示了一个 Recharge 子树的模型内引用示例；在技能层面，动作 moveRoboterPosition 被多次复用，仅修改参数（如 name、approachRadius、x 和 y）即可。

克隆-复用（*clone-and-own*）是所有 DSL 中任务层面最常用的复用模式。在包含多个行为树模型的项目中，我们观察到多个模型具有相似行为时，常通过复制一个子树或整个模型，并在其上添加或删除节点的方式实现复用。状态机模型中也通过增加或删除状态来调整模型以适应类似任务。在技能层面，克隆-复用通常仅做少量代码修改。

在我们样本中，行为树与状态机用户更倾向于使用克隆-复用来实现任务层面的复用（48% 的行为树模型和 29% 的状态机模型采用该模式）。但在技能层面，克隆-复用使用频率较低：行为树模型中仅 16% 使用，状态机模型中为 20%。Fig. 11 展示了来自 Dyno 项目（一个基于无人机的包裹投递系统）中的两个行为树模型：M1 为包裹投递任务，M2 为路径调度任务。这是任务层面克隆-复用的示例，开发者复用了一个完整的行为树模型，并通过适当修改使其适配不同任务。[25]

*模型间引用*是语言中技能层面复用最常用的模式。重复技能被实现为外部文件中的动作节点或状态函数，并在主模型

25. 该模型及其他项目模型可在在线附录 [29] 中查看。

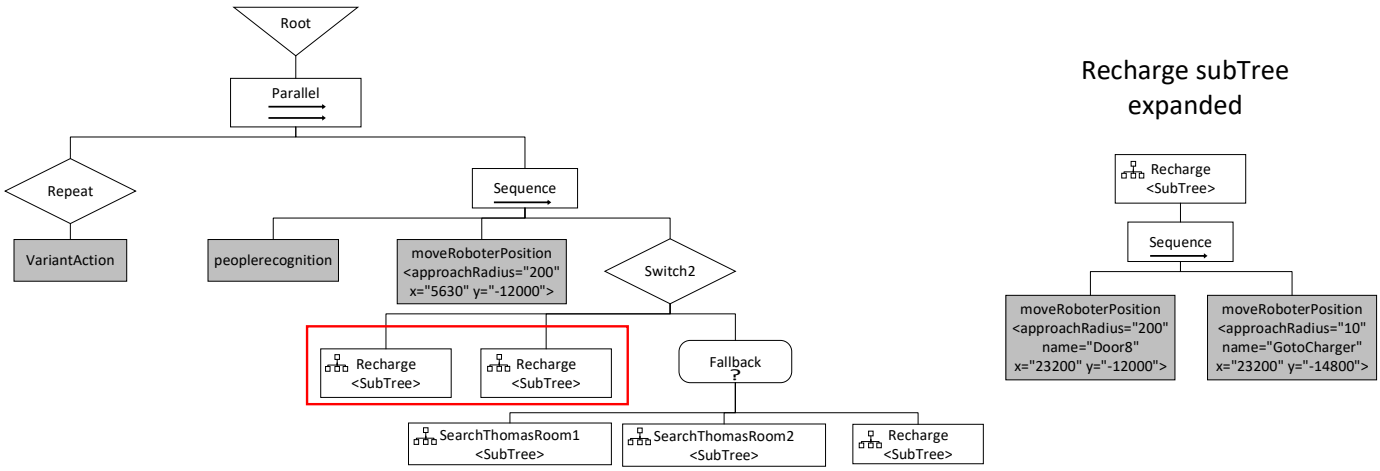

Fig. 10: 项目 bundles 中某养老院服务机器人任务的 Behavior tree 模型。红框部分展示了一个名为 Recharge 的子树在模型内引用的示例（右侧为其展开结构）。图例见 Figure 2。

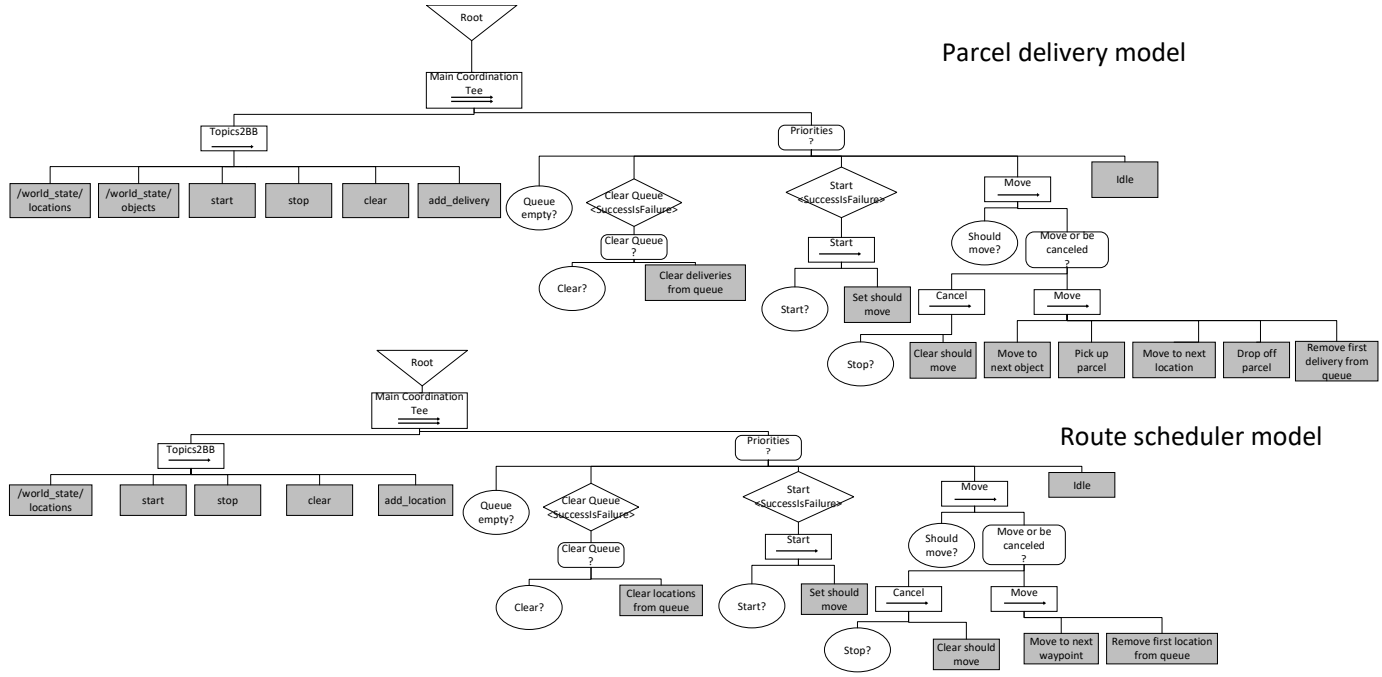

Fig. 11: Dyno 项目中行为树模型的克隆-复用示例。每个模型属于不同的任务：M1（包裹投递）与 M2（路径调度）。图例见 Figure 2。

中导入。这允许多个模型间共享相似技能。在任务层面，也可将重复任务定义为外部的行为树或状态机模型，并在主模型中作为子树或嵌套状态机引入。

在各语言中，模型间引用是技能层面最常用的复用模式（67% 的状态机模型与 56% 的行为树模型使用）。但在任务层面，仅有两个状态机项目与一个行为树项目使用该模式（分别占 8% 和 3%）。

我们注意到 `FlexBe` 的开发者通过 API `flexbe_state`[26]、状态库 `generic_flexbe_states`[27] 与 `Flexible Navigation`[28] 提供了预定义技能。该机制被 73% 的 `FlexBe` 模型所使用。类似地，`SMACH` 与 `PyTrees_ros` 项目中也大量复用如 `move_base` 等 `ROS Navigation Stack` 技能（分别出现在 55% 和 60% 的项目中）。这表明，预定义技能 API 能有效促进复用，是值得开发者进一步探索的机制。

总体来看，我们在样本中观察到复用机制的选择受 DSL 实现方式及其面向技能或任务的使用目标影响。模型间引用因其简单易用，在技能层面被广泛采用。相比之下，在任务层面，模型间引用使用频率最低，克隆-复用最受青睐。因为相似任务可能共享一组技能，但组合方式可能因具体需求而变化，克隆-复用更灵活。深入分析各 DSL，我们发现 `Behavior-Tree.CPP` 项目中的复用频率明显高于 `PyTrees_ros`，可能因为 `Behavior-Tree.CPP` 提供了 XML 表达方式与 Groot 图形化工具，使模型脱离代码而独立表达，而 `PyTrees_ros` 的模型则与代码紧耦合。模型的可视化与抽象可能是推动机器人任务复用的重要方向。

我们推测，当前识别出的简单复用机制足以满足所研究机

26. https://github.com/FlexBE/generic_flexbe_states
27. https://github.com/FlexBE/generic_flexbe_states
28. https://github.com/FlexBE/flexible_navigation/tree/ros2-devel-alpha/flex_nav_flexbe_states

器人项目的需求。尚不清楚引入通用编程语言中的更安全与丰富的复用机制（如命名空间与复用契约）是否有实际效益，因为这些机制可能对用户学习与使用造成较大负担。是否能实现足够轻量且安全的复用机制，仍需进一步研究。

7 有效性威胁

内部有效性（Internal Validity）我们提供了一个机器人领域中包含行为树（behavior tree）和状态机（state machine）模型的数据集。该数据集中可能包含来自课程或教程的项目，也可能遗漏了一些使用我们所研究库表达的 behavior tree 或 state machine 模型的项目。为缓解这一威胁，我们在挖掘过程中采用了多步过滤机制，并根据观察结果不断调整。每一步过滤后，我们都会随机抽样检查结果，以判断是否需要修改过滤机制。

另一个可能影响结果的威胁是用于计算模型指标的 Python 脚本存在错误。在行为树部分，作为质量控制的一种方式，我们手动统计了节点类型并与脚本结果进行对比；在此过程中，我们排除了注释部分和未使用的节点类型。在状态机部分，我们通过人工检查模型来复核自动计数结果。

外部有效性（External Validity）一个外部威胁在于我们量化结果和识别的复用模式的泛化能力，因为我们是从挖掘出的模型中随机采样的。为缓解这一问题，我们将模型根据其数据分布划分为不同大小的池，并从每个池中随机抽取样本，从而确保覆盖不同规模的项目模型。此外，Fig. 9 所示的项目领域分布也表明我们的样本覆盖了多个领域类别。

我们识别的 开源机器人项目可能遗漏了来自 Bitbucket 和 GitLab 的示例。虽然这两个平台在机器人社区中也有使用，但由于它们不提供代码搜索 API，使得我们难以进行代码层面的系统搜索。我们曾尝试在 Bitbucket 和 GitLab 网页端使用“behavior trees”和“state machine”作为关键词进行搜索。然而，在行为树方面，我们未能从搜索结果中识别出真实的机器人项目；在状态机方面，GitLab 返回的结果难以确认是否使用了我们研究的库。我们最终选择使用 GitHub 作为主要挖掘平台，原因在于其提供灵活的 API、良好的与 Python 等语言的集成能力，以及其作为开源项目发布的主流平台在开发者中的广泛使用程度。

我们仅考虑了使用 Python 和 C++ 并支持 ROS 的项目，但也可能存在使用其他语言的开源机器人项目。我们承认将搜索范围限制在支持 ROS 的语言可能会遗漏其他项目。然而，我们聚焦于 ROS 中最主流的两种语言，是基于 ROS 作为当前最具代表性的 开源机器人框架的假设。

8 相关工作

在已有文献中，研究人员主要从理论角度将行为树（behavior trees）与机器人中常见的控制架构进行比较，如有限状态机（finite state machines）、子行为架构（subsumption architecture）和决策树（decision trees），并指出行为树对这些架构进行了泛化 [6], [15], [30]。这些研究还基于机器人控制架构设计中重要的属性，探讨了各类控制架构的优缺点。Colledanchise 等人 [14] 通过示例说明了与行为树相比，在多机器人场景中使用状态机的局限性。在另一项工作中，Colledanchise 等人 [67] 展示了如何将状态机中不同类型的行为表示为行为树，并介绍了支持这类表示的软件库。行为树的模块性、响应性、表达能力和可读性等属性也被以形式化的方式进行研究，并与状态机进行比较 [11]。Colledanchise 和 Natale [67] 指出，与状态机相比，行为树的工具生态尚不成熟，但这一结论并未建立在对现有工具的系统分析基础之上。在我们的研究中，我们观察到所研究的行为树和状态机 DSL（工具）在语言设计上均有良好的实践，也存在一些不足。然而，我们无法得出状态机 DSL 比行为树 DSL 设计得更好的结论。每种语言设计各有其优劣。

现有相关工作在很大程度上并未基于真实世界机器人项目中的软件工程实践得出结论。而在本研究中，我们从软件语言的角度聚焦于机器人中的行为树 DSL，并将其与状态机 DSL 进行比较，这是以往研究尚未涉及的内容。与以往从理论角度进行对比的研究不同，我们将 behavior trees 与 state machines 的比较转向了真实世界场景。我们并未报告某种模型相对于另一种模型的优势或劣势，而是聚焦于支持这两种模型的语言在机器人中的相似性与差异性。我们通过分析开源项目中行为树与状态机 DSL 的使用情况，报告其实际使用现状。在本研究中，我们重点关注开源项目中所使用语言的结构属性以及代码复用情况。因此，我们的研究对现有相关研究形成了补充。此外，我们还提供了一批行为树和状态机模型作为社区数据集，这是此前文献中未曾提供的资源，该数据集可用于后续研究。

9 结论

我们开展了一项关于行为树（behavior-tree）和状态机（state-machine）DSL 的研究，并分析了它们在开源 ROS机器人应用中的使用情况。我们系统性地比较了主流 behavior tree DSL 提供的语言概念，并将其与成熟的响应式建模 DSL——即状态机语言进行对比。我们从代码仓库中挖掘开源项目，并从其代码库中提取行为树与状态机模型。我们分析了模型样本的结构特征，使用了哪些语言构件，以及这些模型的代码是如何复用的、复用的机制为何。我们同时贡献了一份包含模型的数据集、分析脚本和补充数据，公开发布于在线附录 [29]。

研究结果与启示

我们的分析揭示了一些由语言工程之外社区设计、但在机器人这一充满活力的重要领域中被广泛采用的 DSL 实践。我们认为研究建模与语言工程方法对于语言工程社区和机器人社区都是有益的，这有助于改进语言工程工具与方法，也能促进语言设计和使用实践的演进。

研究表明，这两类 DSL 在设计上都遵循了良好的实践，有些方面甚至相互借鉴并取得改进。例如，提供图形化界面（GUI）用于模型构建、编辑和运行时监控是一个优秀的实践。GUI 能提升模型可视化与抽象构建能力。我们观察到，`FlexBe` 和 `Behavior-Tree.CPP` 都提供了类似工具，用户在这些平台中比在 `SMACH` 与 `PyTrees_ros` 中表现出更高的代码复用频率。

此外，研究结果也展示了建模与语言工程方法在实践中的实际适用性。我们看到，采用务实的方式开发 DSL——不依赖冗长的规范文档，而是构建简单但可扩展的元模型，甚至在没有显式元模型的情况下，也能取得成功。这种策略吸引了未受过语言工程训练的从业者，使他们能够从底层编程范式转向更高层次的抽象建模，从而更高效地构建机器人任务。

这些语言的设计也采纳了软件工程的一些理念。例如，`Behavior-Tree.CPP` 遵循“关注点分离（separation of concerns）”和“角色分离（separation of roles）”的设计思想 [66]，使得模型能够与机器人系统解耦，便于可视化与测试。在我们的分析过程中，即使未配置 ROS 或相关依赖，我们也可以通过 Groot 工具轻松加载模型。此外，检查其 XML 表达也相对容易。相比之下，`PyTrees_ros` 和状态机 DSL 缺乏便捷的可视化工具，我们不得不采取非常规手段分析模型。

并非所有研究语言都同等程度支持角色分离。例如 `FlexBe` 采用模型驱动设计进行代码生成以处理模板代码，而`Behavior-Tree.CPP` 依赖内部 DSL 编写定制逻辑，需具备编程能力。虽然代码生成与外部 DSL 相较内部 DSL 在灵活性上更受约束，但通过代码生成处理重复结构的能力非常有价值，这使得更多建模工作可以由领域专家完成。此外，`FlexBe` 中还支持语法和语义一致性检查，这在外部 DSL 中体现较

好。然而，所有研究对象的用户都需具备一定的语言导向编程（meta-programming）能力，这对于多数非语言工程背景的机器人开发者来说仍是挑战。这为语言设计社区提出了一个有趣的问题：如何降低模型集成至机器人架构的门槛。

近年来，机器人社区正逐步推广和采纳模型驱动工程（model-driven engineering, MDE）实践，以提升系统的可复用性和可维护性 [5], [22], [78], [79], [99], [101], [102], [103], [104]。在本研究中，我们观察到这些 DSL 语言在建模实践中体现了许多 MDE 与软件工程的通用理念，且在开发者中获得了良好反馈。未来，这些行为建模语言的设计改进可继续在机器人领域中试验与推广。

未来工作

未来，我们希望基于当前的观察成果，开展引入用户参与的进一步研究。我们的目标是比较所研究的行为建模语言在可用性、可理解性与表达能力方面的差异，并探索其改进方向。特别值得研究的是，系统性地分析如何在不同行为建模语言中实现任务需求（例如，以自然语言文本形式给出的任务规范），并评估其表达能力，以及模型的简洁性与直观性等非功能性特征。

我们还希望总结并发布一套有助于推广模型驱动工程（MDE）实践的行为建模语言设计指南，同时结合当前的实际使用现状进行提炼。最后，在我们的分析过程中，我们注意到大多数项目缺乏明确的任务规范。因此，我们计划利用构建的数据集开发自动化工具，从现有的 behavior tree 模型中生成自然语言形式的任务规范。若能实现这一目标，将有助于构建面向遗留项目的自动逆向工程工具，从而为其他从业者复用这些项目提供支持，避免从零开始开发。

另一个有价值的未来方向是收集更广泛的实证数据，研究行为建模语言在机器人项目中的实际使用情况，而不限于行为树和状态机 DSL。尽管我们认为这两类是最主要的建模语言类型，但我们也相信在开源项目中还存在其他类型的语言。然而，识别这些语言的使用并不容易，这本身就值得开展一项或多项后续研究。

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[1] S. García, D. Strüber, D. Brugali, T. Berger, and P. Pelliccione, "Robotics software engineering: A perspective from the service robotics domain," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 593–604.

[2] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification patterns for robotic missions," *IEEE Trans. Software Eng.*, vol. 47, no. 10, pp. 2208–2224, 2021. [Online]. Available: https://doi.org/10.1109/TSE.2019.2945329

[3] F. Michaud and M. Nicolescu, "Behavior-based systems," in *Springer handbook of robotics*. Springer, 2016, pp. 307–328.

[4] D. Kortenkamp, R. Simmons, and D. Brugali, "Robotic systems architectures and programming," in *Springer handbook of robotics*. Springer, 2016, pp. 283–306.

[5] D. Brugali and A. Shakhimardanov, "Component-based robotic engineering (part ii)," *IEEE Robotics & Automation Magazine*, vol. 17, no. 1, pp. 100–112, 2010.

[6] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018.

[7] J. Chen and D. Shi, "Development and composition of robot architecture in dynamic environment," in *International Conference on Robotics, Control and Automation Engineering (RCAE)*, 2018.

[8] F. W. Heckel, G. M. Youngblood, and N. S. Ketkar, "Representational complexity of reactive agents," in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 2010.

[9] D. Isla, "Handling complexity in the Halo 2 AI," *GDC 2005 Proceedings*, 2005. [Online]. Available: https://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php?page=2

[10] K. Mcquillan, "A survey of behaviour trees and their applications for game AI," 2015, course CP5330 final report.

[11] O. Biggar, M. Zamani, and I. Shames, "An expressiveness hierarchy of behavior trees and related architectures," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5397–5404, 2021.

[12] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegwart, and C. Smith, "On the programming effort required to generate behavior trees and finite state machines for robotic applications," *arXiv preprint arXiv:2209.07392*, 2022.

[13] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard *et al.*, "An integrated system for autonomous robotics manipulation," in *International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[14] M. Colledanchise, A. Marzinotto, D. V. Dimarogonas, and P. Ögren, "The advantages of using behavior trees in multi robot systems," in *47th International Symposium on Robotics (ISR)*, 2016.

[15] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on robotics*, vol. 33, no. 2, pp. 372–389, 2016.

[16] M. Colledanchise, R. Parasuraman, and P. Ögren, "Learning of behavior trees for autonomous agents," *IEEE Transactions on Games*, vol. 11, no. 2, pp. 183–189, 2018.

[17] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[18] P. Ögren, "Increasing modularity of uav control systems using computer game behavior trees," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2012.

[19] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[20] S. García, P. Pelliccione, C. Menghi, T. Berger, and T. Bures, "High-level mission specification for multiple robots," in *12th International Conference on Software Language Engineering (SLE)*, 2019.

[21] A. Klöckner, "Interfacing behavior trees with the world using description logic," in *AIAA Guidance, Navigation, and Control Conference (GNC)*, 2013.

[22] E. de Araújo Silva, E. Valentin, J. R. H. Carvalho, and R. da Silva Barreto, "A survey of model driven engineering in robotics," *Journal of Computer Languages*, vol. 62, p. 101021, 2021.

[23] G. L. Casalaro, G. Cattivera, F. Ciccozzi, I. Malavolta, A. Wortmann, and P. Pelliccione, "Model-driven engineering for mobile robotic systems: a systematic mapping study," *Software and Systems Modeling*, pp. 1–31, 2021.

[24] A. Nordmann, N. Hochgeschwender, and S. Wrede, "A survey on domain-specific languages in robotics," in *International conference on simulation, modeling, and programming for autonomous robots*. Springer, 2014, pp. 195–206.

[25] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[26] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, and A. Wasowski, "Behavior trees in action: a study of robotics applications," in *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, 2020, pp. 196–209.

[27] N. Bencomo, R. B. France, B. H. C. Cheng, and U. Aßmann, Eds., *Models@run.time—Foundations, Applications, and Roadmaps*, vol. 8378. Springer, 2014.

[28] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *IEEE Computer*, vol. 42, no. 10, pp. 22–27, 2009.

[29] "Online appendix," https://bitbucket.org/easelab/behaviortrees, 2020. [Online]. Available: https://doi.org/10.5281/zenodo.7515222

[30] M. Colledanchise, "Behavior trees in robotics," Ph.D. dissertation, KTH Royal Institute of Technology, 2017.

[31] A. Wasowski and T. Berger, *Domain-specific Languages: Effective Modeling, Automation, and Reuse*. Springer, 2023. [Online]. Available: http://dsl.design

[32] S. Dragule, T. Berger, C. Menghi, and P. Pelliccione, "A survey on the design space of end-user-oriented languages for specifying robotic

missions," *Software and Systems Modeling*, vol. 20, pp. 1123–1158, 2021.
[33] M. Colledanchise and P. Ögren, "How behavior trees generalize the teleo-reactive paradigm and and-or-trees," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 424–429.
[34] D. C. Conner and J. Willis, "Flexible navigation: Finite state machine-based integrated navigation and control for ros enabled robots," in *SoutheastCon 2017*. IEEE, 2017, pp. 1–8.
[35] S. Macenski, F. Martín, R. White, and J. G. Clavero, "The Marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2718–2725.
[36] J. M. Zutell, D. C. Conner, and P. Schillinger, "Ros 2-based flexible behavior engine for flexible navigation," in *SoutheastCon 2022*. IEEE, 2022, pp. 674–681.
[37] S. Garcia, P. Pelliccione, C. Menghi, T. Berger, and T. Bures, "Promise: High-level mission specification for multiple robots," in *42nd International Conference on Software Engineering (ICSE), Demonstrations Track*, 2020.
[38] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.
[39] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.
[40] "ScenarioRunner for CARLA documentation," https://carla-scenariorunner.readthedocs.io, 2022.
[41] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and AI," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.
[42] M. Colledanchise and L. Natale, "Analysis and exploitation of synchronized parallel executions in behavior trees," *arXiv preprint arXiv:1908.01539*, 2019.
[43] I. Millington and J. Funge, *Artificial intelligence for games*. CRC Press, 2009.
[44] M. L. Crane and J. Dingel, "UML vs. classical vs. rhapsody statecharts: not all models are created equal," *Software & Systems Modeling*, vol. 6, no. 4, pp. 415–435, 2007.
[45] M. von der Beeck, "A comparison of statecharts variants," in *Proc. Third International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, ser. Lecture Notes in Computer Science, H. Langmaack, W. P. de Roever, and J. Vytopil, Eds., vol. 863. Springer, 1994, pp. 128–148. [Online]. Available: https://doi.org/10.1007/3-540-58468-4_163
[46] Object Management Group, "OMG unified modeling language 2.5.1," 2017. [Online]. Available: https://www.omg.org/spec/UML/
[47] B. P. Douglass, "Chapter 5 - design patterns for state machines," in *Design Patterns for Embedded Systems in C*, B. P. Douglass, Ed. Newnes, 2011, pp. 257–356.
[48] M. Samek, "A crash course in UML state machines," *Quantum Leaps, LLC*, 2009.
[49] H. Kubátová, K. Richta, and T. Richta, "Petri nets versus UML state machines," in *Proc. SDOT*, 2013, pp. 53–59.
[50] B. Hajji, A. Mellit, and L. Bouselham, *Finite State Machines*. Singapore: Springer Singapore, 2022, pp. 175–205.
[51] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of computer programming*, vol. 8, no. 3, pp. 231–274, 1987.
[52] G. Lüttgen, M. Von der Beeck, and R. Cleaveland, "A compositional approach to statecharts semantics," *ACM SIGSOFT Software Engineering Notes*, vol. 25, no. 6, pp. 120–129, 2000.
[53] D. Stonier, N. Usmani, and M. Staniaszek, "Py Trees library documentation," https://py-trees.readthedocs.io/en/devel/, 2020.
[54] D. Faconti and M. Colledanchise, "BehaviorTree.CPP library documentation," https://www.behaviortree.dev, 2018.
[55] J. Bohren and S. Cousins, "The SMACH high-level executive [ros news]," *IEEE Robotics & Automation Magazine*, vol. 17, no. 4, pp. 18–20, 2010.
[56] P. Schillinger, S. Kohlbrecher, and O. von Stryk, "Human-robot collaborative high-level control with an application to rescue robotics," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.
[57] P. Schillinger, "An approach for runtime-modifiable behavior control of humanoid rescue robots," *Technische Universitat Darmstadt*, 2015.
[58] S. Kohlbrecher, A. Stumpf, A. Romay, P. Schillinger, O. Von Stryk, and D. C. Conner, "A comprehensive software framework for complex locomotion and manipulation tasks applicable to different types of humanoid robots," *Frontiers in Robotics and AI*, vol. 3, p. 31, 2016.
[59] J. Bohren and S. Cousins, "SMACH library documentation," http://wiki.ros.org/smach/Documentation, 2010.
[60] P. Schillinger, "FlexBE library documentation," http://philserver.bplaced.net/fbe/documentation.php, 2016.
[61] D. Stonier, N. Usmani, and M. Staniaszek, "Py Trees ROS library documentation," https://py-trees-ros.readthedocs.io/en/devel/, 2020.
[62] D. Faconti and M. Colledanchise, "BehaviorTree.CPP library tutorials," https://www.behaviortree.dev/tutorial_01_first_tree/, 2018.
[63] D. Stonier, N. Usmani, and M. Staniaszek, "Py Trees ROS library tutorials," https://py-trees-ros-tutorials.readthedocs.io/en/devel/tutorials.html, 2020.
[64] P. Schillinger, "FlexBE library tutorials," http://wiki.ros.org/flexbe/Tutorials, 2021.
[65] J. Bohren and S. Cousins, "SMACH library tutorials," http://wiki.ros.org/smach/Tutorials, 2021.
[66] D. Faconti, "MOOD2Be: Models and tools to design robotic behaviors," European Union's Horizon 2020 Research and Innovation Programme, 2019. [Online]. Available: https://github.com/BehaviorTree/BehaviorTree.CPP/blob/master/MOOD2Be_final_report.pdf
[67] M. Colledanchise and L. Natale, "On the implementation of behavior trees in robotics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5929–5936, 2021.
[68] A. Romay, S. Kohlbrecher, A. Stumpf, O. von Stryk, S. Maniatopoulos, H. Kress-Gazit, P. Schillinger, and D. C. Conner, "Collaborative autonomy between high-level behaviors and human operators for remote manipulation tasks using different humanoid robots," *Journal of Field Robotics*, vol. 34, no. 2, pp. 333–358, 2017.
[69] P. Schillinger, "An approach for runtime-modifiable behavior control of humanoid rescue robots," Master's thesis, TU Darmstadt, 2015, https://www.sim.informatik.tu-darmstadt.de/publ/da/2015_Schillinger_MA.pdf.
[70] M. AlMarzouq, A. AlZaidan, and J. AlDallal, "Mining GitHub for research and education: challenges and opportunities," *International Journal of Web Information Systems*, 2020.
[71] I. Malavolta, G. A. Lewis, B. Schmerl, P. Lago, and D. Garlan, "Mining guidelines for architecting robotics software," *Journal of Systems and Software*, vol. 178, p. 110969, 2021.
[72] G. Robles, T. Ho-Quang, R. Hebig, M. R. Chaudron, and M. A. Fernandez, "An extensive dataset of UML models in GitHub," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 519–522.
[73] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.
[74] T. Berger and J. Guo, "Towards system analysis with variability model metrics," in *Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems*, 2014, pp. 1–8.
[75] J. A. Cruz-Lemus, M. Genero, and M. Piattini, "Using controlled experiments for validating UML statechart diagrams measures," in *Software Process and Product Measurement*. Springer, 2007, pp. 129–138.
[76] ——, "Metrics for UML statechart diagrams," in *Metrics for Software Conceptual Models*, 2005, pp. 237–272.
[77] R. van Tonder and C. Le Goues, "Lightweight multi-language syntax transformation with parser parser combinators," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2019, pp. 363–378.
[78] S. Garcia, D. Strueber, D. Brugali, A. D. Fava, P. Pelliccione, and T. Berger, "Software variability in service robotics," *Empirical Software Engineering*, 2022.
[79] S. Garcia, D. Strueber, D. Brugali, A. D. Fava, P. Schillinger, P. Pelliccione, and T. Berger, "Variability modeling of service robots: Experiences and challenges," in *13th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, 2019.
[80] D. Brugali and E. Prassler, "Software engineering for robotics [from the guest editors]," *IEEE Robotics & Automation Magazine*, vol. 16, no. 1, pp. 9–15, 2009.
[81] I. A. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I.-H. Shu *et al.*, "CLARAty: Challenges and steps toward reusable robotic software," *International Journal of Advanced Robotic Systems*, vol. 3, no. 1, p. 5, 2006.
[82] M. Colledanchise, "BT++ library documentation," https://github.com/miccol/ROS-Behavior-Tree/blob/master/BTUserManual.pdf, 2017.

[83] F. Rovida, "SkiROS2 library documentation," https://github.com/RVMI/skiros2/wiki, 2020.
[84] RoboSoft AI, "SMACC library documentation," https://smacc.dev/, 2018.
[85] M. Steinbrink, P. Koch, S. May, B. Jung, and M. Schmidpeter, "State machine for arbitrary robots for exploration and inspection tasks," in *Proceedings of the 2020 4th International Conference on Vision, Image and Signal Processing*, 2020, pp. 1–6.
[86] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *Proceedings of the 13th International Conference on Embedded Software*, 2016, pp. 1–10.
[87] F. Rovida, M. Crosby, D. Holz, A. S. Polydoros, B. Großmann, R. Petrick, and V. Krüger, "SkiROS —a skill-based robot control platform on top of ROS," in *Robot operating system (ROS)*. Springer, 2017, pp. 121–160.
[88] F. Rovida and V. Krüger, "Design and development of a software architecture for autonomous mobile manipulators in industrial environments," in *2015 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, 2015, pp. 3288–3295.
[89] P. Laker, "Blackboard design pattern," https://social.technet.microsoft.com/wiki/contents/articles/13215.blackboard-design-pattern.aspx, 2012.
[90] J.-P. Tolvanen and S. Kelly, "How domain-specific modeling languages address variability in product line development: Investigation of 23 cases," in *23rd International Systems and Software Product Line Conference*, ser. SPLC, 2019.
[91] A. Alami, Y. Dittrich, and A. Wasowski, "Influencers of quality assurance in an open source community," in *11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2018.
[92] E. F. Moore *et al.*, "Gedanken-experiments on sequential machines," *Automata studies*, vol. 34, pp. 129–153, 1956.
[93] J. Bosch, "Design patterns as language constructs," *Journal of Object Oriented Programming*, 1997.
[94] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE software*, vol. 31, no. 3, pp. 79–85, 2013.
[95] J. A. Cruz-Lemus, A. Maes, M. Genero, G. Poels, and M. Piattini, "The impact of structural complexity on the understandability of UML statechart diagrams," *Information Sciences*, vol. 180, no. 11, pp. 2209–2220, 2010.
[96] M. Genero, D. Miranda, and M. Piattini, "Defining and validating metrics for UML statechart diagrams," *Proceedings of QAOOSE*, vol. 2002, 2002.
[97] J. A. Cruz-Lemus, M. Genero, M. Piattini, and A. Toval, "Investigating the nesting level of composite states in UML statechart diagrams," *Proc. QAOOSE*, vol. 5, pp. 97–108, 2005.
[98] ——, "An empirical study of the nesting level of composite states within UML statechart diagrams," in *International Conference on Conceptual Modeling*. Springer, 2005, pp. 12–22.
[99] D. Brugali and P. Scandurra, "Component-based robotic engineering (part i)[tutorial]," *IEEE Robotics & Automation Magazine*, vol. 16, no. 4, pp. 84–96, 2009.
[100] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An exploratory study of cloning in industrial software product lines," in *17th European Conference on Software Maintenance and Reengineering (CSMR)*, 2013.
[101] L. Gherardi and D. Brugali, "Modeling and reusing robotic software architectures: The hyperflex toolchain," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6414–6420.
[102] C. Schlegel, A. Lotz, M. Lutz, D. Stampfer, J. F. Inglés-Romero, and C. Vicente-Chicote, "Model-driven software systems engineering in robotics: covering the complete life-cycle of a robot," *it-Information Technology*, vol. 57, no. 2, pp. 85–98, 2015.
[103] D. L. Wigand, A. Nordmann, M. Goerlich, and S. Wrede, "Modularization of domain-specific languages for extensible component-based robotic systems," in *2017 First IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2017, pp. 164–171.
[104] SPARC Robotics, "Robotics 2020 multi-annual roadmap for robotics in Europe," *SPARC Robotics, EU-Robotics AISBL, The Hauge, The Netherlands*, 2017.

Razan Ghzouli is currently working towards a PhD degree at Chalmers University of Technology, Gothenburg, Sweden, where she is part of the software engineering division at the Department of Computer Science and Engineering. She received her master degree in applied data science from the university of Gothenburg, Sweden and her bachelor degree in computer and automation engineering from Damascus university, Syria. Her PhD focuses on facilitating the migration to model-based design and systems to enable reusable and maintainable robotic missions.

Thorsten Berger is a Professor in Computer Science at Ruhr University Bochum in Germany. After receiving the PhD degree from the University of Leipzig in Germany in 2013, he was a Postdoctoral Fellow at the University of Waterloo in Canada and the IT University of Copenhagen in Denmark, and then an Associate Professor jointly at Chalmers University of Technology and the University of Gothenburg in Sweden. He received competitive grants from the Swedish Research Council, the Wallenberg Autonomous Systems Program, Vinnova Sweden (EU ITEA), and the European Union. He is a fellow of the Wallenberg Academy—one of the highest recognitions for researchers in Sweden. He received two *best-paper* and two *most-influential-paper* awards. His service was recognized with distinguished reviewer awards at the tier-one conferences ASE 2018 and ICSE 2020. His research focuses on model-driven software engineering, program analysis, and empirical software engineering.

Einar Broch Johnsen is a Professor at the Department of Informatics of the University of Oslo in Norway. He is the strategy director of Sirius, a center for research-driven innovation with long-term funding from the Research Council of Norway. He has been prominently involved in many national and European research projects; in particular, he was the coordinator of the EU FP7 project Envisage (2013-2016) on formal methods for cloud computing and the scientific coordinator of the EU H2020 project HyVar (2015-2018) on hybrid variability systems. His research focuses on formal methods, programming models and methodology, and program analysis in domains such as distributed and concurrent systems, cloud computing, digital twins and robotics.

Andrzej Wasowski is Professor of Software Engineering at the IT University of Copenhagen. He has also worked at Aalborg University in Denmark, and as visiting professor at INRIA Rennes and University of Waterloo, Ontario. His interests are in software quality, reliability, and safety in high-stake high-value software projects. This includes semantic foundations and tool support for model-driven development, program analysis tools, testing tools and methods, as well as processes for improving and maintain quality in software projects. Many of his projects involve commercial or open-source partners, primarily in the domain of robotics and safety-critical embedded systems. Recently he coordinates the Marie-Curie training network on Reliable AI for Marine Robotics (REMARO). Wasowski holds a PhD degree from the IT University of Copenhagen, Denmark (2005) and a MSC Eng degree from the Warsaw University of Technology, Poland (2000).

Swaib Dragule is a PhD Fellow in Computer Science and Software Engineering at Chalmers University of Technology and Makerere University. He holds MSc. and BSc. in computer science. He is an academic staff of Makerere university, College of Computing and Information Sciences. His research interests are in programming languages, domain-specific languages, and robotics.