RPL：一种基于反应式Agent的机器人编程语言

田昌海 杨 硕 陈 寅 毛新军
（国防科技大学计算机学院 长沙 410073）

摘 要 开放环境下的机器人具有环境敏感性、行为自主性和并发性、反应实时性等特点，这对支撑这类机器人的控制软件及其编程语言提出了新的要求，包括支持对环境进行显式表示，支持自主和并发的行为，需要对行为间在时间、空间、物理上的关系进行规约等等。面向Agent的编程语言将软件系统的基本执行单元视为自主的软件Agent，它为机器人控制软件的构造提供了新的方法和思路。针对开放环境下机器人特点对其编程语言的要求，提出了基于反应式Agent的编程模型RECA和编程语言RPL。RECA将单个机器人的软件系统视为一个反应式Agent，它包括SensorEvent、EventRule和ScenarioBehaviour 3个组成部分，其中SensorEvent是对机器人所处环境信息变化的一种封装；ScenarioBehaviour是对机器人的不同行为进行的规约；EventRule定义了机器人环境输入到行为输出的动态绑定关系。RPL提供了一系列的机制来支持机器人控制软件的编程，包括事件机制、多线程机制、优先级描述、行为动态绑定。最后介绍了RPL程序开发和运行支撑环境的技术框架，并基于NAO机器人分析了机器人作为老人生活助理的案例，验证了该编程模型、语言和运行支撑环境的有效性。

关键词 机器人，控制软件，面向Agent编程

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.3.003

RPL：A Robot Programming Language Based on Reactive Agent

TIAN Chang-hai YANG Shuo CHEN Yin MAO Xin-jun
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract Robots situated in open environment are expected to be context-aware and autonomous, to support concurrent behaviours, and to react to external stimuli in a timely manner. To program such robots, a robot programming language is required to explicitly represent environmental factors, support decision-making and concurrency, specify temporal, special and physical relations among robot behaviours, and prioritize concurrent robot behaviours to avoid collision. Agent-oriented programming (AOP) provides a new solution to robot programming by taking software units as autonomous agents. Based on the requirements for robot programminges language, agent-oriented programming model—RECA and programming language—RPL based on reactive agent were proposed. RECA programming model which takes robot software as a reactive agent consists of three elements. SensorEvent shows environmental changes; ScenarioBehaviour are the different behaviour specifications for robots, and EventRule defines the dynamic mapping relations from environmental inputs to behavioral outputs. RPL was designed to meet the needs of robot programming, by providing various mechanisms supporting event-based programming, multi-thread programming, prioritization of robot behaviours, and dynamic binding of robot behaviours. We designed and implemented a programming and runtime environment for the RPL, and demonstrated the expressiveness of RPL and the effectiveness of its runtime environment through a case study of elder assistant robot.

Keywords Robots, Control software, Agent-oriented programming

1 引言

当前机器人的应用领域正变得越来越广，由工业领域扩展到了智能家居、医疗服务、救灾、探月、军事等领域。应用领域的变化对机器人的研制提出了一系列的挑战：机器人所处的环境由几乎封闭、静态，可控的工业环境变成了开放、动态、难控的复杂环境；单个机器人所面临的任务也由单一任务转变为多样化的任务。要在这种开放的、动态的环境中完成给

到稿日期：2014-04-03 返修日期：2014-07-23 本文受国家自然科学基金(61379051)资助。

田昌海(1989－)，男，硕士生，CCF学生会员，主要研究方向为面向Agent的软件工程、面向机器人的软件工程，E-mail：jamestch@163.com；杨 硕(1992－)，男，主要研究方向为面向Agent的软件工程；陈 寅(1988－)，男，博士生，CCF学生会员，主要研究方向为面向机器人的软件工程、面向Agent的软件工程；毛新军(1970－)，男，博士，教授，CCF会员，主要研究方向为面向Agent的软件工程、自适应软件。

定的任务，就需要机器人具有环境敏感性、行为自主性和并发性、反应实时性等特点。除了先进的硬件设施外，机器人的控制软件是保证机器人在上述环境下完成任务的关键。如何针对开放环境下的机器人的特点来有效地支持机器人控制软件的开发，对传统的机器人软件开发方法提出了挑战。

传统的机器人软件开发方法通常采用面向过程和面向对象的范型。面向过程的范型存在抽象层次较低、程序代码重用性较低等缺点；而面向对象的范型虽然为数据和算法提供了较高层次的封装，但作为一种被动的实体，对象仍无法很好地描述软件实体的自主性。上述特点使得这些范型在直接用于开放环境下机器人控制软件的开发时存在诸多不足。近年来，面向 Agent 的范型引起了人们的关注。Agent 对软件实体进行了更高层次的封装，每个 Agent 除了具有一般的数据（状态）与算法（行为），还具有反应性、自主性、社会性等特点[3,15]。反应性是指 Agent 在运行过程中可以实时响应其外部环境的变化或内部自身状态的变化；自主性是指 Agent 可以根据它感知到的环境信息模型进行自主决策，以决定当前应该执行何种行为；社会性是指 Agent 可以与其它 Agent 进行交互的能力。面向 Agent 的范型的这些特点为开放环境下机器人控制软件的开发提供了新的思路。

面向 Agent 的范型将软件系统的基本执行单元视为自主的软件 Agent。目前面向 Agent 的编程所支持的个体 Agent 软件模型主要有 3 种：反应型（reactive）、慎思型（deliberative）以及混合型。慎思型 Agent 通过推理（reasoning）、规划（planning）等人工智能方法，有效地支持了 Agent 基于逻辑进行行为决策，但通常效率有限，制约了机器人行为的实时性；混合型 Agent 综合了慎思型和反应型两种 Agent 模型的优点，试图平衡 Agent 的自主性和反应性，但也带来了如何取得最佳平衡点的问题。因此，本文将单个机器人的软件系统视为单纯的反应式 Agent，提出一种基于反应式结构的面向 Agent编程模型 RECA，并基于此，提出了面向 Agent 的机器人编程语言 RPL，来解决开放环境下机器人的控制软件的开发问题。

本文第 2 节阐述开放环境下机器人的特点对其编程语言提出的要求；第 3 节介绍基于反应式 Agent 的机器人编程模型和语言；第 4 节阐述机器人程序开发和运行支撑环境的整体技术架构，并通过案例分析验证编程模型和语言的可行性；第 5 节对国外内机器人编程语言的相关研究工作进行对比分析；最后进行总结、并指出下一步研的究工作。

2 开放环境下机器人的特点及其对编程语言的要求

本节将结合具体的案例来对开放环境下机器人的特点进行分析、说明，进而总结其对机器人编程语言的要求。

2.1 案例：老年人生活助理

本案例以 NAO 机器人为平台（关于 NAO 机器人的介绍详见文献[16]），实现了开放家居环境下机器人作为老年人生活助理的实际案例。本案例中，机器人需要完成以下几个场景的任务：

T1：NAO 每天早上 9:00 点提醒老人按时吃药。

T2：每隔一定时间间隔，在房间内“巡视”检查是否有垃圾，如果发现垃圾则将垃圾捡起送到垃圾桶内。完成清理垃圾的过程中 NAO 需要自己避开障碍。

T3：机器人电量过低时进行语音提醒，同时返回“加油站”进行充电。

T4：当 NAO 听到老人“Help!”的语音后，应该立即通知其家人。本案例中，通过按房间内特定的红色按钮来实现通知其家人的行为。

2.2 开放环境下的机器人的特点

结合 2.1 节中的案例，开放环境下机器人的特点分析如下。

· 环境的不确定性（contextual uncertainty）

开放环境中的各个要素都处在持续变化中，且这种变化具有不确定性和不可预知性。案例中 T2 场景就体现了环境的开放性：该场景下垃圾的出现是随机的，是不能事先预知的；障碍物的出现也是无法事先完全预知的，因为可能会随机出现移动中的障碍物，如行人等。

· 对环境的敏感性（context-awareness）

在开放环境下要完成既定任务，就需要机器人具有自主决策的能力，而机器人进行自主决策的前提是机器人能感知所处的环境信息和自身的状态信息，即要求机器人具有对开放环境信息敏感的特性。案例中机器人完成既定任务都离不开对环境信息进行感知。

· 自主性（autonomy）

要在开放环境中顺利完成给定的任务，就必须让机器人的行为具有自主性，即机器人根据感知的环境或自身状态信息，能在运行时自主地动态改变、调整自身的行为，以适应环境从而更好地完成既定任务。案例中 T2 场景下，机器人需要根据环境变化自主地调整自身的行为：发现垃圾，机器人对垃圾进行清理；在清理过程中遇到障碍物，机器人需要将自身行为调整为避障；避开障碍后，机器人继续对垃圾进行清理。

· 并发性（concurrency）

机器人要完成给定任务，可能需要两个或以上的行为同时进行，即机器人行为存在并发性的特点。案例中，机器人需要同时完成 T1 和 T2 的任务。在 T3 中，机器人在电池电量过低的情况下需要同时执行语音提示和充电两个行为。

· 物理限制（physical constraint）

机器人的行为大都需要实际的物理执行机构来执行实现，各种物理执行机构会存在时间、空间、资源等诸多方面的限制和要求。因此，如果不对机器人的并发行为作出一定限制规约，它们之间就可能会存在冲突。受物理限制，案例中机器人无法在电池电量低的情况下完成清理垃圾的任务，即 T2 和 T3 中的行为不能并发执行，且 T3 的优先级应该比 T2 高。

· 实时反应性（real-time reactivity）

对于不同的环境变化或自身状态变化，机器人对实时性的要求也是不一样的。如案例中机器人对老年人求救的响应行为的优先级应该最高，该行为对反应的实时性要求最高。

2.3 机器人编程语言要求

开放环境下机器人的特点对其编程语言提出了如下要求。

• 对环境进行显式表示

机器人身上大多具有丰富的传感器，如超生波传感器、红外线传感器、扬声器、麦克风、压力传感器、摄像头等，机器人控制软件通过调用机器人提供的标准函数接口可以获取机器人各个传感器的数据流。但这些原始数据流不能直接用于控制软件自主决策，还需要对传感器的数据流进行处理和封装。通常情况下，机器人执行某一行为往往需要多个传感器的数据或多方面的信息。因此，机器人的编程语言需要提供一种处理、封装这些传感器数据的机制，能显式地表示机器人所处环境中的变化，以方便机器人作出各种行为决策。

• 支持自主决策

机器人的编程语言应该为其行为的自主决策提供机制支持，使机器人控制软件能够根据不同的环境自主地选择不同的行为执行，以适应相应的环境。

• 支持并发性编程

机器人的编程语言应该支持多线程编程，为机器人行为的并发性提供机制支持。

• 对优先级进行描述

机器人的编程语言应该能根据时间、空间和资源等对不同机器人行为的物理限制，对不同机器人行为的优先级进行描述，从而有效避免机器人行为之间的冲突，同时体现不同行为对实时性的不同要求。

3 基于反应式 Agent 的机器人编程模型与语言

3.1 基于反应式 Agent 的机器人编程模型和运行机制

机器人的控制软件是控制机器人完成任务的关键，它需要感知环境信息，然后根据环境信息进行行为决策，最后执行相应的行为。机器人控制软件的 Agent 模型如图 1 所示。本文将单个机器人的控制软件视为一个反应式 Agent。

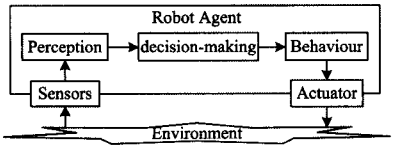

图 1 机器人控制软件的 Agent 模型

基于反应式 Agent 的机器人编程语言 RPL 主要包括 3 个要素：SensorEvent、EventRule、ScenarioBehaviour。RPL 的编程模型如图 2 所示。其中 SensorEvent 主要是用于监控机器人所处环境的变化，这种环境变化包括机器人自身状态的变化和机器人所处的物理环境的变化两个方面；ScenarioBehaviour是对机器人在不同场景下行为进行的规约；Event-Rule用来决定某 Event 发生时应该加入或退出某个 Scenario-Behaviour。

图 2 RPL 编程模型 RECA

基于反应式 Agent 的机器人编程不仅仅是简单地将反应式 Agent 的 ECA 模型引入到程序设计层中，还要针对机器人对编程语言的要求引入相应的支持机制。基于 RECA 模型，RPL 引入了如下机制：

• 基于 SensorEvent 的编程机制

编程者可以根据机器人所要完成的任务对机器人的传感器信息进行处理和封装，从而自定义 SensorEvent 以显式表示机器人所处环境的变化。这些 SensorEvent 通过回调函数的机制和 RPL 程序完成整合。如果用户自定义了某个 SensorEvent，那么一旦环境发生变化将致使该 SensorEvent 发生，程序就会去执行该 SensorEvent 对应的回调函数以应对这种变化。

• EventRule 的多线程机制

EventRule 中的 EventActions 部分定义了 SensorEvent 回调函数中应该执行的动作。这些 EventRule 之间是一种异步关系，每个 SensorEvent 发生后，都会激活一个新的线程去处理该 SensorEvent 触发的 EventRule。

• EventRule 优先级描述机制

EventRule 还定义了各自的优先级参数，通过优先级来对机器人行为之间的关系进行规约。这为机器人行为物理性导致的冲突提供了解决机制；同时也为机器人行为的不同实时性要求提供了支持。

• ScenarioBehaviour 的动态绑定机制

ScenarioBehaviour 是机器人在不同场景下行为的规约。机器人在不同的环境和状态下，可以通过 join、quit、suspend、resume 4 种基本操作来实现与不同机器人行为 ScenarioBehaviour的动态绑定。这种动态行为绑定机制在一定程度上为机器人行为的自主性提供了机制支持。

3.2 RPL 语言的语法

RPL 语言的程序由一系列的事件定义（SensorEvent）、行为定义（ScenarioBehaviour）和规则（EventRule）组成，其 EBNF 定义如下：

Prog::={SensorEvent}{ScenarioBehaviour}{EventRule}

• SensorEvent

SensorEvent 的 EBNF 语法定义如表 1 所列。

表 1 SensorEvent 部分语法

1	⟨SensorEvent⟩	::="@"⟨EventName⟩ ["["⟨inputParams⟩"]"]
2		["(" outputParams ")"]"{"⟨EventStatements⟩ "}"
3	⟨inputParams⟩	::= ⟨inputParam⟩ {"," ⟨inputParam⟩}
4	⟨outputParams⟩	::=⟨outputParam⟩ {"," ⟨outputParam⟩}
5	⟨EventStatements⟩	::= {⟨RobotStatement⟩}+

其中，EventName 为用户自定义的 SensorEvent 的名字，且每个 EventName 前面都有一个标识符"@"。inputParams、outputParams 分别表示该 SensorEvent 的输入参数和输出参数。RobotStatement 为机器人底层提供的基本编程语句。SensorEvent 主要用于封装、处理机器人的传感器信息，通过事件显式地表示环境信息。

• EventRule

EventRule 的 EBNF 语法定义如表 2 所列。

表 2 EventRule 部分语法

1	⟨EventRule⟩	::=["#(" ["＞(" ⟨EventNames⟩ ")";"] ["＜("
2		⟨EventNames⟩ ")"]")"] "@" ⟨EventName⟩ ["("
3		logical_expression ")"] "{" ⟨RuleStatements⟩ "}"
4	⟨EventNames⟩	::=⟨EventName⟩ {"," ⟨EventName⟩}
5	⟨RuleStatements⟩	::={⟨RuleStatement⟩}$^+$
6	⟨RuleStatement⟩	::=⟨EventAction⟩ | ⟨EventRule⟩
7	⟨EventAction⟩	::=("join" | "quit" | "suspend" | "resume")"$"
8		⟨ScenarioName⟩)

其中，EventName 为用户自定义的 SensorEvent 的名字。Logical_expression 是表示某些 SensorEvent 发生后可能还需要满足一些条件才能执行该 EventRule 对应的动作，它是可以缺省的。通常情况下，不同的 EventRule 是在不同的线程异步执行的。EventRule 中"#(＜(EventNames);＞(EventNames))"是该规则的优先级描述参数："#"为优先级参数的标识符；＜(EventNames)表示括号中事件触发的 EventRule 的优先级比当前 EventRule 高；＞(EventNames)表示括号中事件触发的 EventRule 的优先级比当前 EventRule 低。SensorEvent 触发 EventRule 后会先对相应的优先级作比较，如果有比该Event-Rule优先级低的动作在执行，则机器人会终止低优先级的行为，转而执行优先级高的行为；如果有比该 Event-Rule优先级高的动作在执行，则该规则失效，即不能触发相应的动作执行。从表 2 中第 6 行可以看出，EventRule 中可以嵌套 EventRule，这为子 ScenarioBehaviour 的描述执行提供了语言层面的机制支持。

• ScenarioBehaviour

ScenarioBehaviour 的 EBNF 语法定义如表 3 所列。

表 3 ScenarioBehaviour 部分语法

1	⟨ScenarioBehaviour⟩	::="$"⟨ScenarioName⟩["["⟨inputParams⟩"]"]
2		["("outputParams ")"] "{"⟨Statements⟩"}"
3	⟨inputParams⟩	::=⟨inputParam⟩ {"," ⟨inputParam⟩}
4	⟨outputParams⟩	::=⟨outputParam⟩ {"," ⟨outputParam⟩}
5	⟨Statements⟩	::={⟨RobotStatement⟩}$^+$

其中，ScenarioName 为用户自定义的机器人行为规约 ScenarioBehaviour 的名字，且每个 ScenarioName 前面都有一个标识符"$"。inputParams、outputParams 分别表示执行该行为所需要的输入参数和执行该行为后的输出参数。RobotStatement 为机器人底层提供的基本编程语句。

4 开发和运行支撑环境及案例分析

4.1 RPL 程序的开发和运行支撑环境

机器人控制软件的开发和运行支撑环境的整体技术架构如图 3 所示。

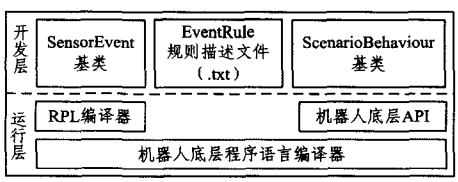

图 3 机器人软件系统的开发和运行支撑环境的技术架构

• 开发层：提供了基类 SensorEvent 和 ScenarioBehaviour。开发者通过实例化基类 SensorEvent 来自定义感兴趣的 SensorEvent，以 SensorEvent 的发生与否来显式地表示环境信息的变化；通过实例化 ScenarioBehaviour 基类来封装机器人的行为，将不同的行为规约到不同的 ScenarioBehaviour 实例中。EventRule 文件用于描述不同 SensorEvent 实例到不同 ScenarioBehaviour 实例的映射关系。

• 运行层：通过 RPL 编译器对 RPL 程序进行整合编译，同时与机器人底层提供的编程语句整合链接起来；最后通过机器人底层编程语言的编译器编译成可执行代码执行。

4.2 案例分析

本节采用 RECA 编程模型和 RPL 编程语言对 2.1 节中描述的机器人作为老年人生活助理的案例进行了分析和实现。该案例中基于 SensorEvent 的 ScenarioBehaviour 迁移如图 4 所示。

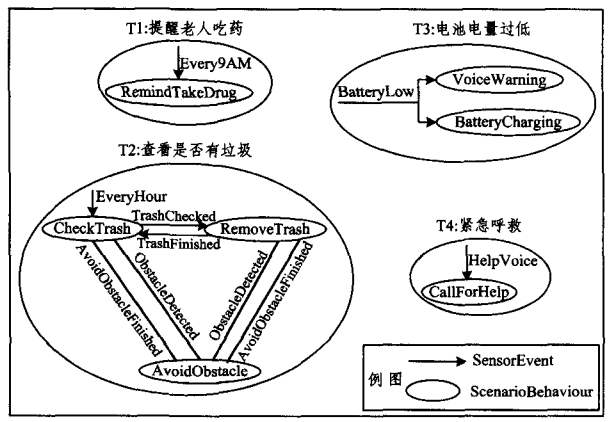

图 4 基于 SensorEvent 的 ScenarioBehaviour 迁移图

案例中，T1、T4 场景下机器人行为单一，编程实现较为简单。T2 场景下，机器人需要根据环境变化自主地调整自身的行为：(1)EveryHour 触发行为规约 CheckTrash。(2)如果 TrashChecked 发生(即发现垃圾)，则 suspend 行为 CheckTrash，join 行为 RemoveTrash；垃圾处理完后即 TrashFinished 发生，quit 行为 RemoveTrash，resume 行为 CheckTrash。(3)在行为规约 RemoveTrash 和 CheckTrash 中，若 ObstacleDetected 发生，机器人应该 suspend 当前行为，join 行为规约 AvoidObstacle；避障完成即 AvoidObstacleFinished 发生，则机器人 quit 行为规约 AvoidObstacle，resume 之前 suspend 的行为。T3 场景下的 SensorEvent 是电池电量过低 BatteryLow，它同时触发了两种行为规约：(1)VoiceWarning，用来控制机器人发出"BatteryLow"的语音警告；(2)BatteryCharging，用来控制机器人返回"加油站"充电。两个行为并发执行。上述任务或行为之间还存在着一些要求和限制：

• T1、T2 要能同时完成；

• T3 无法与 T1、T2 同时完成，其优先级比 T1、T2 高；

• T4 的优先级最高。

该案例的具体实现代码：案例中的 SensorEvent 和 ScenarioBehaviour 根据其 EBNF 语法由 NAO 底层提供的 C++ 编程语句进行封装，由于代码较长，不再列出；EventRule 代码部分如表 4 所列。

表 4　EventRule 代码

```
#(<(BatteryLow,HelpVoice)) @Every9AM                    //T1 场景下
    {
    join $RemindTakeDrug;
    }
#(<(BatteryLow,HelpVoice)) @EveryHour                   //T2 场景下
    {
    join $CheckTrash;
    @TrashChecked
        {
        suspend $CheckTrash;
        join $RemoveTrash;
        @ObstacleDetected
            {
            suspend $RemoveTrash;
            join $AvoidObstacle;
            }
        @AvoidObstacleFinished
            {
            quit $AvoidObstacle;
            resume $RemoveTrash;
            }
        }
    @TrashFinished
        {
        quit $RemoveTrash;
        resume $CheckTrash;
        @ObstacleDetected
            {
            suspend $RemoveTrash;
            join $AvoidObstacle;
            }
        @AvoidObstacleFinished
            {
            quit $AvoidObstacle;
            resume $RemoveTrash;
            }
        }
    @ObstacleDetected
        {
        join $AvoidObstacle;
        }
    }
#(<(HelpVoice)>(Every9AM,EveryHour)) @BatteryLow        //T3
    {
    join $VoiceWarning;
    }
#(<(HelpVoice)>(Every9AM,EveryHour)) @BatteryLow
    {
    join $BatteryCharging;
    }
#(>(Every9AM,EveryHour,HelpVoice)) @HelpVoice           //T4
    {
    join $CallForHelp;
    }
```

5　机器人编程语言的相关研究工作

当前，机器人的主流编程语言主要分为两类：传统的高级计算机语言和专门面向机器人的编程语言。这些机器人编程语言都有各自的特点，但在用于开放环境下的机器人编程时都存在一定的问题。

传统高级计算机语言，比如 C/C++、Java、Python 等，经常被用于机器人的编程[5-7]。但这些高级计算机语言本来并不是专门为开发机器人程序而设计的。机器人是物理信息融合体，要完成既定任务，离不开与环境进行交互。机器人程序需要能够感知环境中的变化进而决定采取何种行为来应对这种变化。为了达到这一目的，采用传统高级计算机编程语言来开发机器人程序往往需要对这类编程语言做一些机制扩展[8]。

同时也出现了很多专门面向机器人的编程语言，比如 Urbiscript、KUKA、RoboLogix 等。Urbiscript 是一种动态的、面向对象的脚本语言，它支持并行编程和基于事件的编程(event-based programming)[9]。但它是一种无类型语言，无法做静态类型检查；不能对事件之间的时序性约束提供支持；不能在运行时支持机器人行为的自主、动态调整。KUKA 是由 KUKA 公司开发的一种面向工业机器人的编程语言[10]，工业机器人所处的封闭、可控的环境决定了该语言不适合用于开放环境下的机器人编程。RoboLogix 是一种动作级的脚本语言，它由一系列的指令组成，每个指令对应一个或多个动作。RoboLogix 程序由数据对象和指令集合组成。这种语言编程比较简单，不能接受复杂的传感器信息，不能很好地将各种算法融合在一起，缺乏与环境进行交互的机制，不能对机器人行为的自主性提供支持[11]。近年来，事件驱动编程(event-driven programming)由于提供了一种良好的与环境交互的机制，在机器人程序开发、自适应系统开发等方面得到了较多的使用。文献[4,12,13]提出了几种基于事件驱动的编程语言。但这些语言仍存在一些不足，如不支持机器人行为的并发性，无法动态改变机器人的行为以应对环境变化等。文献[14]开发了机器人编程语言 HAWRL，但该语言面向特定机器人应用——弧焊接的机器人编程语言。上述机器人编程语言的特点总结如表 5 所列。文献[2]回顾了机器人编程语言的发展历史，分析了机器人编程语言的特点，并按编程层次将机器人的编程语言分为机器人级语言和任务级语言两类。文献[1]总结了当前 AOP 语言在用于自主机器人的编程时需要做的改进或扩展。

表 5　机器人编程语言对比

机器人编程语言	显式表示环境	行为自主决策	并发性编程	对行为优先级进行描述
C/C++、Java、Python			√	
Urbiscript KUKA、RoboLogix	√		√	
INI、EventScript	√	√		
RPL	√	√	√	√

结束语　开放环境下机器人的特点对机器人的编程语言提出了新的挑战。为了解决这一问题，提出了基于反应式结构的面向 Agent 机器人的编程模型和语言。该语言支持基于事件的编程模式，开发者可以根据机器人所面临的任务自定义 SensorEvent 以显式表示机器人所处环境的变化。RPL 程序用不同的线程处理不同的 EventRule，很好地支持了机器人行为的并发性特点。RPL 语言在编程中考虑了这些行为之间的时序性关系和优先级关系，从而有效地避免了行为之间的冲突，同时考虑了机器人行为对实时性的不同要求。更重要的是，RPL 语言引入了机器人行为的动态绑定机制，将机器人的行为决策放到了运行时。机器人可以根据环境变化动态地加入或退出不同的行为规约，进而执行不同的行为以适应相应的环境。

本研究工作处于刚起步的阶段，下一步的工作包括：(1)本文仅对对 RPL 语言的可行性进行了验证，未对其运行效率方面进行深入研究，后续工作将在程序运行效率方面与其他

机器人编程语言进行对比分析，并以此对 RPL 语言的语法和机制进行改进和完善；(2)本文是将单个机器人当作一个 Agent来考虑，未能充分利用 Agent 的社会性等特性，后续工作将尝试用多 Agent 技术来考虑单个机器人的控制软件开发问题以及考虑多机器人系统的软件开发问题；(3)本文基于反应式 Agent 来考虑机器人的编程问题，但反应式 Agent 模型仅仅是通过对事件处理机制即反应式规则来支持个体Agent 的自主决策，因而其自主决策能力非常有限。考虑采用Agent 的其他软件模型(如慎思型、混合型等)来支持机器人的编程以提高机器人自主决策的能力将是下一步研究的重点之一。

参 考 文 献

[1] Ziafati P, Dastani M, Meyer J J, et al. Agent Programming Languages Requirements for Programming Autonomous Robots [M]// Programming Multi-Agent Systems. Springer Berlin Heidelberg, 2013: 35-53

[2] 戴齐，姚先启. 机器人程序设计语言[J]. 机器人，1997，19(5)：390-400

[3] Jennings N R, Sycara K, Wooldridge M. A roadmap of agent research and development[J]. Autonomous agents and multi-agent systems, 1998, 1(1): 7-38

[4] Le T G, Fedosov D, Hermant O, et al. Programming Robots with Events [M] // Embedded Systems: Design, Analysis and Verification. Springer Berlin Heidelberg, 2013: 14-25

[5] Auyeung T. Robot programming in "C" [OL]. 2006-02-15 [2014-07-01]. http://wild-puppy.drtak.org/teaches/ARC/cisp299_bot/b-ook/book.pdf

[6] Kang S C, Chang W T, Gu K Y, et al. Robot Development Using Microsoft Robotics Developer Studio[M]. CRC Press, 2011

[7] Preston S. The definitive guide to building Java robots [M]. Apress, 2006

[8] Jayaram K R, Eugster P. Context-oriented programming with Event-Java[C] // International Workshop on Context-Oriented Programming. ACM, 2009: 9

[9] Baillie J C. Urbi: Towards a universal robotic low-level programming language[C] // Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'05). 2005: 820-825

[10] Chinello F, Scheggi S, Morbidi F, et al. Kuka control toolbox [J]. Robotics & Automation Magazine, IEEE, 2011, 18(4): 69-79

[11] Logic Design Inc. Robologix[OL]. 2014-06-28[2014-07-01]. http://www.robologix.com/programming_robologix.php

[12] Cohen N H, Kalleberg K T. EventScript: an event-processing language based on regular expressions with actions[J]. ACM Sigplan Notices, 2008, 43(7): 111-120

[13] Holzer A, Ziarek L, Jayaram K R, et al. Putting events in context: aspects for event-based distributed programming[C] // Proceedings of the tenth international conference on Aspect-oriented software development. ACM, 2011: 241-252

[14] 张连新，高洪明，张广军，等. 混合式弧焊机器人编程语言[J]. 焊接学报，2006，27(7)：105-108

[15] 毛新军. 面向 Agent 软件工程：现状，挑战与展望[J]. 计算机科学，2011，38(1)：1-7

[16] Aldebaran Robotics. Nao software documentation[OL]. [2014-07-02]. https://community.aldebaran.com/doc/

(上接第 12 页)

[26] Viyanon W, Madria S K. A system for detecting xml similarity in content and structure using relational database[C] // Proceedings of the 18th ACM Conference on Information and Knowledge Management. HongKong, China, 2009: 1197-1206

[27] Li Pei, Dong X L, Maurino A, et al. Linking temporal records [C] // Proceedings of the 37th International Conferenceon Very Large Data Bases (VLDB 11). Seattle, Washington, USA, 2011

[28] 王宏志，樊文飞. 复杂数据上的实体识别技术研究[J]. 计算机学，2011，38(10)：1843-1852

[29] 杨丹，申德荣，于戈，等. 数据空间中时间为中心的集合实体识别策略[J]. 计算机科学与探索，2012，39(11)：1673-9418

[30] Papadakis G, Ioannou E, Palpanas T, et al. A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces[J]. IEEE Trans. Knowl. Data Eng. (TKDE), 2013, 25(12): 2665-2682

[31] Papadakis G, Ioannou E, Niederée C, et al. Efficient entity resolution for large heterogeneous information spaces[C] // WSDM 2011. 2011: 535-544

[32] de Vries T, Ke H, Chawla S, et al. Robust Record Linkage Blocking Using Suffix Arrays[C] // Proc. 18th ACM Conf. Information and Knowledge Management (CIKM). 2009: 305-314

[33] Jin L, Li C, Mehrotra S. Efficient Record Linkage in Large Data Sets[C] // Proc. Eighth Int'l Conf. Database Systems for Advanced Applications (DASFAA). 2003

[34] Baxter R, Christen P, Churches T. A Comparison of Fast Blocking Methods for Record Linkage[C] // Proc. Workshop Data Cleaning, Record Linkage and Object Consolidation at SIGKDD. 2003: 25-27

[35] Gravano L, Ipeirotis P, Jagadish H, et al. Approximate String Joins in a Database (Almost) for Free[C] // Proc. 27th Int'l Conf. Very Large Data Bases (VLDB). 2001: 491-500

[36] Kalashnikov D V, Mehrotra S. Domain-independent data cleaning via analysis of entityrelationship graph[J]. ACM Trans. Datab. Syst., 2006, 31(2): 716-767

[37] Nuray-Turan R, Kalashnikov D V, Mehrotra S. Adaptive connection strength models for relationship-based entity resolution [J]. Journal of Data and Information Quality, 2013, 4(2)

[38] Yakout M, Elmagarmid A K, Elmelegy H, et al. Behavior based record linkage[C] // Proceedings of VLDB. 2010

[39] Whang S E, Garcia-Molina H. Entity Resolution with Evolving Rules[J]. Proceeding of the VLDB Endowment, 2013 (1/2): 1326-1337

[40] Ramadan B, Christen P, Liang Hui-zhi, et al. Dynamic Similarity-Aware Inverted Indexing for Real-Time Entity Resolution[C] // Trends and Aplplications in Knowledge Discovery and Data Mining. Gold Coast Australia, Volume 7867, 2013: 47-58

[41] Altwaijry H, Kalashnikov D V, Mehrotra S. Query-Driven Approach to Entity Resolution [J]. PVLDB, 2013, 6(14): 1846-1857

[42] Kenig B, Gal A. MFIBlocks: An effective blocking algorithm for entity resolution[J]. Inf. Syst. (IS), 2013, 38(6): 908-926

[43] 王颖颖，黄杜英，许多顶. 向量空间中基于隐私保护的记录链接协议[J]. 现代电子技术，2009，32(14)：138-141