ATLANTIS PRESS

Advances in Engineering Research, volume 86

2nd International Conference on Electrical, Automation and Mechanical Engineering (EAME 2017)

RPL: A Robot Programming Language Based on Reactive Agent

Xinyang Wang[1] and Jian Zhang[2]

[1]Guangdong Vocational Technology Institute, China

[2]Foshan Southern Insitute of Data Science and Technology, China

***Abstract*—According to the requirements for robot programmings language, agent oriented programming model-RECA and programming language-RPL based on reactive agent were proposed. It is defined that 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 behaviors, and dynamic binding of robot behaviors. Furthermore It is designed and implemented a programming development and technical framework of run time environment for the RPL. This paper verify the feasibility of the programming model and language by case analysis. To conduct comparative analysis on the foreign and domestic researches related to RPL for further research is pointed in the conclusion.**

Keywords-robots; control software; agent oriented programming

I. INTRODUCTION

Currently, robots are applied in more widely fields, which vary from the industrial field to smart home, medical service, disaster relief, lunar exploration, military, etc. The change in the application fields has posed a series of challenges for the R&D of robots: the environment of robots is almost closed and static and the controllable industrial environment has become open, dynamic and uncontrollable; the single task faced by a single robot becomes diversified. The complication of tasks in such open and dynamic environment calls for the robots to have context-awareness, behavior autonomy and concurrency as well as real-time reactivity. In addition to advanced hardware, the control software is the key to make robots complete tasks in the above environment. To effectively support the development of robots' control software based on the features of robots in the open environment poses a challenge to the traditional software development method of robots.

The traditional robot software development methods usually adopt the process-oriented paradigm and object-oriented paradigm. The process-oriented paradigm has the disadvantages of low level of abstraction and low procedure code reusability, etc.; the object-oriented paradigm can provide high level of encapsulation for data and algorithm, object which is a kind of passive entity cannot effectively describe the autonomy of the software entity. The above features bring many deficiencies for the paradigms when directly used in open environment for the development of robot control software. In recent years, the Agent-oriented paradigm has attracted extensive attention. Agent can provide higher level of encapsulation to software entity. In addition to the general data (state) and algorithm (behavior), each Agent is featured with reactivity, autonomy and sociality, etc. [3]. Reactivity refers to the real-time response of Agent in operation to the change of external environment or the change of internal state; autonomy means that Agent can make independent decision according to the perceived environmental information model in order to determine the behavior it should execute currently; sociality stands for the interaction capability between the Agent and other Agents. The features of Agent-oriented paradigm provide a new thought for the robot software development in the open environment.

The Agent-oriented paradigm takes the basic execution unit of software system as autonomous software Agent. Currently, there are three kinds of individual Agent software models supported by Agent oriented programming (AOP), namely reactive, deliberative and mixed Agents. The deliberative Agent can effectively support the behavior decision-making based on logic by reasoning, planning and other artificial intelligence methods. But its limited efficiency restricted the real-time of robot behaviors. The mixed Agent, which is integrated with the advantages of both the deliberative Agent and reactive Agent, attempts to balance the autonomy and reactivity of Agent, but how to get the best balance point is a new problem to be solved. Therefore, this paper takes the software of a single as pure reactive Agent and proposes RECA, a kind of Agent-oriented programming model based on the reactive structure. Based on which the RPL, a robot programming language based on reactive agent, is put forward to solve the development issue of robot control software in the open environment.

The section 2 explains the requirements by features of robots in open environment for programming language; section 3 introduces the robot programming model and language based on reactive Agent; section 4 describes the overall technical architecture of robot programming development and run time infrastructure and verify the feasibility of the programming model and language by case analysis; section 5 conducts comparative analysis on the foreign and domestic researches related to RPL; finally, further research is pointed in the conclusion.

II. Features of Robots in Open Environment and Requirements for Programming Language

This section will analyze and explain the features of robot in open environment with specific case and draw a conclusion on the requirements for programming language.

A. Case: Life Assistant of the Aged

In this case, the NAO robot (see literature [16] for the detailed introduction to NAO robot) realized the life assistant of the aged in open home environment. The robot needs to complete the tasks in the following scenarios:

T1: remind the aged to take medicine at 9:00 am every morning;

T2: make an inspection tour to check if there is rubbish in the room for a certain time interval. If any, send the rubbish to the garbage can. NAO needs to avoid the obstacles by itself in the cleaning process;

T3: when the power is too low, NAO needs to make voice prompt and go back to the "filling station" for charging;

T4: when hearing the voice "Help" cried by the aged, NAO needs to promptly notify the family members of the aged. In this case, the notice is achieved by pressing the specific red button in the room.

B. Features of Robot in Open Environment

Based on the case in 2.1, the features of robot in open environment are as follows:

1) Contextual uncertainty

The factors in open environment are constantly changing with uncertainty and unpredictability. T2 scenario in the case reflects the openness of environment: the occurrence of rubbish and obstacle is random and unpredictable for that the moving obstacle such as people may appear randomly.

2) Context awareness

In order to complete the given tasks in open environment, the robot needs to have independent decision-making ability. The precondition of independent decision-making is the awareness of context information and the status information of itself, that is, the robot should have the feature of context awareness. All the tasks in the case require the robot to have the context awareness.

3) Autonomy

To smoothly complete the given tasks in open environment, the behavior of robot must have autonomy, that is, the robot can autonomously and dynamically change and adjust behaviors according to the perceived context or status information in order to adapt the environment and better complete the given tasks. In the T2 scenario of the case, the robot needs to independently adjust its behavior according to the changes of environment: find clean rubbish; the robot needs to adjust its behavior into "obstacle avoidance" to avoid the obstacle in the cleaning process; after avoid the obstacle, the robot continue to clean the rubbish.

4) Concurrency

The robot may need to simultaneously conduct two or more behaviors in order to complete the given tasks, that is, the behaviors of robot have concurrency. In the above case, the robot needs to complete the tasks in T1 and T2 at the same time. In T3, the robot needs to execute the voice prompt and charging simultaneously under low power.

5) Physical constraint

Most behaviors of robots need to be achieved by actual physical actuators which may have the constraints and requirements in time, space, resource and many other aspects. Therefore, if no constraints and specifications are made on the concurrent behaviors of the robot, conflicts may arise. With the physical constraint, the robot in the case is unable to finish the cleaning tasks under low power situation, that is, the behaviors in T2 and T3 cannot be executed concurrently and the priority of T3 is higher than that of T2.

6) Real-time reactivity

The requirements for real-time reactivity of robot vary with the changes in environment or self-status. In the case, the reactivity behavior of robot to the calling for help should have the highest priority, and this behavior has the highest requirement for real time reactivity.

C. Requirements for RPL

The requirements of robot's features in open environment for programming language are as follows:

1) Explicit expression on environment

Most robots have rich sensors such as ultrasonic sensor, infrared sensor, loudspeaker, microphone, pressure sensor and camera, etc. The robot control software may obtain the data flows of various sensors by transfer the standard function interfaces. But those data flows cannot be directly used by the control software for independent decision-making which needs to process and encapsulate the data flows of sensors. Generally, to execute certain behavior, the robot tends to obtain the data of multiple sensors or the information in multiple aspects. Therefore, the RPL needs a mechanism to process and encapsulate the data of these sensors to explicitly express the changes of the robot environment so that the robot can make various behavior decisions.

2) Support independent decision-making

The RPL should provide mechanism support for the independent decision-making of behaviors so that robot control software can autonomously choose different behavior executions according to different environments in order to adapt the corresponding environment.

3) Support concurrent programming

The RPL should support multithread programming to provide mechanism support for the concurrency of robot behaviors.

4) Description of priority

The RPL should be able to describe the physical constraint and priority of different behaviors according to the time, space and resource, etc. in order to effectively avoid the conflicts between robot behaviors and reflect the different requirements of behaviors for real-time reactivity.

III. THE ROBOT PROGRAMMING MODEL AND LANGUAGE BASED ON REACTIVE AGENT

A. The Robot Programming Model and Operation Mechanism Based on Reactive Agent

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

Robot control software is the key to control the robot to complete tasks, which needs to aware perceive the environmental information and make behavior decisions according to the environment information and finally execute the corresponding behaviors. The Agent model of robot control software is shown as Figure 1. This paper takes the control software of a single robot as a reactive Agent.

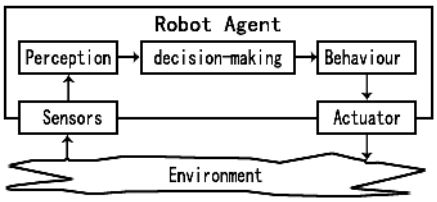

FIGURE I. AGENT MODEL OF ROBOT CONTROL SOFTWARE

The RPL based on reactive Agent mainly includes 3 elements: SensorEvent, EventRule and ScenarioBehaviour. SensorEvent is mainly used to monitor the changes of the environment of the robot, including the changes in the status of the robot and the physical environment changes; ScenarioBehaviour are the different behavior specifications for robots in different scenarios, and EventRule is used to determine to add or exit a certain ScenarioBehaviour when an Event occurs.

The robot programming based on reactive Agent not only import the ECA model of reactive Agent into the programming design level, but also introduce the corresponding support mechanism according to the requirements of robot for programming language. Based on the RFCA model, RPL introduces the following mechanism:

1) Programming mechanism based on SensorEvent

The programmer processes and encapsulates the sensor information of robot based on the tasks need to be completed by the robot and thus self-define SensorEvent to explicitly express the environment changes. These SensorEvents will complete the integration through the callback function mechanism and RPL procedure. If a certain SensorEvent is defined by a user, any environment change will cause the occurrence of the SensorEvent, and the procedure will execute the callback function corresponding to the SensorEvent to response such change.

2) Multiple-threads architecture of EventRule

The EventActions in EventRule define the actions that should be executed in the callback function of SensorEvent. Asynchronous relationship exists among these EventRules. The occurrence of every SensorEvent may activate a new thread to process the EventRule triggered by the SensorEvent.

3) Priority description mechanism of EventRule

EventRules also define the respective priority parameter. The priority specifies the relationship among the robot behaviors, which provides the settlement mechanism for the conflict caused by the physical property of robot behavior and provides support for the different real time requirements of the robot behaviors.

4) Dynamic binding mechanism of ScenarioBehaviour

ScenarioBehaviour is the behavior specification of robot in different scenarios. In different environments and statuses, robot can realize the dynamic binding with different robot ScenarioBehaviours by four basic operations, namely joint, quit, suspend and resume. Such dynamic binding mechanism provides the mechanism support for the autonomy of robot behavior in some degree.

B. The Grammar of RPL

The RPL procedure is consisting of a series of SensorEvent, ScenarioBehaviour) and EventRule, the EBNF is designed as follows:

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

SensorEvent: The EBNF grammar of SensorEvent is defined as Table 1.

TABLE I. PARTIAL GRAMMAR OF SENSOREVENT

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

In which, EventName is the name of SensorEvent defined by user, and there is an identifier "@" before every EventName. InputParams and outputParams respectively refer to the input parameters and output parameters. RobotStatement stands for the basic programming statement provided at the low-level of robot. SensorEvent is mainly used to encapsulate and process the sensor information of robots and explicitly express the environment information through the event.

EventRule: The EBNF grammar of EventRule is defined as Table 2

TABLE II. PARTIAL GRAMMAR OF EVENTRULE

```
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>)
```

EventName stands for the name of user-defined SensorEvent. Logical expression refers to the action corresponding to the EventRule that can be executed upon meeting some conditions after the occurrence of some SensorEvents, which can be default. Normally, different EventRules are asynchronously executed in different thread. In the EventRule, # (＜(EventNames);＞(EventNames)) refers to the priority description parameters of the rule: "#" is the identifier of the priority description parameter; ＜(EventNames) means that the priority of the Event Rule triggered by the event in the bracket is higher than the current EventRule; ＞(EventNames) means that the priority of the Event Rule triggered by the event in the bracket is lower than the current EventRule. After triggering an EventRule, the SensorEvent will compare the corresponding priority. If there is certain lower priority action is being executed, the robot will stop the action with lower priority and continue to execute the action with higher priority; if there is any action with higher priority than that of the EventRule is being executed, such rule will be invalid. That is, the corresponding action execution cannot be triggered. The 6th line in Table 2 shows that an EventRule can be nested into another EventRule, which provides the language level mechanism support for the description execution of the sub-ScenarioBehaviour.

·ScenarioBehaviour: The EBNF grammars of ScenarioBehaviour are shown in Table 3.

TABLE III. PARTIAL GRAMMARS OF SCENARIOBEHAVIOUR

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

ScenarioName refers to the user-defined name of ScenarioBehaviour, and there is an identifier "$" before every ScenarioName. InputParams and outputParams respectively stand for the input parameters and output parameters needed to execute the behavior. RobotStatement represents the basic programming statement provided at the low-level of robot.

IV. DEVELOPMENT AND RTI AS WELL AS CASE ANALYSIS

A. *The Development and RTI of RPL Procedure*

The overall technical architecture of the development and RTI of robot control software is shown in Figure 2.

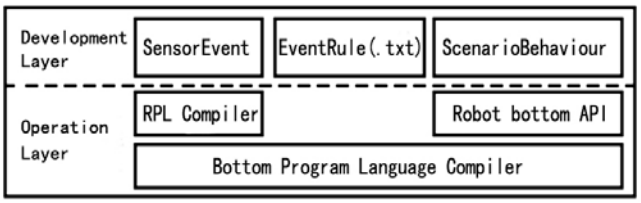

FIGURE II. OVERALL TECHNICAL ARCHITECTURE OF THE DEVELOPMENT AND RTI OF ROBOT CONTROL SOFTWARE

Development layer: base classes SensorEvent and ScenarioBehaviour are provided. The developer customizes the interesting SensorEvent by the instantiated base class SensorEvent and explicitly expresses the changes in environment information according the occurrence of SensorEvent. Different behaviors are specified into different ScenarioBehaviour cases by encapsulating the robot behaviors through the instantiation of ScenarioBehaviour base classes. EventRule files are used to describe the mapping relation between different SensorEvent cases and different ScenarioBehaviour cases.

Runtime layer: integrate and compile the RPL procedure with the RPL compiler, and integrate with the robot low-level programming language; execute the executable code compiled by the compiler of the robot low-level programming language.

B. *Case Analysis*

This section adopts RECA and RPL to analyze and realize the case in section 2.1 where the robot is served as the living assistant.

In this case, the behaviors in T1 and T4 are simple, and the programming realization is easy. In T2, the robot needs to independently adjust its behaviors according to the environment changes: (1) EveryHour triggers behavior specification CheckTrash. (2) If TrashChecked occurs (trash detected), suspend CheckTrash, and join RemoveTrash; after remove the trash (TrashFinished), quit RemoveTrash and join CheckTrash. (3) In the behavior specifications RcmoveTrash and CheckTrash, if ObstacleDetected occurs, the robot should suspend current behavior and join the behavior specification AvoidObstacle; when obstacle avoidance is finished, the robot will quit the behavior specification AvoidObstacle and resume the previous suspended behavior. In T3, the SensorEvent is BatteryLow, which triggered two behavior specifications: (1) VoiceWarning, used to control the robot to send "BatteryLow" voice warning; (2) BattcryCharging, used to control the robot to go back to the "charging station" for charging. The two behaviors are executed concurrently. There are some requirements and limitations between the above tasks or behaviors:

1) T1 and T2 need to be completed simultaneously;

2) T3 has a higher priority than T1 and T2, which cannot be completed simultaneously with T1 and T2;

3) T4 has the highest priority.

The specific realization code in the case: the SensorEvent and ScenarioBehaviour are encapsulated by the C++

programming statements provided by NAO low level according to the EBNF grammar. Because the code is too long, it will not be listed here。

V. RESEARCHES RELATED TO RPL

Currently, the mainstream programming languages are divided into two classes: the traditional advanced computer language and the robot-oriented programming language. Those robot programming languages have their own features, but all of them have certain problems when used to program the robot in open environment. The traditional advanced computer languages such as C/C++, Java and Python are usually used in robot programming [5-7]. But those traditional advanced computer languages are not specially designed for the development of robot procedure. As the integration of physical information, the robot cannot complete the given tasks without the interaction with environment. Robot procedure needs to be able to perceive the changes in the environment to determine the behavior to respond such changes. To achieve this objective, some mechanism extension needs to be conducted when traditional advanced computer programming languages are used to develop robot procedures .

Meanwhile, many robot-oriented programming languages such as Urbiscript, KUKA and RoboLogix are emerged. Urbiscript is a kind of dynamic object-oriented script language, which supports parallel programming and event-based programming [9]. But it is an untyped language which cannot conduct static checks, provide support for time sequence constraint between events and support the autonomous and dynamic adjustment of robot behaviors in the running process. KUKA is a kind of industrial robot-oriented programming language developed by KUKA Company. The closed and controllable environment of industrial robots decides that such language is not suitable for the robot programming in open environment. RoboLogix is a kind of operative script language consists of a series of instructions with each instruction corresponding one or more actions. RoboLogix procedure is consisting of data objects and instruction set. This language programming is simple and cannot receive complicated sensor information; cannot be integrated with various algorithms; lacks the interaction mechanism with environment; and cannot provide support for the autonomy of robot behaviors [1]. In recent years, event-driven programming is widely used in robot program development, adaptive system development and other aspects for that it provides a good interaction mechanism with the environment. Literatures [4] propose several event-driven programming languages, but these languages still have some deficiencies. For instance, they neither support the concurrency of robot behaviors, nor have the ability to dynamically change the robot behaviors to respond the environment changes. Literature [4] developed HAWRL, a robot programming language for the specific robot application-arc welding. The features of the above said robot programming languages are concluded as Table 5. Literature [2] reviews the development of robot programming language, analyzes the features of robot programming languages and classifies the robot programming languages into robot-level language and task-level language according to programing levels. Literature [1] concludes the improvement and extension need to be made when the existing AOP languages are used in autonomous robot programming.

TABLE IV. ROBOT PROGRAMMING LANGUAGE

Programming	Display representation environment	Behavioral decision making	Concurrent programming	Priority description
C/C++、Java、Python			✓	
Urbiscript	✓		✓	
KUKA、RoboLogix				
INI、EventScript	✓	✓		
RPL	✓	✓	✓	✓

VI. CONCLUSION

The features of robot in open environment pose new challenges to robot programming languages. To solve this problem, the Agent robot programming model and language based on reactive structure is proposed. The language supports the event-based programming model which allows developers to customize SensorEvent according to the tasks faced by robot in order to explicitly express the changes in the robot environment. RPL procedure can process different EventRules with different threads, which well supports the concurrency of robot behaviors. RPL language takes the time sequence relation and priority relation between those behaviors in to consideration and thus effectively avoids the conflicts between the behaviors. Meanwhile, it also considers the different real-time requirements of robot behaviors. More importantly, RPL introduces the dynamic binding mechanism of robot behaviors and place the robot behavior decision-making in the running process. Robots can dynamically join or quit different behavior specifications based on the environment changes and further execute different behaviors in order to adapt the corresponding environment.

This research is just in the starting stage, and the next work includes: (1) because this paper mainly verified the feasibility of RPL languages but does not conduct in-depth study on the running efficiency, the comparative analysis between the program running efficiency and other robot programming languages will be included in the follow-up work and further enhance and improve the grammar and mechanism of RPL language; (2) because this paper only takes sing robot as an Agent and does not make full use of the sociality and other features of Agent, more Agent technologies will be attempted in the follow-up work in order to consider the control software development issues of single robot and the software development issues of multi-robot system; (3) the robot programming problems are analyzed based on reactive Agent which only supports the independent decision-making of individual Agent through event handling mechanism (reactive rule), thus its independent decision-making ability is very limited. Therefore, one of the next research priorities will be to improve the robot independent decision-making ability by supporting the robot programming with other software models (such as deliberative and mixed models) of Agent.

REFERENCES

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

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

[4] Mao Xinjun. Agent-oriented Software Engineering Status, Challenge and Outlook [J]. Computer Science, 2011,38(1):1-7

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

[6] Li Pei, Dong XL , Maurino A, et al. Linking temporal records[C]//Proceedings o1 the 37th International C'onferenceon Very LargeData Bases (VLDB 11). Seattle,Washington,USA,2011

[7] Wang Hongzhi, Fan Wenfei. Research on Entity Identification Technology on Complex Data [J]. Computer Science, 2011,38(10):1843-1852

[8] Yang Dan, Jin Derong, Yuge, et al. Tim-centered Set Entity Identification Strategy in Data Space [7]. Computer Science and Exploration, 2012,39(11):1673-9418