IEICE TRANS. INF. & SYST., VOL.E101–D, NO.7 JULY 2018

PAPER

AutoRobot: A Multi-Agent Software Framework for Autonomous Robots*

Zhe LIU[†a)], **Xinjun MAO**[†b)], ***Nonmembers***, ***and*** **Shuo YANG**[†c)], ***Student Member***

SUMMARY Certain open issues challenge the software engineering of autonomous robot software (ARS). One issue is to provide enabling software technologies to support autonomous and rational behaviours of robots operating in an open environment, and another issue is the development of an effective engineering approach to manage the complexity of ARS to simplify the development, deployment and evolution of ARS. We introduce the software framework *AutoRobot* to address these issues. This software provides abstraction and a model of accompanying behaviours to formulate the behaviour patterns of autonomous robots and enrich the coherence between task behaviours and observation behaviours, thereby improving the capabilities of obtaining and using the feedback regarding the changes. A dual-loop control model is presented to support flexible interactions among the control activities to support continuous adjustments of the robot's behaviours. A multi-agent software architecture is proposed to encapsulate the fundamental software components. Unlike most existing research, in *AutoRobot*, the ARS is designed as a multi-agent system in which the software agents interact and cooperate with each other to accomplish the robot's task. *AutoRobot* provides reusable software packages to support the development of ARS and infrastructure integrated with ROS to support the decentralized deployment and running of ARS. We develop an ARS sample to illustrate how to use the framework and validate its effectiveness.

key words: *autonomous robot, multi-agent system, AutoRobot*

1. Introduction

With the continuous integration of robotics and information technologies, such as the Internet, artificial intelligence, big data, and service computing, more robots are being developed to operate in open, unknown and unpredictable environments (e.g., family, hospital, and battlefield) and are expected to autonomously behave without human intervention to achieve assigned tasks [1]. Typically, we call such robots autonomous robots. Essentially, an autonomous robot is a system in which software plays an important role. Autonomous robot software (ARS) typically operates to (1) manage and control physical devices (e.g., arms, motors, and legs) of robots and (2) decide and plan robot behaviours and drive robots to act.

The complexity of ARS derives from its diversity and integrity of software elements, the dynamic evolution of software structures and behaviours to adapt to various changes occurring in the environment and itself, and specific requirements including high-level behaviour decisions, flexible adjustments and adaptation, safety, and real-time response [2]. With the rapid development of robot technologies and the increasing demands on robot applications, the development of a complex software system for an autonomous robot has become an open issue in the fields of robotics and software engineering.

In the past years, substantial effort has been focused on the research studies and practices of ARS. Many software technologies have been proposed to support the development of ARS, including programming languages [3], [4], software architecture [5], [6], and design patterns. In the academic and industry field, dozens of software frameworks [4], [7], [8] have been proposed. According to the technologies adopted, the existing software frameworks can be roughly classified into two categories: component-based approaches [9], [10] and agent-based approaches [11]–[14]. The component-based approach focuses on the construction of software elements of ARS and their inter-operations and aims to simplify the development and promote the reuse of software component technologies. As all of the behaviour decisions are encapsulated into software components, the component-based approach seldom considers the autonomous decisions and flexible adjustment problems of ARS. The agent-based approach focuses on improving autonomy for robots, in which ARSs are modelled as autonomous agents that interact with the environment and exhibit autonomous behaviours. However, most previous studies have taken ARS as a single software agent exhibiting coarse granularity, resulting in challenges to reuse and self-management, and the proposed technologies have focused on the agent hood and lack consideration of the integration of robot technologies with mainstream software engineering technologies.

In essence, as a domain-specific software, ARS poses certain challenges. One issue is to provide enabling software technologies to support autonomous and rational behaviours of robots operating in the open environment. Another issue is the development of an effective engineering approach to manage the complexity of ARS, thereby simplifying the development, deployment and evolution of ARS. We propose the software framework *AutoRobot* to address these issues. Several important software engineering technologies can be used to enhance the framework. (1) Cognition and abstraction: the use of accompanying behaviour as

Manuscript received November 24, 2017.
Manuscript revised March 3, 2018.
Manuscript publicized April 4, 2018.

[†]The authors are with National University of Defense Technology, China.

*This work was supported by National Nature and Science Foundation of China under Granted No. 61532004.

a) E-mail: liuzhe16@nudt.edu.cn
b) E-mail: xjmao_nudt@163.com
c) E-mail: yangshuo@nudt.edu.cn

DOI: 10.1587/transinf.2017EDP7382

the model to describe the complex behaviour patterns and robot interactions. (2) Control: the use of a dual control loop as the control model to enrich the interactions, feedback, adjustment and adaption. (3) Development: the use of a multi-agent system as a software architecture to support the encapsulation, reuse, and deployment of software elements and implement autonomy and rationality functionalities. The *AutoRobot* framework provides reusable packages and infrastructure to support the development, deployment and operation of the ARS.

The rest of this paper is organized as follows: Sect. 2 discusses the software engineering challenges of ARS based on the analysis of its features with a motivating example. Section 3 presents the design objectives for the ARS framework and the critical software engineering technologies used to support the framework. Section 4 provides an overview of *AutoRobot*. Section 5 introduces the software development and running supports provided by *AutoRobot*. Section 6 illustrates our approach with a case study. Section 7 discusses related works. Conclusions are drawn in Sect. 8.

2. Features and Software Engineering Challenges for ARS

This section analyses the features of ARS and discusses the software engineering challenges based on a motivated example of autonomous robot application.

2.1 A Motivation Example

Let us consider a domestic service robot example to motivate our research (see Fig. 1). The service robot operates in an open home environment with several rooms and is designed to care for elderly inhabitants by identifying whether they have fallen and thereby providing the necessary information services, e.g., calling an ambulance or notifying the family. As the elderly person moves from one place to another, the service robot should follow the person at a safe distance to timely perceive her/his moving information, determine her/his safety status, and accept her/his service requests.

- *Scenario 1 (searching for the elderly person)*: the robot should search for the target by moving through the rooms and recognizing her/his body features. If the target is lost, then the robot repeats the search for the target.

- *Scenario 2 (following the elderly person)*: when finding the target, the robot follows the target when she/he moves. The robot should follow the target closely to avoid losing the target.

- *Scenario 3 (lock on to the elderly person)*: when there is someone else next to the target, the robot should lock on to the target and avoid interference from others.

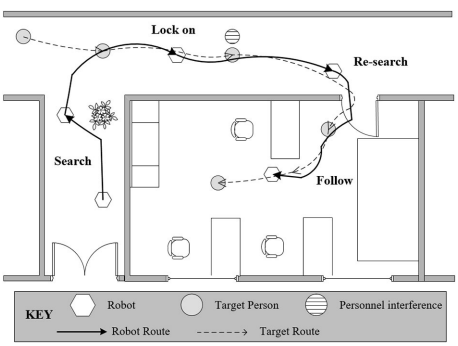

Fig. 1 The motivating example of an autonomous robot

2.2 Features and Requirements of ARS

From this example, we find that autonomous robots are designed to perform tasks independently or with limited external control [3]. We also find from this example that autonomous robots are different from traditional industrial robots and other cyber-physical systems. ARS is the core of autonomous robots and provides support for the implementation of autonomous robot tasks. We list below distinct features of the ARS that depend on the characteristics of autonomous robots:

- *Perform autonomous and rational behaviours.* Autonomous robots typically have limited resources and external control. Autonomous robots must achieve their objects via their own plans and must meet the demands of the external and internal environments. Furthermore, autonomous robots must interact with an unknown environment and sometimes with human beings, as in our example. These behaviours raise the issue that autonomous robots' behaviours should be autonomous and rational, as determined by the ARS.

- *Support robots operating in an open environment.* Autonomous robots are generally situated in open environments that involve physical, social and cyber factors. The environment is also diverse, evolving, unpredictable, and even partially obscured. As in our example, the robot cannot predict when obstacles will appear, and the environment around the robot can be expected to change with the movement of the elderly inhabitants. To this end, the ARS should provide support for autonomous robots to operate in such environments.

- *Increasing demands and complexity.* The development and popularization of autonomous robots results in increasing demands for the ARS to satisfy various and personalized requirements of autonomous robot applications. Furthermore, increasing expectations exist regarding software (non-)functional features, such as intelligence and autonomy, in the ARS. Based on these increasing demands, the ARS integrates multiple software components and software structures to satisfy various demands, making the ARS more complex.

Because of the differences from traditional robotic software,

Table 1 The requirements of autonomous robot software.

Levels	Functional requirements	Non-functional requirements
High-level: Task&Application	plan, schedule, allocate, decide, execute, cooperate	real-time, adaptive, autonomous, safe, rational, friendly
Low-level: Devices&Service	control, analyse, sense, manage, compute	real-time, secure, self-management, accurate, robust

ARS has relatively specific requirements based on its distinct features. Generally, ARS addresses two types of requirements at different levels in Table 1.

2.3 Open Issues for ARS

Based on the features and requirements of ARS, open issues exist in three levels of ARS:

- *Abstraction.* When autonomous robots perform a task, multiple behaviours should be executed. For different tasks, the required behaviours are also different. Current development methods of ARS lack general cognitions and abstractions, which depend on the behavioural characteristics and divide behaviours into different categories. Based on these cognitions and abstractions, the relationship between the behaviours of an autonomous robot can be intuitively described, and the complexity in planning and performing behaviours is simplified. Thereby, ARS can make decisions, adjust behaviours and control robots in a relatively autonomous and rational manner.

- *Control.* As discussed in the last subsection, autonomous robots are situated in an open and dynamic environment and should obtain multi-feedback to perceive environmental changes timely. Moreover, the autonomous robot must monitor self-changes, such as behaviour execution states or electricity information. Based on these changes, ARS can adjust and update robots' behaviours in real-time to adapt to the environment and to themselves to achieve as many of their objectives as possible. This approach requires the ARS to provide an effective control model, which should include sensitive feedback and perception methods. In addition, this model should support the behaviour patterns of autonomous robots.

- *Development.* In the last subsection, we proposed that the ARS becomes more complex because of the increasing demands. For example, when we design and develop an ARS, the existence of many factors, such as the heterogeneity problem, tends to increase the process complexity. Simplifying the design and development for the ARS is a challenge that requires a general software engineering method. This method should provide an effective encapsulation (such as reusable software packages) and integration among software components; furthermore, the method should support control models and provide effective mechanisms in ARS.

3. Design Objectives and Critical Technologies for Autonomous Robot Software Framework

In this section, we present the design objectives for the ARS framework and, based on these objectives, propose three critical technologies: (1) the accompanying behaviour provides an applicable cognition and abstraction for behaviours for ARS and provides support for the realization of autonomy and rationality, (2) the D-SMPA control model improves the reactivity of autonomous robots through enhancement of the feedback capability, and (3) multi-agent software architecture abstracts different parts of the ARS into agents and allows for better system integration.

3.1 Design Objectives for Autonomous Robot Software Framework

We hypothesize that developing a suitable software development framework for ARS is a satisfactory means to solve the open issues. This framework should provide an effective and convenient software development method to develop ARS that meets the design requirements. In the development and construction of this software framework, we suggest that the following design objectives apply:

- *Supporting the realization of autonomy and rationality for robots.* The autonomy and rationality of robots are reflected primarily by the robotic behaviours. Unfortunately, few proper theories of robot behavioural patterns exist that provide appropriate abstractions of the behaviours and describe the intelligent cooperation of these behaviours to achieve the purpose of autonomy and rationality. Therefore, we hypothesize that the development of an ARS framework requires a proper behaviour pattern to implement an improved description of autonomous robot behaviours. Under this behaviour pattern, we can design the decision and allocation mechanism in a top-down fashion for ARS, thereby providing support for the realization of autonomy and realization.

- *Enhancing the reactive and flexible adjustment of behaviours.* The capability to perceive changes is the basis for autonomous robots to achieve autonomy and rationality. Only by being more sensitive to changes, the autonomous robots can better adapt to the environment and make timely behaviour adjustments. For improving the capability of perception, the ARS framework should provide a robot control model with a universal feedback mechanism to strength the connection between robots and not only the environment but also the components in robots.

- *Integration and interaction.* ARS is a type of hybrid software system that is constructed with numbers of components and systems. Determining how to integrate these components and provide favourable interactions between them is a critical problem that our framework should

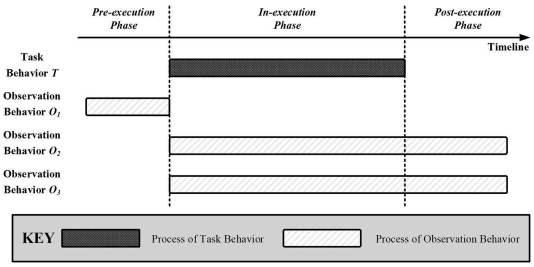

Fig. 2 Three phases in accompanying behaviour

solve. We might choose a layered software framework that classifies various types of components or systems into different layers according to their functions and a proper software model for ARS to solve the distribution and integration of different components. In addition, the framework must provide tools, middleware and mechanisms to support the interaction between different layers and different components.

3.2 Accompanying Behaviour for Autonomous Robot Software

The behaviours taken by an autonomous robot are diverse in responsibilities when accomplishing tasks. Some of the behaviours are responsible for observing the environment and recording changes, whereas others are critical activities to achieve tasks. Therefore, the behaviours of an autonomous robot can be abstracted into two categories according to their responsibilities: one category is task behaviour typically performed by actuators, e.g., arms, legs, and motors; the other category is observation behaviour typically performed by sensors or probes, e.g., sonar and cameras. Although the two categories of behaviours are typically performed in an independent and autonomous manner, the ARS should enhance their synergy when achieving tasks so that task behaviours can obtain on-demand feedback to adjust, optimize and self-manage the planned behaviours.

We refer to this synergistic relationship between observation behaviours and task behaviours as accompanying behaviour that appears as a behaviour pattern. The accompanying behaviour is a cognitive-level technology that provides abstractions of behaviours to support the realization of autonomy and rationality. The accompanying behaviour defines the occasions in which an accompanying relationship must be established; these occasions can be summarized as three phases (as illustrated in Fig. 2):

(i) *Pre-execution phase*: Before the execution of the robotic tasks, the system should ensure that the environment is suitable for the execution. For this purpose, certain specific observation behaviours are executed before the formal behaviours of the task to observe the surroundings and determine whether the required conditions are met as expected. In Fig. 2, the observation behaviour O_1 is in the pre-execution phase of the task behaviour T. For example, in *Searching Scenario* of our motivation example, before robot starts to search for the target, the bumper sensor or cliff sensor (observation behaviour) will help the power driving device (task behaviour) to determine whether the surrounding states satisfy the moving conditinos.

(ii) *In-execution phase*: When the robot is performing a task-oriented behaviour with a planned step, the observation behaviours should check the run-time safety conditions during the execution of the task behaviours and provide feedback to the robotic system and task behaviours. Furthermore, the task behaviours determine which observation should be performed. In Fig. 2, the observation behaviours O_2 and O_3 are in the in-execution phase of the task behaviour T. For example in *Searching Scenario*, when robot is searching for the target, the RGB and depth sensors will help the robot avoid the obstacles.

(iii) *Post-execution phase*: After execution of the task-oriented behaviour, the expected effects of the plan must be compared to the actual effects in a real-world environment that result from the plan execution. In that case, the corresponding observation behaviour is invoked immediately after execution of the task behaviour to observe the related environmental information and determine whether the expected effects have been fully achieved. In Fig. 2, the observation behaviours O_2 and O_3 lie in the post-execution phase of the task behaviour T. For example, after *Searching Scenario*, the RGB sensor is responsible to ensure whether the target is in robot's view and the depth sensor is responsible to determine whethe there is obstacle in front of the robot. Note that observation behaviours can be in different phases of a task behaviour for different responsibilities.

Based on accompanying behaviour, we have identified a set of behaviour patterns which describe the accompanying relations between robot task-achieving and observing activities and can be viewed as a type of running modes. These behaviour patterns specify different coordination and scheduling schemes between robot task and observation behaviors, each of which copes with different tasks or environment situations. And the system-level behaviours can deliver different behaviour patterns to task behaviours and observation behavious to tell them how to accompany during execution of tasks. At present, we have identified three types of behavior patterns, including accident pattern, observation behaviors cooperation pattern, and priority pattern.

(i) *Accident pattern*: Accident pattern is designed for some emergencies in runtime for ARS and it occurs between one observation behaviour and one task behaviour. Under this pattern, for accident that may occur in the environment, task behaviour will arrange corresponding observation behavior to detect it. When accident occurs, the observation behaviour will send

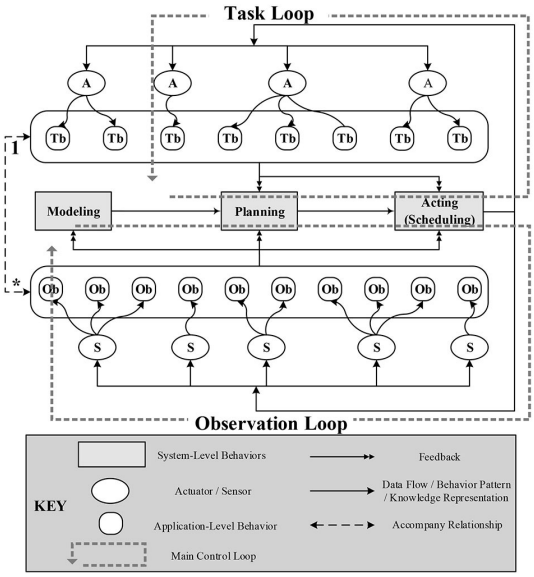

Fig. 3 The structure of D-SMPA control model

alarm message to task behaviour and task behaviour will make self-adjustment to solve the accident.

(ii) *Observation behavior*: Observation behavior cooperation pattern is designed for the interaction between observation behaviors in runtime between multiple observation behaviours and one task behaviour. The interaction between observation behaviours can improve the perception ability for ARS and assist task behaviour execute robotic task more autonomously.

(iii) *Priority pattern*: Priority pattern is designed for the situation in which specific sensing information needs to be handled first and it occurs between multiple observation behaviours with different processing priorities and one task behaviour. Under this pattern, the processing priority of observation behaviours will be predefined and task behaviour will process sensing information according to observation behaviours processing priorities.

3.3 Dual-Loop Control Model

According to the design objectives, we present a dual-loop control model called D-SMPA for autonomous robots as a control level technology. In contrast to the traditional single loop control model, this dual-loop control model can improve the reactivity of robots via multi-source feedback and provide a top-down control model in the high-level layer to support the realization of autonomy and rationality of autonomous robots via the supporting accompanying behaviour. The structure of D-SMPA is illustrated in Fig. 3. The D-SMPA control model has the following properties:

- *Separation of behaviours.* In the D-SMPA control model, we divide the ARS behaviours into two categories: system-level behaviours and application-level behaviours. The system-level behaviours include modelling, planning and acting (scheduling) behaviours. The application-level behaviours include task behaviours (Tb in Fig. 3) and observation behaviours (Ob in Fig. 3). Task behaviours are executed by the various actuators (A in Fig. 3), and observation behaviours are executed by sensors (S in Fig. 3).

- *Dual-loop for different behaviours.* According to the design objectives, we have designed two main control loops in D-SMPA, namely, the task loop and the observation loop, which are illustrated in Fig. 3 and are designed for improving the perception capability of the robots. In D-SMPA, the system-level behaviours are shared by these two main control loops; the difference between these two loops is that each loop has different actuators, behaviours and feedback loops. In each loop, the acting behaviour dispatches a related task sequence to its actuators (sensors), and these tasks are executed in a decentralized manner by different actuators (sensors). During the task execution, actuators (sensors) send feedback of the execution status (sensing data) to describe the system-level behaviours on-demand.

- *Multi-source feedbacks.* In the D-SMPA control model, three types of feedback exist: the feedback between actuators and system-level behaviours, the feedback between sensors and system-level behaviours, and the feedback between actuators and sensors. In the first type of feedback, actuators send their execution status to the planning or acting behaviours on-demand. In the second type of feedback, sensors send sensing data describing system-level behaviours. The last type of feedback is referred to as accompanying behaviours.

- *Accompanying behaviour.* As mentioned, in the D-SMPA control model, the coordination between task behaviours and observation behaviours is defined as the accompanying behaviour. In general, the execution of one task behaviour must be accompanied by one or more sensors illustrated in Fig. 3. The accompanying behaviour is implemented by exchanging the sensing data or status of behaviours directly between the task behaviours and observation behaviours.

3.4 Multi-Agent Software Architecture

The paradigm and technology of MAS that originate from artificial intelligence and recent software engineering fields provide an effective solution to address the development issues of high-level ARS for both modelling and implementation aspects. MAS represents a new method to solve the distributed problem regarding a number of autonomous agents. An autonomous robot is a complex system composed of a number of diverse and interacting components. All of these components work together to achieve the robot's design objectives. Therefore, an autonomous robot can be decomposed and organized as multiple autonomous entities mod-

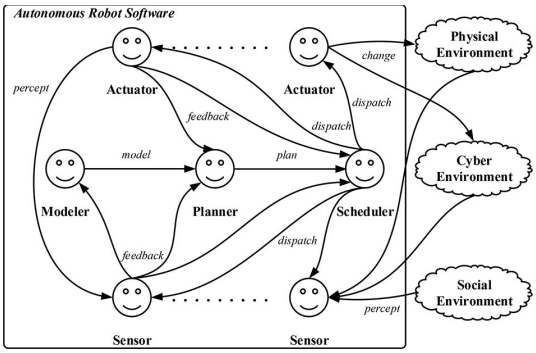

Fig. 4 Multi-agent software architecture for autonomous robot software framework

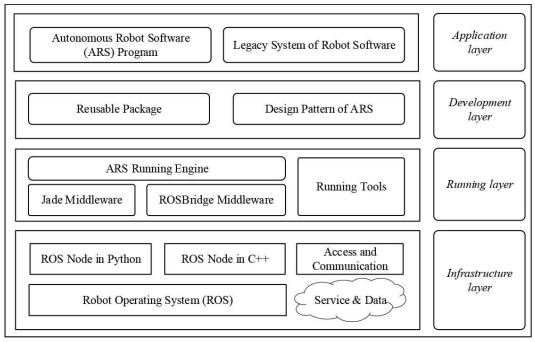

Fig. 5 The architecture of *AutoRobot*

elled as agents.

MAS also presents some important theories and technologies to support the development and implementation of complex software systems [15], ranging from BDI architecture and theory, agent-oriented methodology and the modelling language to the programming language [16] and framework. Such technologies enable developers to construct software that is capable of perceiving the environment, taking autonomous or even pro-active behaviours, and interacting with other systems.

According to the above analysis, MAS implies a novel epistemology and methodology to analyse, design and implement complex software systems. In this study, we intend to model and design high-level ARS as MAS. Each agent in ARS plays different roles, takes distinct behaviours, and cooperates with each other to support the operation of the autonomous robot. In fact, ARS represents a domain-specific application and problem-solving approach that typically consists of several specific agents, such as sensors, actuators, planners, modellers, and schedulers. These agents coordinate with each other to exchange information and access situations and services.

Based on our control model and design objectives, we design a multi-agent software architecture for our framework in Fig. 4 as a development level technology. This multi-agent software architecture can solve the integration and interaction problems in the high-level layer and provide support for accompanying behaviour and multi-source feedbacks. For the integration and interaction between the high-level layer and the low-level layer, we will provide methods to solve these problems in later section.

4. Overview of *AutoRobot*

We have developed the agent-based software development framework *AutoRobot* for ARS based on the critical software technologies that we proposed in the last section. As shown in Fig. 5, the architecture of *AutoRobot* consists of four layers: the infrastructure layer, running layer, development layer and application layer. *AutoRobot* separates the high-level decision making and control and low-level behaviour execution for ARS. Each layer of *AutoRobot* has its own responsibility in the development and operation of the ARS.

- *Infrastructure layer.* This layer provides fundamental robot functionality abstraction to encapsulate ROS node programs constructed in C++ or Python language and their communications via ROS [17] middleware. ROS is a collection of tools, libraries, and conventions for simplifying the task of creating complex and robust behaviour across a wide variety of robotic platforms. Currently, this layer has implemented several aspects of basic robotic functionalities into 25 ROS controllers (approximately 2000 lines of code), including the navigation, manipulation, and object recognition functionalities. In the framework, the interior communication of infrastructure relies on the mechanism of ROS. The nodes encapsulate robot basic functionalities in robotic platforms and directly control robot actuators.

- *Running layer.* This layer provides running supports for the ARS at run-time. The running layer implements the dual-loop control models and corresponding mechanisms to achieve the accompanying behaviour and feedback between software agents and changes in self or the environment. This layer acts as the medium between the development layer and the infrastructure layer. The running and lifecycle management of individual agents are based on the JADE framework [18]. In addition, the connection and interaction between the agents and the ROS nodes are based on the RosBridge middleware.

- *Development layer.* This layer mainly supports the developers who construct and implement ARS by providing reusable software packages and software development tools. The package implements elementary functionalities of MAS into Java-based agent templates, including 5 types of agent templates and 1000 lines of code. Developers of ARS can extend the reusable agents to construct the software systems for autonomous robots.

- *Application layer.* Using the application layer, users or developers construct multi-agent software system for ARS by extending or instancing reusable software packages provided by *AutoRobot*. Various software agents can be

designed and implemented according to the requirements of ARS. The codes are to be complied and executed under the AutoRobot framework.

5. Development and Running Supports of AutoRobot

This section introduces the software development and running supports provided by *AutoRobot* framework in four aspects: reusable software packages as development support, integration with ROS, deployment architecture and management toolkits as running supports.

5.1 Software Package

In *AutoRobot*, we provide development support by a series of reusable software packages in the development, running and infrastructure layers. Figure 6 shows the software packages of AutoRobot. The multi-agent system package encapsulates a variety of agent abstractions and it corresponds to the Reusable Package in the Development layer in Fig. 5. The interaction mechanism package provides multiple communication methods for the interaction between agents and it corresponds to the ARS Running Engine in the Running layer in Fig. 5. The data sharing package is responsible for sharing multimedia data from the sensing information to the visualized monitor tools and accommodating the structured data from the infrastructure layer. The tools package provides PC-based and Android-based visualized monitor tools. The data sharing package and the tools package correspond to the Running Tools in the Running layer in Fig. 5. RosBridge is an open-source software package is responsible for providing methods for the interaction between MAS and ROS and it corresponds to the RosBridge Middleware in the Running layer in Fig. 5. The ROS package is developed under ROS and is used for execution of low-level behaviours. The ROS package corresponds to the Infrastructure layer in Fig. 5. The main reusable packages we provided are the multi-agent system package and the interaction mechanism package.

- *Multi-agent system package.* The class diagram of the multi-agent system package is illustrated in Fig. 7. This package provides agent abstractions, including ModelAgent, PlannerAgent, ScheduleAgent, ActuatorAgent and SensorAgent. In the ARS, we can design agents to inherit the properties and methods of these agent abstractions. The role of each agent in ARS in specific application will be introduced in our case study. These agent abstractions provide the implementation elements for the D-SMPA control model and accompanying behavior.

- *Interaction mechanism package.* Under the JADE framework, a communication model for agents operates via asynchronous message exchange [18]. To communicate, an agent only needs to send a message to a destination. However, this single interaction method cannot satisfy the requirements of agents for tight and diverse interaction. Therefore, we designed an interaction mechanism package in *AutoRobot* that can provide a topics-based and services-based communication mechanism to meet the needs of synchronous and asynchronous interaction between agents. Figure 8 shows the class diagram of the in-

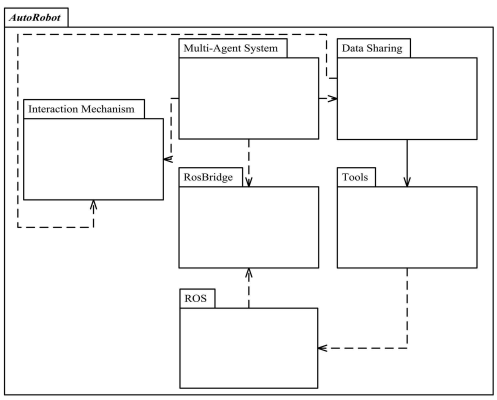

Fig. 6 Software packages of *AutoRobot*

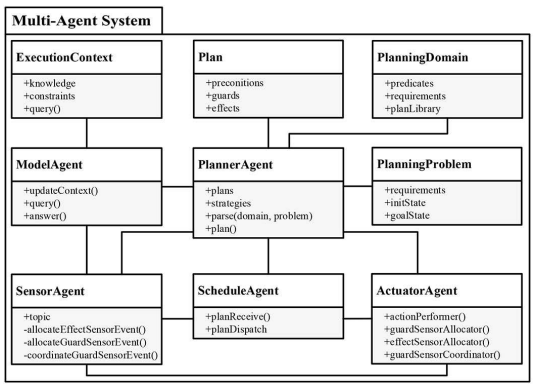

Fig. 7 Multi-agent system design model

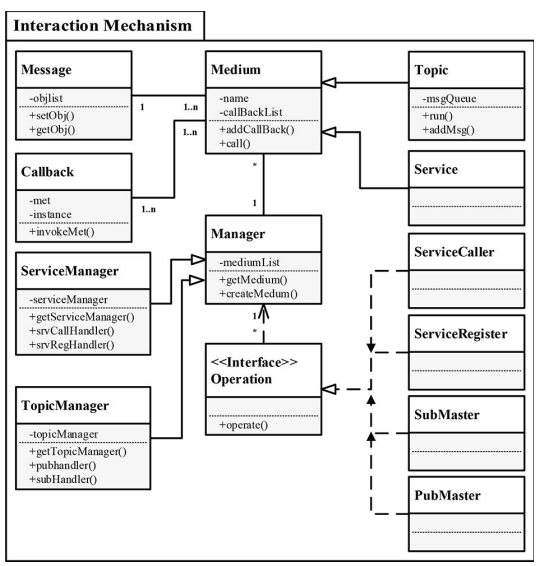

Fig. 8 Class diagram of interaction mechanism package

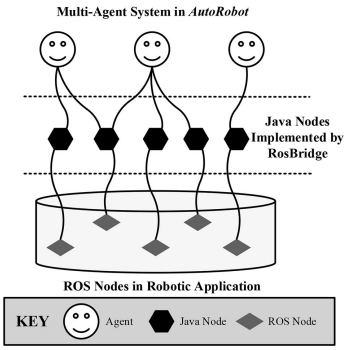

Fig. 9 The integration with ROS in *AutoRobot*

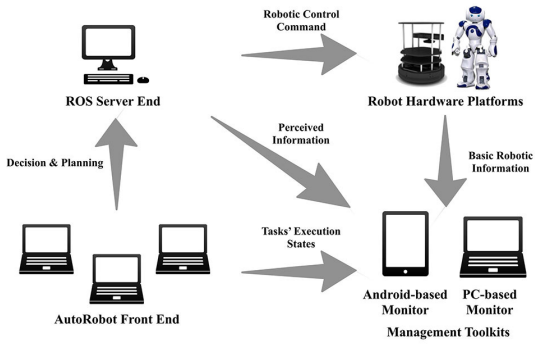

Fig. 10 Deployment architecture of *AutoRobot*

teraction mechanism package. This package implements two new communication mechanisms among agents and improves the interaction and feedback mechanisms between agents.

5.2 Integration with ROS

In *AutoRobot*, agents are implemented in Java under JADE, and the infrastructure robotic control nodes are implemented in C++ or Python under ROS. We choose the open-source software package RosBridge (shown in Fig. 6) for the communication between agents and ROS nodes to integrate ROS for *AutoRobot*. RosBridge provides a JSON API to ROS functionality for non-ROS programs. In autonomous robot applications, we create special Java nodes for each ROS node and obtain the data or messages from the ROS nodes via calling methods provided by RosBridge. Next, Java nodes deliver these messages to agents under ACL on-demand. Although this approach increases the amount of the complexity of the system, it is an effective method to solve the communication between agents and infrastructure and provide a convenient means of integration with ROS for *AutoRobot*. Figure 9 illustrates the integration with ROS in *AutoRobot*

5.3 Deployment Architecture

The four parts of the deployment architecture for *AutoRobot* are shown in Fig. 10: the *AutoRobot* front end, ROS server end, robot hardware platforms and management toolkits. These different parts are deployed in different software frameworks and can be distributed in different platforms.

- *AutoRobot front end.* The *AutoRobot* front end is deployed in JADE and provides reusable software packages for ARS using Java. Autonomous applications can construct the high-level multi-agent systems in the *AutoRobot* front end by inheriting the agent models from reusable software packages. The interaction between the *AutoRobot* front end and the ROS server end is implemented by Java nodes based on RosBridge in the *AutoRobot* front end. The interaction between the *AutoRobot* front end and the management toolkits is implemented by the topics-based and services-based communication mechanisms that we have provided.

- *ROS server end.* The ROS server end is deployed in the ROS robotic software framework and includes a series of ROS nodes for different robots. These ROS nodes are used to implement basic robotic functionalities. The interaction between the ROS server end and the robot hardware platforms is implemented by the control mechanisms provided by ROS. The interaction between the ROS server end and the management toolkits is implemented in the same manner.

- *Robot hardware platform.* At present, our robot hardware platforms are the humanoid robot NAO and the mobile robot Turtlebot2. These two robots can be controlled by ROS nodes in the ROS server end. *AutoRobot* can support multi-robot cooperation. Moreover, robots can directly report their self-state information (such as electricity information) to management toolkits.

- *Management toolkits.* Two visualized monitors based on PC and Android Mobile serve as the management toolkits in *AutoRobot*. These management toolkits are introduced in the next subsection.

5.4 Management Toolkits for ARS

In *AutoRobot*, we designed and implemented two visualized monitor tools as management toolkits based on PC and Android Mobile platforms to provide support for information visualization and management of autonomous robots. These tools display four types of information: (1) task-planning information (such as a behaviours list of robot-oriented tasks); (2) the execution state of tasks; (3) real-time sensing information, including multimedia data (such as video information and audio information) and numerical data or structured data (such as pressure values or distance values after processing and calculation from depth data); and (4) the basic robotic information (such as electricity information).

These two monitors gather multimedia information and numerical sensing data from the ROS nodes via ROS topics and services, and they obtain task-planning information, the

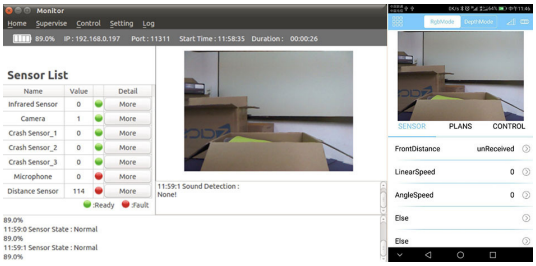

Fig. 11 User interface in the PC-based monitor and Android-based monitor

execution state of tasks and certain structured data by communicating with agents via the proposed agent interaction mechanisms. Figure 11 shows the user interface in the PC-based monitor and the Android-based monitor.

6. Case Study

In this section, we implement the example described in Sect. 2 and validate the effectiveness and applicability of our framework. The hardware platform we adopted in the case study is the Turtlebot2 mobile robot, including a mobile base and Kinect modules, each of which consists of an RGB camera and a depth camera. The software infrastructure consists of the *AutoRobot* server, the ROS server and the visualized monitor tools. The *AutoRobot* server provides the essential run-time containers for the agent entities. The ROS server provides the infrastructure for the node programs to communicate with each other. The visualized monitor tools are designed to inspect the plan execution status and visualize the sensor data.

6.1 Implementation Architecture

The case study for the aforementioned example is implemented using a multi-agent system prototype that is embedded in the *AutoRobot* framework, and the concrete architecture for the case study is illustrated in Fig. 12. We divide this implementation architecture into tow parts, one is *high-level agent programs* and the other is *low-level ROS controllers*. In high-level agent program, the description of the role that each agent plays in this multi-agent system is as follows:

- The PlannerAgent performs modelling and planning jobs, including activities of establishing world models and planning over a specific problem domain.
- The ScheduleAgent acts as a mediator that dispatches the generated plans to a specific actuator agent capable of the corresponding action.
- The ModelAgent establishes the word model on the basis of the sensor inputs gathered by the sensor agents and then offers the specifications of planning domain and problem to the planner agent for task planning.
- An ActuatorAgent is implemented as the abstraction over the robot physical actuator and maintains a simple reactive structure, effectively managing the individual robot actuator and translating the plan into primitive actions.
- A SensorAgent controls an independent sensor device of the robot and is implemented to perform a stimulus-response behaviour aimed towards external state changes.

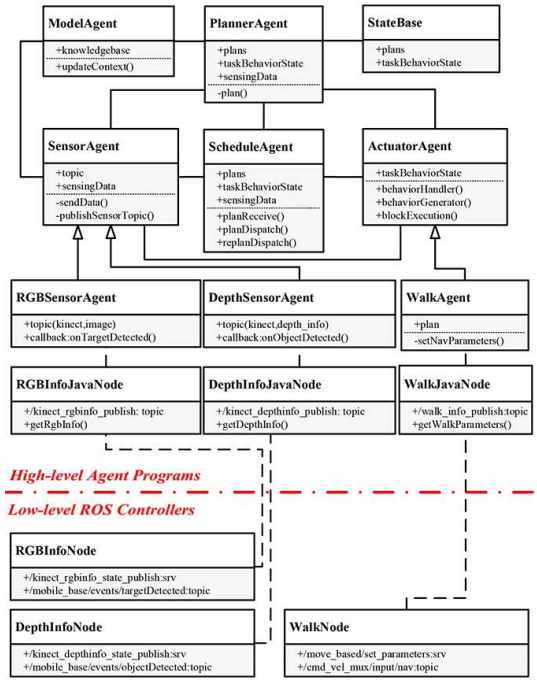

Fig. 12 The multi-agent implementation architecture of the case study

For this specific example and the Turtlebot2 capability, we designed the following: (1) a walk agent to drive the mobile base to move around, (2) an RGB sensor agent and depth sensor agent to obtain the RGB and depth images from the RGB and depth cameras, and (3) the bumper agent to obtain the pressure measurement from the bumper sensor. These agent entities extended the super class of the actuator agent and the sensor agent to implement concrete capabilities of the robot hardware.

Moreover, we implement the low-level robot controllers into ROS nodes that offer the fundamental robot functionalities. Each of the ROS node programs corresponds to a high-level agent entity used to fulfil the agent's capability. The ROS nodes communicate with the high-level agent programs through the Java nodes implemented by RosBridge middleware according to the topics or services communication mechanisms.

6.2 Design Details and Running Results in Scenarios

Under the implementation architecture, we implemented the service scenarios of searching, following and locking on the elderly inhabitants. We explain the design details of the service scenarios as follows:

(a) *Following scenario* (b) *Lock-on scenario*

Fig. 13 The running results of scenarios in real world

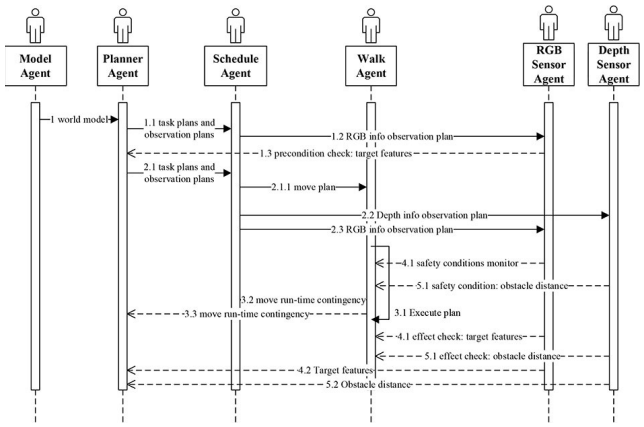

Fig. 14 The sequence diagram of agent collaboration in the *Searching scenario*

- *Searching scenario.* In this scenario, the robot must search for a target in the environment with obstacles. The agent roles involved in the scenario are the following: walk agent, RGB sensor agent, depth sensor agent, planner agent and schedule agent. Through collaboration and interaction, these agents help the robot avoid obstacles and find the target. Figure 14 illustrates the collaboration and interaction behaviours of agents via the sequence diagram.

- *Following scenario.* In this scenario, the robot finds the target and follows the target in an open and complex environment. The involved agents in this scenario are the walk agent, the RGB sensor agent and the depth sensor agent. Figure 13 shows the real-world environment in a messy laboratory and the real implementation of this scenario.

- *Lock-on scenario.* In this scenario, the robot locks on to a target when it is following the target person in the presence of obstacles in the environment, such as other, irrelevant persons. The agent roles involved in this scenario are the same as in the following scenario described above. As shown in Fig. 15, we present the sequence diagram to depict the concrete collaboration and interaction behaviours of agents in the following scenario and the lock-on scenario. Figure 13 shows the actual effect of the implementation of the lock-on scenario.

As noted above, in each scenario, the collaboration and interaction behaviours among agent roles have been implemented under the accompanying behaviour pattern in the D-SMPA control model. In addition, we conclude that the

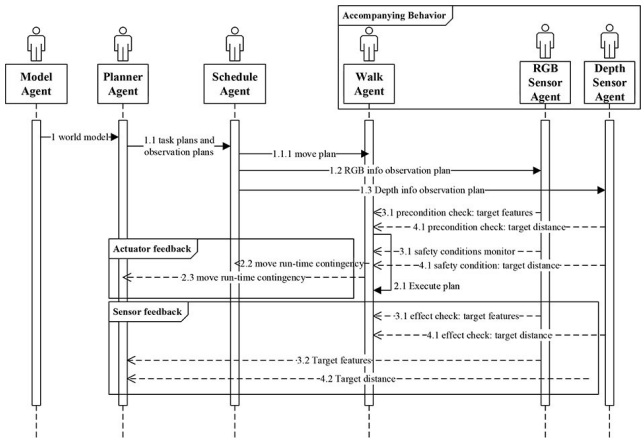

Fig. 15 The sequence diagram of agent collaboration in the *Following* and *Lock-on scenario*

implementation of these scenarios in *AutoRobot* indicates that *AutoRobot* has achieved our design objectives. Moreover, we consider the following scenario and the lock-on scenario to present the concrete collaboration and interaction behaviours of the agents and the implementation of the objectives.

- *Support for realization of autonomy and rationality.* As shown in Fig. 14, we present snapshots of the implemented service scenario to depict the following process of the robot in a dynamic environment. A complete video demonstration is available at the project website[†]. In these scenarios, from the results, the robot can follow and lock on the target person in an open environment smoothly without external control. Examining the process, the accompanying behaviour appearing through the complete implementation of the scenario supports the realization of autonomy and rationality. The concrete accompanying behaviour is achieved by the collaboration between the walk agent and the agents of the RGB and depth sensors (Fig. 15). More specifically, the RGB and depth sensor agents run in synergy with the walk agent to provide the necessary observation results for plan execution. Before the walk agent (task behaviour) starts to execute the move plan, the sensor agents (observation behaviours) expect to observe whether the target is detected to guarantee the applicability of the plan and this is the *pre-execution phase* of accompanying behaviour. While the robot is following the target, the depth sensor agent (observation behaviour) is observing whether there are unexpected obstacles through the depth measurement to ensure the safety of the plan execution, and this is the *in-execution phase* of accompanying behaviour. To ensure the actual result of the move plan, the RGB sensor agent (observation behaviour) can check if the target person is still in the robot vision, and the depth sensor agent (observation behaviour) checks whether the distance between the target person and the robot is within the expected limit, this is the *post-*

[†]https://www.trustie.net/projects/3261/files

execution phase of accompanying behaviour.

- *Enhancing the reactive and flexible adjustment of behaviours.* In our case, we implement the D-SMPA control model, which provides a multi-source and multi-loop feedback mechanism for ARS to enhance the reactive and flexible adjustment. As shown in Fig. 15, two types of feedback are involved in the scenario, namely, the actuator feedback and the sensor feedback. For the actuator feedback, as the robot is moving towards the target person, possible run-time contingencies (such as moveable obstacles) may hinder the normal moving process and result in execution failure; in this case, the walk agent provides the feedback of the run-time execution status to the schedule agent for task re-scheduling and the planner agent for task re-planning. For the sensor feedback, the RGB and depth sensors remain activated to observe the desired surrounding information, such as the features of the target person and the distance with the robot. The feature and distance information are provided as feedback to the walk agent for run-time awareness and the planner agent to update the planning domains.

- *Integration and interaction.* In this case, various software components exist, such as agents, Java nodes and ROS nodes; these are developed and running in different platforms and different programming languages. However, through the architecture of *AutoRobot* and the communication mechanisms in *AutoRobot*, these components are well integrated as a hierarchical pattern and interact and cooperate with each other in the expected manner.

7. Related Work and Discussion

The software architecture of a robot plays an important role in developing robot applications [19]. The implementation of software architecture is represented via software frameworks.

Current frameworks of robotic software development are based primarily on two types of approaches: one type includes the component-based approaches, and the other type includes the agent-based approaches. The *AutoRobot* framework represents our approach to solve development issues in the implementation of ARS and applications under an agent-based architecture. Many other well-known robotic software frameworks exist. In this section, we introduce selected frameworks and compare them with *AutoRobot* to discuss the strengths and weaknesses of our approach.

- *Component-based framework.* The component-based framework is the most popular approach for robotic software development because this framework is essential to any engineering discipline when complexity dictates methodologies that leverage reuse and correct-by-construction approaches [1]. Modularization, encompassing heterogeneity, achieving constructivity and simplifying the development process compose the strengths of the component-based framework. However, the largest drawback of the component-based framework is the lack of support for autonomy. YARP (Yet Another Robot Platform) [20], ArmarX [21], Player/stage [22], Webots [23], ORCA [24], MARIA (Mobile and Autonomous Robotics Integration Environment) [25], ROS [17], and V-REP (Virtual Robot Experimentation Platform) [26] are common component-based frameworks for robotic software.

- *Agent-based framework.* Relative to the component-based framework, the agent-based framework can provide more support for autonomy of ARS. Based on the sociological feature of MAS, in an agent-based framework, every component or robot can appear as an agent, and each agent can interact or cooperate with each other relatively flexibly and autonomously. *AutoRobot* is a type of agent-based framework. VOMAS (Virtual Operator Multi-Agent System) [27] and COROS [13] are common agent-based frameworks for robotic software.

To discuss the strengths and weaknesses of our framework *AutoRobot*, we establish a conceptual framework to compare *AutoRobot* and the other software frameworks mentioned above [28]. In the context of a robotic software framework, here, a platform for the implementation of ARS via an integral and concrete production of software engineering methodologies, tools and libraries, we present three categories of criteria for comparison that cover support for ARS, software engineering characteristics and platform supports.

- Support for ARS:

  - *Support for autonomy.* This is one of our design objects addressed in Sect. 2 and is important for an ARS framework. Autonomy and rationality are the fundamental elements to distinguish an autonomous robot from a traditional industrial robot.
  - *Promotion of reactivity.* This is also one of our design objects addressed in Sect. 2. Robot software reactivity depends on whether the software architecture implements basic sensor-effector reacting loops and executes reactive strategies in response to emergent situations.
  - *Robustness.* As the environment becomes increasingly open and the system becomes increasingly complex, robustness is becoming increasingly important for autonomous robots. Satisfactory robustness can greatly improve the capability of fault-tolerance and adaptability to the environment for autonomous robots.
  - *Multi-robot cooperation.* Multi-robot applications are a major trend of development of autonomous robots. The development of software for a multi-robot software application generally requires that the robotic software framework offers convenient facilities to solve coordination issues of robot teams, such

Table 2 Comparison of robotic software frameworks. Letter codes are as follows: P=Python, J=Java.

Criteria / Frameworks	Support for autonomy	Promotion of reactive	Robustness	Multi-robot cooperation	Reusability	Modularity	Maintenance	Programming language	Management toolkits	Integration with ROS	Reusable packages
YARP	√-	√+	√-	√-	√+	√+	√+	C++	×	×	√+
ArmarX	√+	√-	√+	×	√+	√+	√-	C++, P, J	√+	×	√+
Player\stage	×	×	√-	√+	√-	√-	√-	C++, P, J	√-	×	√-
Webots	×	×	√-	√+	√-	√-	√-	C++, J	√-	×	√-
ORCA	×	×	√-	×	√+	√-	√+	C++	×	×	√+
MARIA	√-	√-	√-	√-	√+	√+	√+	C++	×	×	√+
ROS	√-	√-	√-	×	√+	√-	√-	C++, P	×	√+	√+
V-REP	×	√-	√-	×	×	√-	×	Lua	√+	√-	√+
VOMAS	√+	√+	√+	√+	√-	√+	√+	J	×	×	√-
COROS	√-	√-	√+	√+	√+	√-	√-	C++, P	×	√+	√+
AutoRobot	√+	√+	√+	√-	√+	√+	√+	C++, P, J	√+	√+	√-

as distributed communication protocols, sensor networks, and knowledge bases.

- Software engineering characteristics:

  - *Reusability.* Reusability depends on whether the software architecture or middleware provides carefully designed program interfaces and suitable portability of successful robotic software products among heterogeneous robot platforms.
  - *Modularity.* A modular robotic software architecture consists of several components of high cohesion and low coupling, with the dependences among components being kept at a minimum to obtain a maintainable and scalable software.
  - *Maintenance.* A robotic software being satisfactorily maintained allows flexible and straightforward modification of a certain robotic functionality, such as task-planning algorithms and off-the-shelf knowledge reasoning engines.

- Platform supports:

  - *Programming language.* The programming language used for development is also an important aspect. The greater the number of supported languages, the more flexibility is contributed to the development.
  - *Management toolkits.* Management toolkits can help the developer and application users to identify the execution states of tasks and robotic sensing information more intuitively. These toolkits can facilitate the development and management of autonomous robot application.
  - *Integration with ROS.* ROS, one of the most commonly used robotic frameworks, provides support for the development of the vast majority of autonomous robots. Integration with ROS can improve the extensibility of frameworks.
  - *Reusable packages.* The framework must establish well-structured abstractions over basic robotic functionalities and implement them as reusable packages to ease development efforts for programmers.

The results of the comparison are shown in Table 2. For each framework, a value has been assigned for the criteria based on the publications and user experiences. In addition, two types of assignments are made in the comparison: (1) a designation of "×" for not supported, "√-" for partially supported, "√+" for well supported, and (2) brief text descriptions.

Table 2 indicates that *AutoRobot* outperforms most of other robotic software frameworks in following aspects: (1) *AutoRobot* supports the autonomy, perfect reactivity and favourable robustness of the ARS by providing an explicit behaviour pattern, control model, architectural structures and interaction mechanisms; (2) *AutoRobot* satisfies good software engineering practices; and (3) *AutoRobot* supports users to manage the states of robots and task execution, and *AutoRobot* favourably integrates ROS for robotic control.

However, certain limitations exist with *AutoRobot*.

- First, *AutoRobot* cannot provide sufficient development and running support for multi-robot applications. Although we have designed the communication and interaction mechanisms between robots in *AutoRobot*, we are still developing a single robot at the present stage.

- Second, as shown in Table 2, *AutoRobot* does not provide a sufficient extent of reusable packages for development of ARS because *AutoRobot* is still in the early stage of development; we are currently developing more software packages in *AutoRobot*.

- Third, although *AutoRobot* provides satisfactory support for the autonomy and reactivity of ARS, we suggest that certain aspects remain to be improved. For example, we can improve the autonomy by providing an autonomous decision mechanism or a task allocation mechanism.

- Finally, programmers must use C++ or Python to design ROS nodes and use Java to design agents; this approach is not user-friendly for new programmers.

8. Conclusion

Autonomous robots are software-driven, cyber-physical and social eco-systems that operate in an open environment; software plays an important role in such robots. The requirements of ARS challenge the existing robot software

architecture, framework and resultant implementation approaches from software engineering perspectives. We have provided the ARS framework *AutoRobot* for ARS development and implementation. The contributions of this study are as follows:

1. *New perspectives on ARS.* Through an analysis of the features and requirements of ARS, we conclude that the software development of ARS must satisfy three design objectives: (1) provide support for realizing autonomy and rationality, (2) improve reactivity, and (3) enable improved integration and interaction.

2. *Enabling software technologies.* Based on the design objectives, we have presented three enabling technologies: (1) accompanying behaviours as a behaviour pattern to better realize autonomy and rationality, (2) a dual-loop control model to improve the capability of feedback, and (3) an agent-based software model to implement better integration and interaction.

3. *AutoRobot multi-agent software development framework.* *AutoRobot* achieves our design objectives via the enabling technologies that we have presented. *AutoRobot* provides a series of reusable software packages as software development supports; in addition, *AutoRobot* provides a deployment architecture and visualized monitor tools based on PC and Android Mobile as deployment and management supports.

We have successfully developed several ARS scenarios to validate the effectiveness and applicability of our proposed framework. Our future research studies include the following: (1) providing various reusable packages for the development of additional ARS and autonomous robot applications; (2) designing additional mechanisms to support autonomous decision and task allocation; and (3) developing additional autonomous robot applications to validate and improve the control model and software framework.

References

[1] S. Bensalem, F. Ingrand, and J. Sifakis, "Autonomous robot software design challenge," Proceedings of Sixth IARP-IEEE/RAS-EURON Joint Workshop on Technical Challenge for Dependable Robots in Human Environments, May 2008.

[2] D. Kortenkamp and R. Simmons, "Robotic systems architectures and programming," Springer Handbook of Robotics, pp.187–206, Springer Berlin Heidelberg, 2008.

[3] P. Ziafati, "Programming autonomous robots using agent programming languages," Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems, pp.1463–1464, International Foundation for Autonomous Agents and Multiagent Systems, May 2013.

[4] L.A. Pineda, L. Salinas, I.V. Meza, C. Rascon, and G. Fuentes, "Sitlog: a programming language for service robot tasks," International Journal of Advanced Robotic Systems, vol.10, no.10, p.358, 2013.

[5] A. Chella, M. Cossentino, S. Gaglio, L. Sabatucci, and V. Seidita, "Agent-oriented software patterns for rapid and affordable robot programming," Journal of Systems and Software, vol.83, no.4, pp.557–573, 2010.

[6] R. Brooks, "A robust layered control system for a mobile robot," IEEE Journal on Robotics and Automation, vol.2, no.1, pp.14–23, 1986.

[7] P. Chen and Q. Cao, "A middleware-based simulation and control framework for mobile service robots," Journal of Intelligent and Robotic Systems, vol.76, no.3-4, pp.489–504, 2014.

[8] P. Iñigo-Blasco, F. Diaz-del-Rio, M.C. Romero-Ternero, D. Cagigas-Muñiz, and S. Vicente-Diaz, "Robotics software frameworks for multi-agent robotic systems development," Robotics and Autonomous Systems, vol.60, no.6, pp.803–821, 2012.

[9] T.M. Burke, C.-J. Chung, D.P. Casasent, E.L. Hall, and J. Roning, "Autonomous robot software development using simple software components," Proceedings of Intelligent Robots and Computer Vision XXII: Algorithms, Techniques, and Active Vision, pp.107–117, 2004.

[10] T. Abdellatif, S. Bensalem, J. Combaz, L. De Silva, and F. Ingrand, "Rigorous design of robot software: A formal component-based approach," Robotics and Autonomous Systems, vol.60, no.12, pp.1563–1578, 2012.

[11] P. Skrzypczyński, "Multi-agent software architecture for autonomous robots: a practical approach," Management and Production Engineering Review, vol.1, no.4, pp.55–66, 2010.

[12] S. Benaissa, F. Moutaouakkil, and H. Medromi, "New Multi-Agent's Control Architecture for the Autonomous Mobile Robots," International Review on Computers and Software, vol.6, no.4, pp.477–480, 2011.

[13] A. Koubâa, M.-F. Sriti, H. Bennaceur, A. Ammar, Y. Javed, M. Alajlan, N. Al-Elaiwi, M. Tounsi, and E. Shakshuki, "COROS: A Multi-Agent Software Architecture for Cooperative and Autonomous Service Robots," Studies in Computational Intelligence, 2015, vol.604, pp.3–30, 2015.

[14] J. Lacouture, V. Noël, J.-P. Arcangeli, and M.-P. Gleizes, "Engineering agent frameworks: An application in multi-robot systems," Advances on Practical Applications of Agents and Multiagent Systems, vol.88, pp.79–85, Springer, Berlin, Heidelberg, 2011.

[15] N.R. Jennings, "An agent-based approach for building complex software systems," Communications of the ACM, vol.44, no.4, pp.35–41, 2001.

[16] M.B. van Riemsdijk, "20 years of agent-oriented programming in distributed AI: history and outlook," Proceedings of the 2nd edition on Programming systems, languages and applications based on actors, agents, and decentralized control abstractions, pp.7–10, ACM, Oct. 2012.

[17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and Ng, Y. Andrew, "ROS: an open-source Robot Operating System," ICRA Workshop on Open Source Software, vol.3, no.3.2, p.5, 2009.

[18] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa, "JADE: A software framework for developing multi-agent applications. Lessons learned," Information and Software Technology, vol.50, no.1, pp.10–21, 2008.

[19] L. Natale, A. Paikan, M. Randazzo, and D.E. Domenichelli, "The icub software architecture: evolution and lessons learned," Frontiers in Robotics and AI, vol.3, 24, 2016.

[20] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: yet another robot platform," International Journal of Advanced Robotic Systems, vol.3, no.1, p.8, 2006.

[21] N. Vahrenkamp, M. Wächter, M. Kröhnert, K. Welke, and T. Asfour, "The robot software framework armarx," it-Information Technology, vol.57, no.2, pp.99–111, 2015.

[22] B. Gerkey, R.T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," Proceedings of the 11th International Conference on Advanced Robotics, vol.1, pp.317–323, June 2003.

[23] O. Michel, "WebotsTM: Professional Mobile Robot Simulation," International Journal of Advanced Robotic Systems, vol.1, no.1, 2004.

[24] A. Makarenko, A. Brooks, and T. Kaupp, "Orca: Components for

robotics," International Conference on Intelligent Robots and Systems (IROS), pp.163–168, Oct. 2006.
[25] C. Côté, Y. Brosseau, D. Létourneau, C. Raïevsky, and F. Michaud, "Robotic software integration using MARIE," International Journal of Advanced Robotic Systems, vol.3, no.1, p.10, 2006.
[26] E. Rohmer, S.P.N. Singh, and M. Freese, "V-REP: A versatile and scalable robot simulation framework," Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, pp.1321–1326, IEEE, Nov. 2013.
[27] H.C.-H. Hsu and A. Liu, "A flexible architecture for navigation control of a mobile robot," IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans, vol.37, no.3, pp.310–318, 2007.
[28] S. Yang, X. Mao, S. Yang, and Z. Liu, "Towards a hybrid software architecture and multi-agent approach for autonomous robot software," International Journal of Advanced Robotic Systems, vol.14, no.4, 1729881417716088, 2017.

Zhe Liu received the B.S. degree from Shanghai Jiaotong University, China in 2016. From 2016, he has been a Master course student at National University of Defense Technology. His research interests include robot software framework and robot control system.

Xinjun Mao received his Ph.D. degree of computer science from National University of Defense Technology in China in 1998. He is currently a Professor and Director of computer software and theory Lab at the National University of Defense Technology. His main interests are in the area of software engineering for complex and intelligent systems operating in open environment, like multi-agent systems, self-adaptive and self-organized systems, autonomous robot systems.

Shuo Yang received the B.S. degree from National University of Defense Technology, China in 2015. From 2015, he has been a Master course student at National University of Defense Technology. His research interests include software engineering for robotics and intelligent software technology.