The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

A Component Based Design Framework for Robot Software Architecture

Wei Hongxing[1], Duan Xinming[1], Li Shiyi[1], Tong Guofeng[2], Wang Tianmiao[1]
1. Robotics Institute, Beijing University of Aeronautics and Astronautics
Beijing, 100191, China
2. Institute of A.I. and Robotics and Key Laboratory of Integrated Automation of Process Industry,
Northeastern University, Shenyang, 110004, China

***Abstract*—Componentization is an important method to improve the reusability of robot software and reduce the difficulty of system design. In this paper, we propose a component based design framework for robot software architecture. First, the robot system is functionally decomposed into reusable components. On this basis, the static model and run-time model of component are established, and a component interface definition language based on the model is designed. Second, a lightweight middleware is proposed according to the communication mode between robot components, and a component development tool and a visual component assembly environment based on the middleware are designed to facilitate the developers. Finally, an application based on the framework is introduced to verify the validation of the design framework.**

Keywords: **Robot Software Architecture, Robot Middleware, Component, IDL**

I. INTRODUCTION

WITH the development of robot technology, the complexity of autonomous robot systems has increased dramatically. Besides an increased variety of robots, sensors, actuators, embedded computers; the architecture of the robot software has gained crucial relevance. Monolithic Programming is used in the traditional robot software design. But a variety of robot systems often use different hardware platforms, so the robot system software changes with different hardware platforms and it is difficult to realize the reusability of robot system software. Modern robot software uses multiple functional components. The main aim of the architecture for Robot software is to provide flexible and reliable communication mechanisms for data exchange between the components. For todays robot applications, mostly middleware systems are used to accomplish this task.[1,2]

To solve the heterogeneity between the controller and modules in robot system, the hierarchical architectures has been proposed[3], such as NASREM model proposed by Albus[4-6] and hierarchical intelligent robot software architecture proposed by Saridis[7]. With the development of Component-Based Software Engineering, software component technology is quickly applied to the design of robot software to solve the problem of software reusability, such as OROCOS[6], Miro[7], RT-Middleware[8], MSRS[9], etc. These component-based robot software architectures are often built based on the existing network middleware (such as CORBA, DCOM, etc.). However, these network middleware are complexity enterprise-class application systems, which makes the robot software architecture huge and occupies more system resources. They are not suitable for embedded robot controller system. For these reasons, some companies and research institutes have to build their own robot middleware architectures, such as Open-R[3], K-Middleware[9], ASEBA[8], MiRPA[9], etc.

Based on the robot software system decomposition and component communication mechanism, we present a component-based design framework of robot software architecture. The software system uses lightweight component communication mechanism, which is suitable for embedded robot controller software applications.

The remainder of the paper is organized as follows. Section II presents a component-based frame structure for robot software architecture and analyzes the static model and run-time model of the components. Section III presents a lightweight middleware communication mechanism between components, and sets up a visual component integrate environment. Section IV describes a component-based application example: a visual procedure to track the ball for mobile robot. The last section concludes the paper, and points at the future work.

II. FRAME STRUCTURE

A. System Decomposition

Generally, a robot system can be divided into the control system and the controlled system. Control system is the robot control software or algorithms, and the controlled system is the robot hardware. To achieve modular design of robot system and the reusability of robot software, we present a component-based abstract model to separate the coupling between control systems and the controlled system. In Fig.1 (a), robot middleware divide a robot system into two parts: software and hardware systems. The lower layer of robot middleware is the robot hardware system composed of a variety of sensors, actuators and other hardware devices. The upper layer of robot middleware is the robot control software composed of a variety of robot software components, control algorithms and services. Robot middleware hides the heterogeneity of lower hardware devices and provides the component interface independent to the robot hardware system, for the upper component or application call.

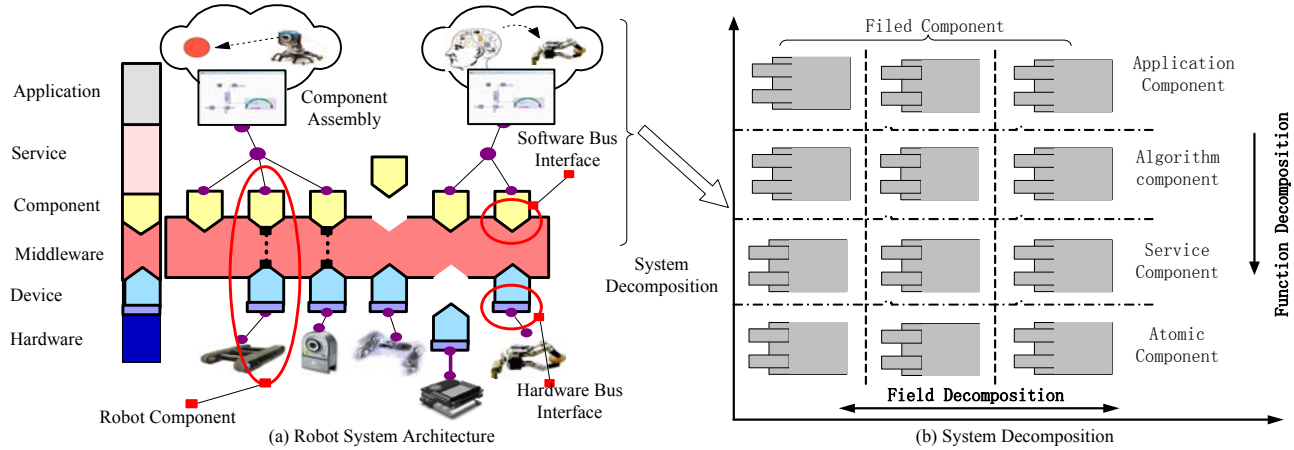

Fig.1 System framework

To improve software reusability, the robot software system is divided a set of different field and different granularity reusable software components.

In Fig.1 (b), in horizontal direction, the robot system is divided into different functional modules. The designer of various functional modules can focus on the development of their own functional modules, and package the functional module into the corresponding components according to the standard. Robot system integrators can rapidly integrate various functional modules and develop robot systems according to standard interface, without the need to understand the specific implementation details of functional modules.

In vertical direction, there are four levels of granularity of the components according to different degree of abstraction and reuse:

- Atomic component: atomic component is the only direct interaction with hardware components. It is a particular type of hardware abstraction and provides the unified interface for the upper component calls. Such as motor control atomic component, various sensors control atomic components, etc.
- Composite component: according to some function requirement and composite rule, several atomic components can be combined into composite components, which provides a higher level and bigger granularity components abstract. For example, robot chassis consists of several motors and sensors. Then, with the corresponding atomic motor components, we can compose the composite chassis component, which provides the specific service interfaces, such as chassis velocity settings: setVelocity (type left, type right). Through this abstraction, whether the chassis are four-wheel drive or two-wheel drive, the superior components can control the robot chassis through the unified service interface.
- Algorithm component: Algorithm component is a platform-independent robot algorithms or generic robot control algorithms, for example, path planning and Kalman filter, etc.
- Application component: application components are the combination of all kinds of atomic components, composite components and algorithm components to complete the application functions of the robot system.

In vertical direction decomposition of robot system, each component relies solely on its subordinate components. Subordinate components provide services to superior components, and the superior component achieves specific functions with the service of subordinate components. Through the dependence relationship between service providers and consumers, various components can be assembled together to form a specific function robot system.

B. Component static mode

After the above decomposition process, component is a high cohesion and low coupling software module. Component and its subordinate component have a kind of relations between dependence and are dependent on. They are the service consumer and provider. In this relationship, there are three kinds of communication modes between the superior and subordinate component:

- Command mode: the superior component issues an order to the subordinate components. Superior components issue an order and continue their operation without waiting for results of the implementation of subordinate components; subordinate components do not direct feedback to the superior components in implementation process. When order execution time is longer or superior components do not care about the implementation results of the case, the command mode is suitable. Command mode is mainly used in the condition that superior components allocate task to subordinate components and subordinate components decide the way to complete task.
- Request/response mode: the superior component sends request to subordinate components, in parallel, and waits for their return. The type mainly is used in component pre-operational configuration and the state access. Under normal operating conditions, it should be as little as possible to use the mode, which also applies to real-time request with

the higher real-time demand.

- Subscribe/release mode (event mode): For some information concerning by superior components, such as sensors to environmental changes, task completed, the system failure and so on, the subordinate component returns them in the form of events.

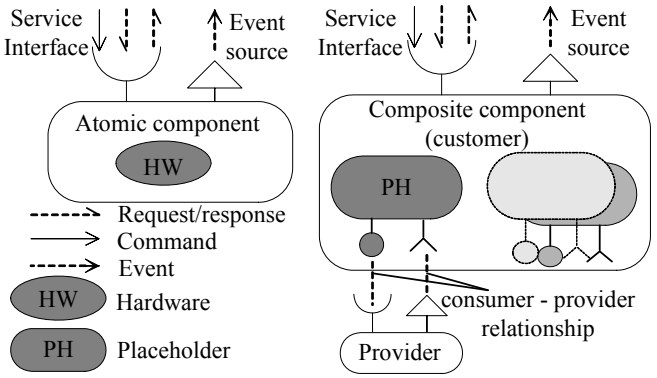

Fig.2 Component static model

Considering these three kinds of models, the event-based programming method is widely used in the robot programming. A component communication model is designed. Fig.2 (a) describes the model of atomic components. Atomic components, as the abstract of robot system hardware device, provide direct interactive services of hardware devices to the superior components, and its component interface includes a service interface and an event source. A number of components can be combined into composite components, which provide a new component interface. It is shown in Fig. 2 (b). In each composite component, there is the placeholder of component which is dependent on; the placeholder is as agent credentials of corresponding components. When composite component is running, placeholder can hides inter-component communication details.

C. Run-time component model

In majority of the robot system, accessing sensor information concerned by the superior component should be in subscription/release mode, not in request/response model; task allocation to subordinate component should be in command mode. And with the command execution error, it should be back in other ways. There are mainly three reasons:

1) Request/response mode is inefficient and slow reaction, and superior components can not immediately make a deal on environment change. This long-distance sampling method often spends more waiting time than the subscription/release model, but also more easily lead to system deadlock.

2) In mobile robot programming, behavior-based programming model is widely used. In the model, for feedback information on the environment, the system should respond in the fastest way, and multi-level arbitration is often used in the realization[10]. It will be convenient for the nearest processing and multi-level arbitration, if the system is in command mode and event mode.

3) It is compatible with distributed control and centralized control. In distributed control, assigning the sampling and initial process of sensor information to different processors, and informing the environmental information in form of event to the superior components, can reduce the burden on the host processor, and also reduce the network communication data load.

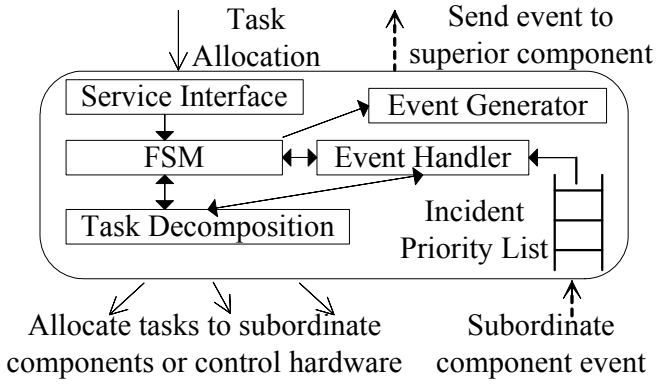

Fig.3 Run-time component model

Based on the above analysis, we construct a general run-time model, as shown in Fig.3. Superior component assigns task to subordinate component in command mode, while subordinate component feedback information in subscription/release mode. In this model, a component mainly consists of three parts: Finite State Machine (FSM), Event Handling and Task Decomposition. FSM is the core of components, maintaining the operational status of the entire component. Component itself can complete one or more tasks, corresponding to different task state. FSM schedules subordinate components to complete the task, depending on different states, and responses the incident feedback from subordinate component in time.

D. Component Interface Description Language

To standardize components and hide internal implementation details, according to robot component model, we define a simple component interface description language to describe component interface. According to the static model, the component interfaces include the service interface and the event (source) interface. Service interface is defined by the keyword interface and event interface by the keyword event. Service interface includes component command interface and request/response interface, and the command interface does not require returning value, therefore the return type is void; and request/response interface contains return value. To process simply and conveniently, we define all returned values in the type of return. And if an operation requires returning multiple values, it will use structure types (keyword: struct).

The BNF (Backus-Naur Form) of component interface description language is as follows:

```
idl_list::= idl
        | idl_list idl
idl::= intf_def
idl::= struct_def
```

```
idl::= event_def
event_def::= EVENT IDENTIFIER '{' member_list '}' ';'
struct_def::= STRUCT IDENTIFIER '{' member_list '}' ';'
member_list::= data_type IDENTIFIER ';'
        | member_list data_type IDENTIFIER ';'
intf_def::= INTERFACE IDENTIFIER '{' oper_dec_list '}' ';'
oper_dec_list::= oper_dec_list data_type IDENTIFIER '('
param_list ')' ';'
        | oper_dec_list VOID IDENTIFIER '(' param_list ')' ';'
        | oper_dec_list data_type IDENTIFIER '(' ')' ';'
        | data_type IDENTIFIER '(' param_list ')' ';'
        | VOID IDENTIFIER '(' param_list ')' ';'
        | data_type IDENTIFIER '(' ')' ';'
param_list::= param_list ',' data_type IDENTIFIER
        | data_type IDENTIFIER
data_type::= DATA_TYPE
        | STRING
        | STRUCT IDENTIFIER
        | ARRAY '<' DATA_TYPE '>'
DATA_TYPE::= int | short | char | float | double
ARRAY::= "Array"
STRING::= "String"
STRUCT::= "struct"
VOID::= "void"
INTERFACE::= "interface"
EVENT::= "event"
IDENTIFIER::= <Symbol>
```

III. MIDDLEWARE DESIGN AND COMPONENT ASSEMBLY

In the framework, inter-component communication is completed by the middleware. Component assembly is using middleware to connect various components to form a new system, according to the functional requirements. The existing network middleware is too complicated and its architecture is also too large. According the above components communication modes, we design a lightweight middleware to achieve components communication. At present, the middleware bases on the TCP/IP and the agreement can apply to any connection-oriented, reliable transmission network protocols.

A. Component communication protocol

Inter-component communication protocol uses the message as a unit, and message format is shown in Tables 1 and 2. Corresponding to the communication modes between the components, we define five types of information (Table 2). Command mode and event mode do not have the direct return mode. To solve the notification mechanism questions above-mentioned, in case of the implementation failure of command mode and event mode, we adopt two methods: for issues in the components communication process, if command or event recipient is not found and the command is not defined, we will send message fifth type CONTROL message to notify the sender; if the command fails in implement the process, the implementation components will send events to superior components to notify.

As shown in Table 1, domain Body corresponds to message body, different with the message type varies. For example, REQUEST and COMMAND type message body composed of the operator logo and the operating parameters. In order to process simple and convenient, for different length data types in message body, message recipient and sender comply with the same byte alignment. Taking the needs of cross-platform into account, different length data type should be multi-byte alignment, but resulting increase in the total length of message; and real-time bus bandwidth, such as CAN bus, is often not enough. The smaller the message length, message extension will be smaller, and the network data throughput will be bigger. Therefore we chose the data alignment which is compatible with the majority of 32-bit systems: 8-bit data one byte-aligned, 16-bit data 2-byte alignment, 32-bit and 64-bit data 4-byte alignment.

Table 1 Component Communication Protocol Format

Domain	Length	Description
Magic	4 bytes	Protocol verification number; to prevent unauthorized access; always 'RIDE'
Priority	4 bytes	The message priority. For real-time system, it can be mapped to the operating system priority and the underlying communication protocol priority
Type	2 bytes	The message type (table 2)
Length	2 bytes	The message length
Identifier	4 bytes	Message recipient identifier
Body	n bytes	Message body, defined by message type

Table 2 Message Type

Type	Sender	Description
COMMAND	Superior component	Corresponds to the command mode
REQUEST	Superior component	Corresponds to the request/response mode
REPLY	Subordinate component	Corresponds to the request/response mode
EVENT	Subordinate component	Corresponds to the incident mode
CONTROL	Superior or subordinate component	Control message, corresponds to all modes. To report the communication process errors, such as receiving messages component is not found, component and message do not meet the definition of the operation, or message format errors.

Through the agreement, three communication modes between components can be completed with the message interaction. In this paper, we use the object-oriented design methods and C++ programming language; implement the Lightweight middleware based on the TCP/IP. This lightweight middleware provide interactive encapsulation, component management, and connection management functions of the agreement above-mentioned.

B. Component Development Tools

To realize the complex interaction between the agreements above-mentioned solely by developers according to the agreement interaction details, the developers have to very deeply understand the agreements, and it is very prone to error. To hide the complexity of agreement interaction, we design component tools ***rcidl*** to support component production. To create a component, we simply use interface description language to describe the required function interface of components. We use the component tools to complete the basic framework of components, without the need to focus on the interaction details of communication protocol.

The ***rcidl*** tool is a component framework code generator with the importation of component interface description

document. Using the rcidl tool, the production of components can complete the framework design. We hide the complex interaction details of the communication protocols to developer, which makes the developer can focus on the functional implementation components, as shown in Fig.4.

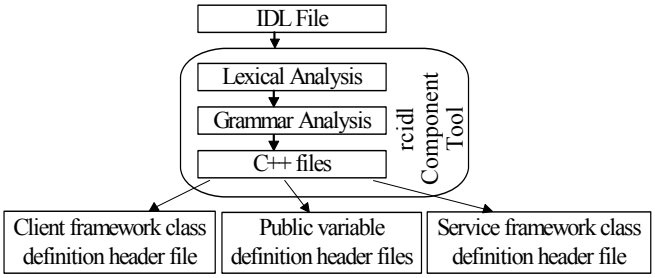

Fig.4 ***rcidl*** flowchart

According to the definition document of component interfaces, component tool ***rcidl*** separately generates the components client framework code, service client framework code and public variable definition file, as shown in Figure 4. Firstly, through lexical analysis, ***rcidl*** divides the IDL file into several marks (token). Then, through grammar analysis, it determines the grammar relationship between marks and generates the grammar analysis tree. Finally it parses of grammar analysis tree to generate the corresponding C++ framework code. The client framework class definition corresponds to placeholder framework code, and the service framework client class definition corresponds to the framework code of components implementation. These framework codes implement all the interactive content of the agreement; the developer can call the corresponding interface to implement communication between the components.

C. *Visual Assembly*

In the framework， the components assembly using a combination or aggregation mode in object-oriented design patterns. Using object-oriented programming to achieve component assembly requires more in-depth understanding on the entire component model and implementation approach. The robot system integrators, especially the users and robot enthusiasts, need a simple and convenient assembly way to achieve the rapidly design of robot application software.

In this paper, we use QT signal/slot mechanism to implement the component visual assembly environment. The QT signal/slot mechanism is an object communication mechanism designed by Trolltech Company, and it is an incident programming abstraction. The signal transmission corresponds to the event trigger, and the slot corresponds to the event process[11]. The link between signal and slot makes signal transmission be able to call the corresponding slot[11].

We pack the component placeholder as the QT custom control, translate the event received by components into the QT object signal transmission, and map the component command interface to the QT object slot. At the same time, we also design several man-machine interfaces and logic the QT-related controls to support the components assembly, such as light control, timer control, constant control, comparator controls, etc. In this way, the process of component assembly is the process of forming application through the combination of controls.

We can build suitable applications according to the needs of different platforms, through the target compiler and C + + compiler, using the above control to construct the dependence of various components on the robot visual development environment.

IV. DESIGN EXAMPLE

To validate the proposed component-based framework for robot software architecture, we design an application example: a visual procedure to track the ball for mobile robot. In the example, the platform includes a PC and a mobile robot system. PC connects the mobile robot system using wireless network. The mobile includes vision sensors, mobile chassis, and so on. The hardware and software configuration of the robot controller and PC is shown in Table 3.

The camera collects image environment to identify a specific color ball and guides the robot to track the ball. The example relates to the visual component and the robot chassis component.

Table 3 The Hardware and Software Configuration

	OS	CPU	Memory	Wireless LAN
Robot Controller	arm-linux 2.6.9	Intel X-Scale PXA270A ARM	64M SDRAM	VIA VNT6656GUA00
PC	Windows XP	Processor 0: Intel(R) Pentinum(R) 4 CPU 3.00GHz Processor 1: Intel(R) Pentinum(R) 4 CPU 3.00GHz	1001.9Mi B	TP-LINK TL-WN322G+ 54 Mbps

A. *Visual component*

Visual component is the abstract of a certain machine vision module. In the example, the visual component VS defines the recognition of a particular object, like robot soccer recognizes football. It often recognizes relative coordinate information of objects in the camera in accordance with the color of objects. And the interface definition of visual components is shown in Table 4.

Table 4 Visual Component Interface

Interface	Function	Communication Mode
int initialize()	Initialize visual component	Request/response mode
int setThreshold(unsigned int, unsigned int, unsigned int, unsigned int)	Set threshold	Request/response mode
void start()	Start recognition	Command mode
void stop()	Stop recognition	Command mode
ObjectPositionEvent	Return the location information of objects	Event mode

The camera in the robot system uses the color recognition method based on threshold vector to recognize objects. The recognition process and state transition diagram are separately shown in Fig.5 and Fig.6.

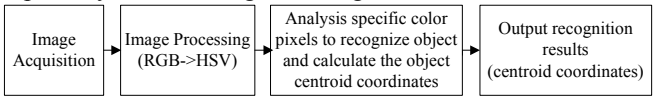

Fig.5 The color recognition process based on threshold vector

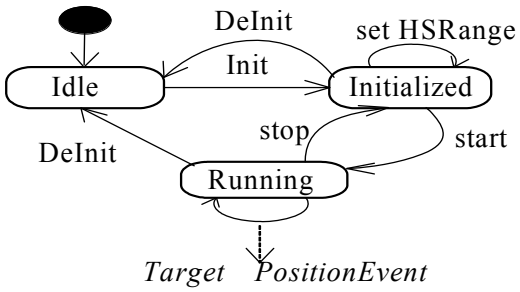

Fig.6 Visual component state transition diagrams.

B. Control Program Design

The function of control program is to guide the robot to make the ball always at the center position of robot camera. (x, y) is the object location coordinates returned by visual component, if (x, y) is in the left side of the center (x_0, y_0), the robot should turn left; instead, turn right; if (x, y) is in the top of the center, the robot should move forward; instead, move backward. We separately deal with the x and y values and figure out the velocity of the chassis, and then superimpose the calculated speed to get the actual velocity of the chassis. That is:

$$\begin{cases} V_{Lx} = k_1(x - x_0) \\ V_{Rx} = -k_1(x - x_0) \\ V_{Ly} = V_{Ry} = k_2(y - y_0) \\ V_L = V_{Lx} + V_{Ly} \\ V_R = V_{Rx} + V_{Ry} \end{cases}$$

With adder, multiplier and comparator controls, describing the above formula can realize the design of visual procedures to recover the ball.


Fig.7 Result of the example.

V. CONCLUSION

To achieve the industrial development of robot technology and reduce the difficulty of robot system design, the modular design of robot system and component-based design of robot software is an inevitable trend in robot system design. In this paper, we presented a component-based design framework of robot software architecture, set up the static model and run-time model of robot software component, and defined the Interface Description Language of robot component. In this paper, we designed a lightweight robot middleware to achieve components communication according to the component model. In order to facilitate the development of robot systems in the design framework, we also designed the robot component tools and the robot visual assembly environment. To verify the feasibility of design framework of robot software architecture, we used component-based design methods to realize a visual procedure to track the ball. In the further work, we plan to transplant the software architecture into real-time bus and real-time operating system to satisfy real-time control requirements of robot system.

ACKNOWLEDGMENT

This work is supported by the 863 Program of China (2007AA041701 and 2007AA041702), National Natural Science Foundation of China (Grant No. 60525314), and the 973 Program of China (2002CB312204-04)

REFERENCES

[1] Mizukawa M. Robot technology (RT) trend and standardization [A]. In: Proceedings of IEEE Workshop on Advanced Robotics and its Social Impacts[C], 2005: 249-253
[2] Gill C, Smart B. Middleware for Robots? [A]. In: Proceedings of AAAI Symposium Workshop on Intelligent and Distributed Embedded Systems[C], 2002: 1-5
[3] Fujita M, Kageyama K. An open architecture for robot entertainment [A]. In: Proceedings of International Conference on Autonomous Agents [C], 1997: 435-442
[4] Albus J S, Mcain H G, Lumia R. NASA/NBS standard reference model for telerobot control system architecture (NASREM)[R]. Technical Report 1235. National Institute of Standards and Technology, Gaithersburg, MD, 1989
[5] Albus J S. A theory of intelligent machine systems[C]. In: Proceedings of IEEE/RSJ International Workshop on Intelligent Robots and Systems IROS'91,vol. 1, Osaka, Japan, 1991: 3-9
[6] Proctor F M, Albus J S. Open-architecture controllers [J]. IEEE Spectrum, 1997, (6): 60-64
[7] Saridis G N. Architecture for intelligent controls[C]. In: Proceedings of IEEE Symposium on Implict and Nonlinear Systems, Ft. Worth, TX, 1992: 13-25
[8] Fernandz J A, Genzalez J. The nexus open system for integrating robotics software [J]. Robotics and Computer Integrate Manufacturing, 1999, 15: 430-440
[9] Choi D H, Kim S H, Lee K K, et al. Middleware architecture for module-based robot[A]. In: Proceedings of SICE-ICASE International Joint Conference [C], 2006: 4202-4205
[10] Mizukawa M, Matsuka H, Koyama T, et al. ORiN: open robot interface for the network - the standard and unified network interface for industrial robot applications [A]. In: Proceedings of SICE Annual Conference [C], 2002: 1160-1163
[11] Bruyninckx H. Open robot control software: the OROCOS project [A]. In: Proceedings 2001 ICRA. IEEE International Conference [C], 2001: 2523 – 2528
[12] Utz H, Sablatnög S, Enderle S, et al. Miro - middleware for mobile robot applications[J]. IEEE Transactions on Robotics and Automation, 2002, 18(4): 493-497
[13] Ando N, Suehiro T, Kitaqaki K, et al. RT-middleware: distributed component middleware for RT (robot technology) [A]. In: Proceedings of Intelligent Robots and Systems, IEEE/RSJ International Conference [C], 2005:3933-3938
[14] JACKSON J. Microsoft Robotics Studio: A Technical Introduction [J]. Robotics & Automation Magazine, IEEE, 2007, 14(4): 82-87
[15] Jones J J. Robot Programming: A Practical Guide to Behavior Based Robotics [M]. Beijing: China Machine Press, 2006: 63-86
[16] TrollTech Corp. Qt Reference Documentation [OL]. http://doc.trolltech.com/4.3/index.html
[17] Tong Shizhong. The modular principle design method and application [M]. Beijing: China Standard Press, 2000
[18] Wang Zhijian, Fei Yukui, Lou Yuanqing. Software Component Technology and Application [M]. Beijing: Science Press, 2005: 195-197