Development and Execution of OProS Components

Seungwoog Jung, Byoungyoul Song, Choulsoo Jang, Sungmoon Kim and Michael E. Shin

1 Electronics and Telecommunications Research Institute, 218, Daejeon, KOREA
(Tel: +82-042-860-6441; E-mail: {swjung, sby, jangcs, saint}@etri.re.kr)
2 Department of Computer Science, Texas Tech University, Lubbock, TX 79409-3104, USA
(Tel: +01-806-742-3527; E-mail: michael.shin@ttu.edu)

Abstract: The Open Platform for Robotic Services (OPRoS) supports the full life cycle for robot software development and execution by providing the robot software component model, component execution engine, various middleware services, development tools, and simulation environment. This paper describes the OProS component development process and the software architecture of OProS component execution engine executing robot components.

Keywords: robot software, component, component execution engine, OProS

1. INTRODUCTION

Recently intelligent service robots have gained much attention for overcoming the saturation of the industrial robot market. Intelligent service robots perform various tasks by continuously interacting with humans in an uncertain and dynamic environment, whereas most industrial robots continuously repeat their jobs in a static and structured environment [1]. Successful commercialization of intelligent service robots may require that the manufacturing cost of each intelligent robot needs to be reasonably low and the robots provide various intelligent functions such as user recognition and autonomous navigation. However, technically, the complexity of robot software has increased more and more in recent years in order to implement intelligent services under competitive manufacturing costs.

Recently component-based development [2] has gained a lot of attention as an emerging approach in robotics to reducing the complexity of robot software. The component-based robot software development can realize high productivity and quality of robot software by lessening the software complexity by means of modularity, reusability, and easy integration and maintenance. There have been several research activities on component-based robot software platforms, such as Robot Technology Component (RTC) [3], Open Robot Control Software (OROCOS) [4] and Mobile and Autonomous Robotics Integration Environment (MARIE) [5].


Open Robot Control Software (OROCOS) [4] is a C++ framework, targeting the implementation of control systems. In particular, it provides a real-time toolkit for various real-time control applications, and allows application designers to build highly configurable and interactive component-based real-time control applications.

Mobile and Autonomous Robotics Integration Environment (MARIE) [5] is a middleware framework for building robot software systems by integrating existing and new software components. It uses a generic communication framework to create a flexible distributed component system that allows robotics developers to share software programs and algorithms.

This paper describes the OProS component development process and the software architecture of OProS component execution engine that is required to execute robot components. The rest of this paper is organized as follows. In section 2, the OProS component development process is described. Section 3 explains the component execution engine. Section 4 concludes this paper.

2. OPROS COMPONENT DEVELOPMENT

As a software platform for intelligent robots, the OProS platform [7, 8, 9] supports the full development life cycle for robot software. It provides the robot software component model, component execution engine, various middleware services, development tools and simulation environment. When developing robot components using the OProS, developers can use two kinds of patterns: one is to request services of a component through method invocations; the other is to
send and receive data or events.

In the OPRoS component, the service invocation or data/event exchange is performed through the port mechanism. A component has one or more ports as its interface. The OPRoS component model specifies three types of port - service port, data port, and event port - to support robot software development patterns. A service port provides a set of methods that other components can invoke. A data port is used for exchanging data between components. An event port transmits events to other components. Although data ports and event ports transmit similar data, they are different. Events are processed immediately when received, whereas the received data at a data port are buffered and then processed later.

A robot application is made of several components connected each other. The connections between components are established between the ports of components. An OPRoS component is classified as an atomic component or composite component. A composite component can contain atomic or other composite components.

Figure 1 depicts the OPRoS component development process. The component editor tool is used to develop an atomic component. The tool generates a component package composed of a component binary (DLL or shared object) and a component profile. A component profile describes the characteristics of a component, such as component identifier, execution semantics, and information of component ports. The generated component package can be stored in a component repository.

The component composer tool (Fig. 1) is used to assemble and compose components to a robot application that includes interconnection between components. The tool also generates an application profile that describes the characteristics of an application, such as application identifier and connection between components. The robot simulator (Fig. 1) can be used to simulate or debug the robot application. A robot application can be deployed to the component execution engine that executes the application.

![Fig. 1. OPRoS component development process](image-url)
3. OPROS COMPONENT EXECUTION ENGINE

The OPROS component execution engine is responsible for management and execution of components. The engine provides robot component developers with several functionalities, such as thread management, resource allocation, and state management, so that the developers can concentrate on the core algorithm of components. This ensures the efficient development of robot applications.

Figure 2 shows the architecture of the OPROS component execution engine. The engine runs on either MS Windows or Linux OS supporting X86 architecture and ARM processor. A robot application package generated by the component composer is deployed to the engine. The component deployer in the engine receives the package and stores it in the local repository. After storing, the component deployer reads the application profile in the package and registers the component profiles in the application package to the component manager. The component composer can monitor the status of running components through the component monitor.

The component manager interprets the component profiles, instantiates the participating components onto memory, establishes port connections between components and starts the execution of the components. Components are executed periodically or non-periodically according to the execution mode described in their component profiles. A component is executed by an executor that is a logical thread. Whereas a periodic component is executed periodically, a non-periodic component is executed just once. The executor manager allocates an executor to the components which have the same cycle time. This removes unnecessary context switching between executors (threads) so that performance of the system is enhanced.
The OPRoS component execution engine has a fault recovery mechanism that self-manages faults in the engine. The component execution engine may encounter faults while it runs user components. The faults need to be handled appropriately so that the engine should not fall into failures. The fault recovery mechanism detects faults in the engine and repairs them autonomously to avoid the engine failures. In particular, the fault recovery mechanism focuses on the reliability of executors encapsulated in the engine. Some executor that executes components with the same cycle time may not be able to finish executing all the components within the cycle time. This causes a violation of the timeliness of the components.

Figure 3 depicts the overview of OPRoS fault recovery mechanism [10]. Each executor is monitored to detect the timeliness violation of components being executed with a cycle time. When an executor finishes processing a component, it notifies the executor monitor checking whether the components assigned to the executor can be finished completely within the cycle time. If an executor may not finish processing the assigned components within a cycle time, the components assigned to the executor are split and reassigned to a new executor.

4. CONCLUSIONS

This paper has described the OPRoS component development process and the component execution engine. The OPRoS supports the full life cycle for robot software development by means of the robot software component model, component execution engine, various middleware services, development tools and simulation environment. The OPRoS provides various tools for developing robot components and robot applications. The OPRoS component execution engine executes robot application systems composed of software components developed using the robot software component model.

The real-time is essential in some robot applications. Even though the OPRoS component execution engine supports soft real-time scheduling feature on Windows and Linux, it needs to support hard real-time scheduling feature. In the near future, we plan to enable the OPRoS engine to provide the hard real-time scheduling feature using the kernel patch, such as RTAI on Linux or device driver level scheduler on Windows. Also, the OPRoS is planned to be ported into a commercial real-time OS such as QNX.

The OPRoS is an open source project. The source codes and documents of the OPRoS can be found through a Web site (http://www.opros.or.kr).
ACKNOWLEDGMENT

This work was supported by the Industrial Foundation Technology Development Program of MKE/KEIT, Rep. of Korea [10030826, Development of Reliable OPRoS Framework].

REFERENCES


