An Introduction to Robot Component Model for OPRoS(Open Platform for Robotic Services)

Byoungyoul Song, Seungwoog Jung, Choulsoo Jang, Sunghoon Kim

u-Robot Reserch Division,
Electronics and Telecommunications Research Institute
138 Gajeongno, Yuseong-gu, Daejeon, KOREA
{sby, swjung, jangcs, saint}@etri.re.kr

Abstract. The OPRoS(Open Platform for Robotic Service) is a platform for network based intelligent robots supported by the IT R&D program of Ministry of Knowledge Economy of KOREA. The OPRoS technology aims at establishing a component based standard software platform for the robot which enables complicated functions to be developed easily by using the standardized COTS components. The OPRoS provides a software component model for supporting reusability and compatibility of the robot software component in the heterogeneous communication network. In this paper, we will introduce the OPRoS component model and its background.

Keywords: OPRoS, robot component, robot component model, composite component, port

1 Introduction

The intelligent service robot has gained much attention recently for overcoming the saturation of the industrial robot market [1]. Most of industrial robots repeat continuously their jobs in a fixed and static environment. But, intelligent service robots interact continuously with humans in uncertain and dynamic environment and they have to perform appropriate tasks for various and many requests from their users [2]. The past stand-alone robots lacked expandability, compatibility and reusability because the interfaces between robot software units were not standardized and robot software modules were too dependent on their hardware devices. For that reason, developing a new robot has been started from the scratch. While interacting with humans, the intelligent service robot can provide various services such as housekeeping, education, entertainment, public services based on the information technology, the artificial intelligence and the network technology. The intelligent service robot decides and performs appropriate services for users based on sensor data from the changing surroundings and the information obtained from the network. Wherever the intelligent service robots are connected to the network, the user can control the robot over the network.

In order to be successfully commercialized, the intelligent service robot should have low manufacturing costs and must be able to provide various intelligent

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

functions such as user recognition and autonomous navigation. But, it is actually hard that one robot company develops all the complicated functions of the robot from device driver to recognition software. If there are enough COTS (commercial off-the shelf) for various functions, it should be easy that a company develops a new robot product with various and complicated functions.

The OPRoS(Open Platform for Robotic Service) is a platform for network based intelligent robots supported by the IT R&D program of Ministry of Knowledge Economy of KOREA. The OPRoS technology aims at establishing a component based standard robot software platform which enables complicated functions to be developed easily by using the standardized COTS components. To acheive that goal, we analyzed the features of robot system as follows.

- **Distributed system**: There are many computational nodes in a robot system
- **Parallel processes**: Multi processes on many nodes may execute concurrently.
- **Real-time system**: Reactiveness of a robot is guaranteed by real-time system
- **Event driven system**: Publish/subscribe model is used.
- **Limited resource**: Efficient sharing of limited resources between processes
- **Periodic data transfer**: Classical control loop - most sensors and actuators transfer data periodically
- **Remote procedure call**: High level control is accomplished with the remote procedure call

Also we defined the key requirements for the OPRoS as follows.

- **Easy to use**: OPRoS should provide convenience to robot developers. OPRoS based programming should not be a burden on robot program.
- **Robot S/W architecture independent**: OPRoS component should not incline to any specific robot S/W architecture such as SPA (Send-Plan-Action), subsumption, hybrid and so on. But, it should be possible to implement various robot S/W architecture using OPRoS components.
- **Distributed, but middleware independent**: OPRoS should be used in distributed environments but independent on the using middleware(CORBA, RPC, PLANET, etc)
- **Data Driven Control and Remote Procedure Call:** OPRoS should support data exchange model for the classical control loop and method invocation of remote component for the high level control.
- **Simple, but Expandable**: The OPRoS component has to be a single component form. The larger component should be made by composing components
- **Execution Semantics**: OPRoS should support various execution semantics such as Periodic, Dedicated, FSM based stimulus-response and so on.
- **Robustness**: Fault detection and recovery capability are necessary
- **Realtime & QoS**: Realtime and QoS supporting are also necessary

Based on these features and requirements, we defined a robot software component model for the OPRoS.

In this paper, we introduce the OPRoS component model and its features. . First, the existing research trend about the robot component model will be presented. Next, we will explain the OPRoS component model in the chapter 3. Finally, a conclusion is presented in the chapter 4.

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

2 Related works

Recently, researches on the component-based robot software development are being in active progress. This section describes existing researches on robot software component model.

2.1 RTC

RTC (Robot Technology Component) [3] is the OMG robot software component standard submitted by AIST of the Japan and RTI of U.S. in the Robotics DTF of OMG in September 2006. The RTC is composed of execution semantics, introspection and lightweight RTC. The lightweight RTC is a simple model containing definitions of concepts such as component, port, and the like. The execution semantics is extensions to the lightweight RTC to directly support critical design patterns used in robotics applications such as periodic sampled data processing, discrete event/stimulus response processing, modes of operation, etc. The introspection is an API allowing for the examination of components, ports, connections, etc. at runtime [3].

2.2 OROCOS

The OROCOS (Open Robot Control Software) [4] is initiated with the open source project for the robot control software development at December 2000. The most of real-time control applications have been developed on the specific operating system and was not able to provide the systematic structure for the feed-back control. Of course, the control system design toolkit like the Simulink [5] exists. However, because of being applied to the large-scale distributed surroundings and the various hardware platforms, it has limits of distributed event processing and synchronous/asynchronous task programming. The OROCOS provides the real-time toolkit (RTT) in order to develop the various real-time control applications. The Orocos RTT provides a C++ framework, targeting the implementation of (realtime and non-realtime) control systems. The Real-Time Toolkit library allows application designers to build highly configurable and interactive component-based real-time control applications.

2.3 MIRO

Miro which was developed at Ulm university of Germany is a distributed object oriented framework for mobile robot control, based on CORBA (Common Object Request Broker Architecture) technology [6]. MIRO use TAO(The ACE ORB), which is developed under the aid of ACE, as their ORB. MIRO consists of 3 layers such as Miro Device layer, Miro Service layer, and Miro Class Framework. Miro Device layer provides classes, by which low-level control board connected through the serial link such as CAN bus can be accessed, to upper layers. Miro Service layer defines service interface for accessing sensors and actuators of robot by using CORBA IDL, and provides those services as CORBA objects. Miro Service layer is

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

independent of underlying platform while Miro Device layer is dependent. Miro Class Framework provides functional modules for control of robot such as mapping, localization, behavior engine, path planning and so on.

3 OPRoS Component Model

The robot components, which can be accessed through their interface, are reusable and replaceable software modules. The component user uses only the interface which a component provides, so he doesn't care about the detail implementation of the component. That is, the component is considered as a black box which anything but its exposed interfaces need not be known of. Accordingly, the internal embodiment can be changed freely while its external interfaces remain unchanged.

The robot application is a kind of the combination of the components. And the component profile which describes the information including characteristics and external interfaces of a component, and etc. is offered together for the combination.

The OPRoS component has two types. One is the atomic component and the other is the composite component. The composite component is regarded as an atomic component with collection of connected components. The atomic component has several features to support RPC, data flow mechanism and event driven mechanism. In this chapter, we describe about the OPRoS component, its features and its development process.

3.1 Atomic Component

In this section, we describe features of the atomic component. The atomic component should be designed to support development patterns frequently used during the robot application development in order to provide the usability of the robot component.

When developing a robot, there are two kinds of development pattern. The first pattern is to invoke the methods or read/write the attributes of a component. The second is to send and receive data or events each other while performing its own task.

The OPRoS component model supports these two patterns for the developing of robot components. In the OPRoS component, the method invocation or data / event exchange is performed through the port between components. There are three types of ports in OPRoS, which are method port, data port, and event port. The component is able to have several ports of each when necessary.

Figure 1 depicts the model of the atomic component of OPRoS. Through a method port, a component invokes the methods and accesses attributes which the other component provides. On the other hand, data or event can be delivered through a data port and an event port. Because method invocation and data/event delivery are carried out through ports, ports of target component should be known in order to interact. Component container sets up the relations between the ports of components according to component profiles which describe the required ports and the provided ports of a component while initiating the component.

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

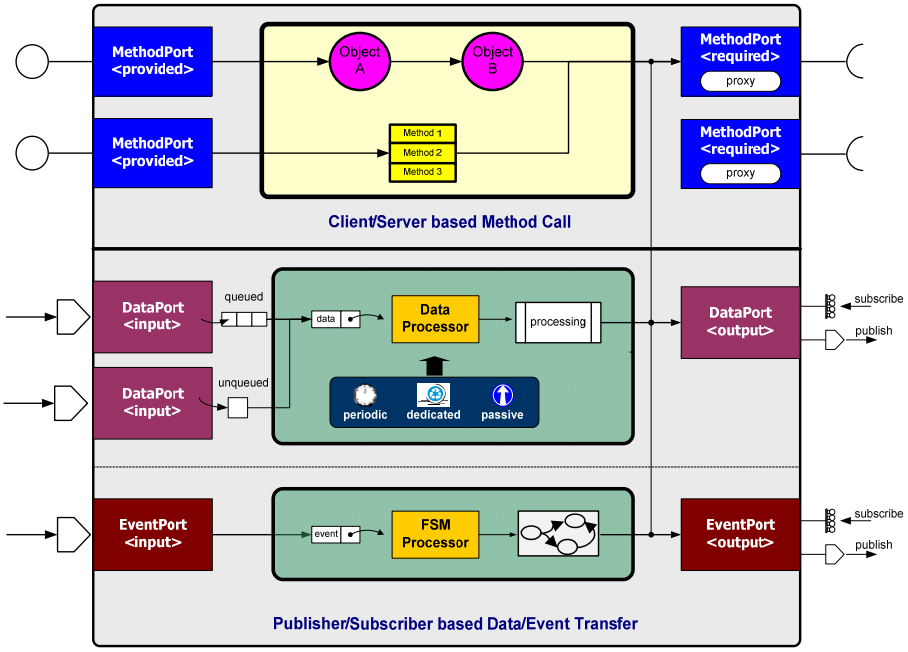

Fig. 1. Atomic component model

OPRoS component model supports the client/server mechanism for control flow and the publisher/subscriber mechanism for data/event flow. The client/server mechanism is used in method ports, and the publisher/subscribe mechanism is used in data/event ports. In the client/server mechanism, a client calls a method of a component and receives the result of invocation. On the other hand, multiple publishers can deliver data or event to multiple subscribers in the publisher/subscriber mechanism.

Lifecycle of a component such as initialization, start, stop, destroy is managed by the component container. When managing the lifecycle of a component, the component container notifies the event so that component can allocate or release resources which component uses. Interfaces between component and component container are required for this contraction. Besides these interfaces, there are various interfaces between them such as lifecycle interfaces, port management interface, property interface, method invocation interface, attribute access interface, data delivery interface, event management interface.

As the calling component might reside in the different node, the interfaces which are called by other components should be invoked remotely. The lifecycle of a component is managed by the lifecycle manager of component container. Corresponding to each lifecycle API, there are callback methods which are called when the state of a component is changed such as onInit(), OnStart(), onExecute(), and so on.

A component has a component profile describing its characteristic. And a component is comprised of one or more port. A component which is called as a

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

composite component can include the other component inside. The user component to be developed inherits the base classes of OPRoS component model, and has to be realized by adding user codes. Figure 2 shows a conceptual class diagram of the OPRoS component model.

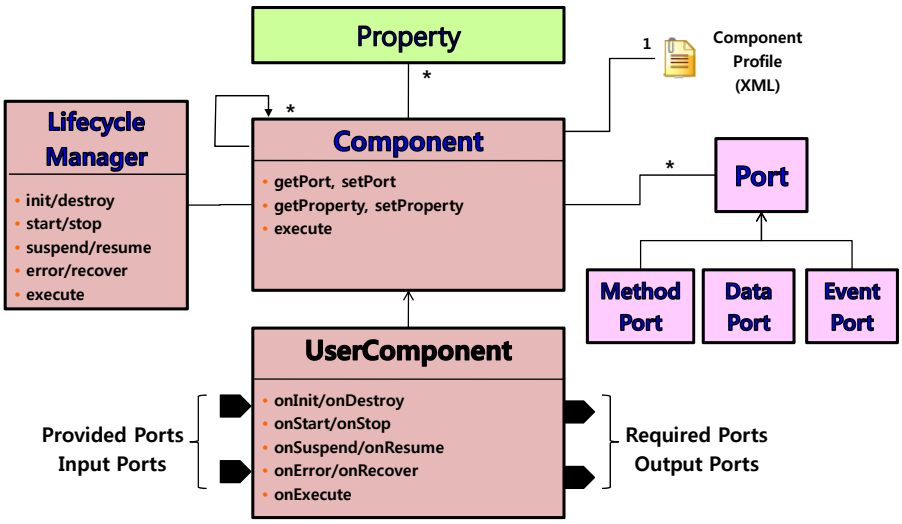

Fig. 2. Conceptual class diagram of the OPRoS component model

Each component has method ports for the life cycle management and monitoring as mandatory. Lifecycle APIs such as init, start, stop, suspend, resume can be called using the lifecycle method port. And, through the monitoring method port, the state of a component can be monitored.

3.2 Method Port

The method port is organized by a set of methods which a component provides. A component user can invoke the method which the method port provides. The required method port and the provided method port have to be organized by the same method profile. That is, in Figure 3, the method port A1 of the component A and method port C1 of the component C have to be comprised of the same method profile.

As to method port, multiple required ports can connect to one provided port. And, method port uses client/server model in which client invokes the methods provided by server through the port.

Method port can support two modes: blocking call mode and non-blocking call mode. In blocking call mode, the component calling a method waits until the result is returned. But, in non-blocking call mode, the component calling a method of a port doesn't wait return value and goes to next operation immediately. This is possible only when the method has no return value.

The method port is described in the method profile which is a XML file. And the component development tool analyzes the method profile and automatically produces

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

proxy code for the required port and skeleton code for the provided port. Proxy performs the role of transmitting a request to the provided port. Skeleton code receives the request and delegates it to the implementation of the requested method.

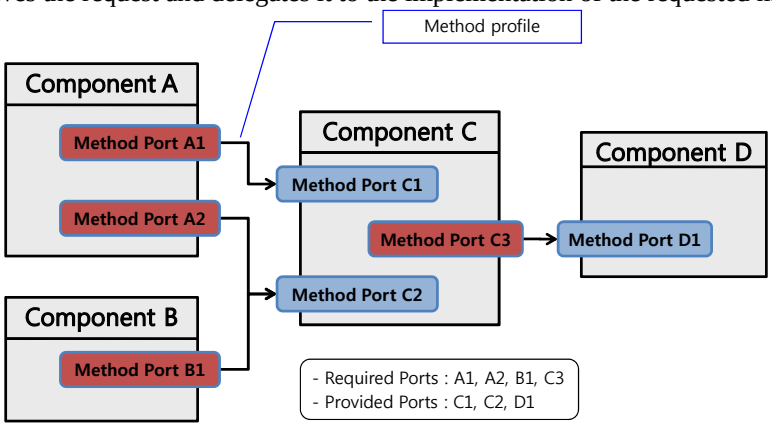

Fig. 3. Connections between method ports

Figure 4 depicts an example of producing phase of a provided port. According to the method profile, many codes such as code for registering each method to the component, code for interface dispatching, and template code for user-defined implementation are generated. After user-defined code is completed, every codes and profiles are compiled and linked to the component package.

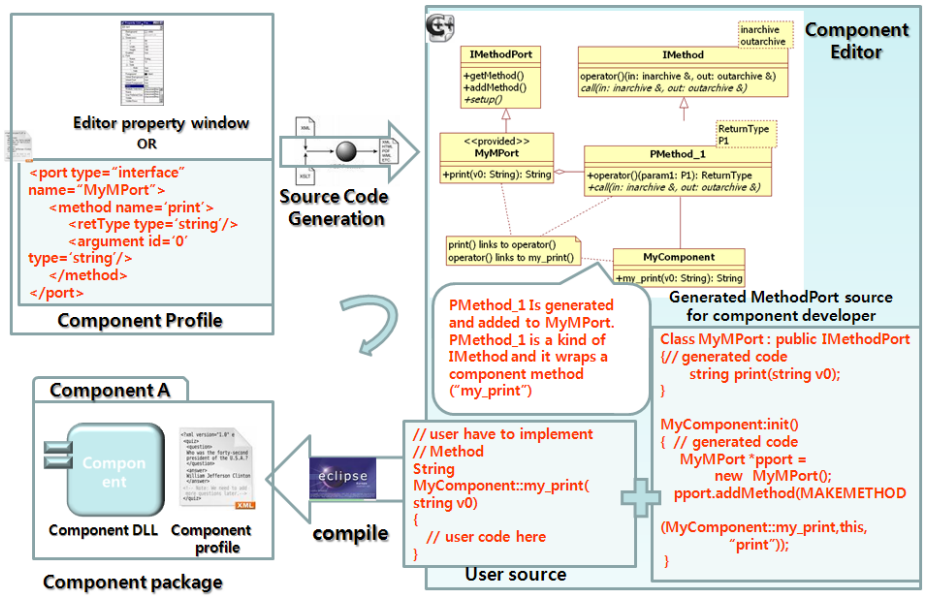

Fig. 4. The producing process of a provided port

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

It is supposed that the required port is connected to the corresponding provided port at runtime. It is possible that corresponding provided ports are those of local component or remote component. Direct method invocation takes away the control privilege of component container. Thus, it is treated like a remote one, even if local component.

3.3 Data Port

The data port is a port for exchanging data. The output data port is used for sending data, whereas the input data port is used for receiving. The output data ports of a component can send data to the input data ports with the same data type of other components. The data port supports the publisher/subscriber model which supports n:m data transmission. A data port can be a publisher or a subscriber. A publisher sends data to all registered subscribers. A subscriber can receive data from all connected publishers. A data port has a queue to store transmitted data whose size is described in the component profile. Only non-blocking call is possible for sending data through the data port.

3.4 Event Port

The event port is a port for transmitting events. The received events are processed by the component container by allocating a thread. The received events are used to process the FSM(Finite State Machine) associated to the target component.

Whereas the data sent to a component through a data port is queued and processed later by the component thread, the received events are processed immediately by the thread allocated by the component container.

The events sent to a component are used only for processing the FSM associated to the component. When an event is received to a component, the component container tries to transit the current state to the next state of the FSM based on the receive event. The states and transition rules of the FSM are described in the component profile. The component container processes the FSM based on the description of the component profile. Each state has several transition rules. The FSM developer should implement the entry action, the exit action and state action of each state. The entry action is executed when entering the state while the exit action is executed when exiting the state. The state action is executed during staying the state.

3.5 Composite Component

The composition of components is what assembles several components and makes a new component. The component including several components in the inside is called as composite component. The composite component provides the function of abstracting several components as one component. And, from outside of a composite component, the interface of each component which is sub-component of the composite component cannot be accessed, but the interface which the composite component allows can only be accessed. If the interface of the composite component is called, the composite component calls the interface of sub-components and returns

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

the result. That is, the composite component abstracts the complicated interfaces of interior components and provides the simple interfaces to outside.

There are some ways of composing component; the first is a connection-oriented composition, the second is hierarchical composition, and the last is hybrid composition combining former two ways. Figure 5 shows the hybrid combination mixing the hierarchic composition and the connection-based composition. The OPRoS component model supports the hybrid combination. The top level composition is connected based on the connection-based configuration, and top-level component can be a composite component or an atomic component.

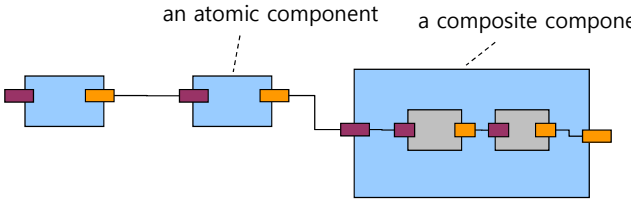

Fig. 5. Hybrid composition

3.6 State transition of a component

A component has a state which is managed by the component container. Figure 6 shows the state transition diagram of a component.

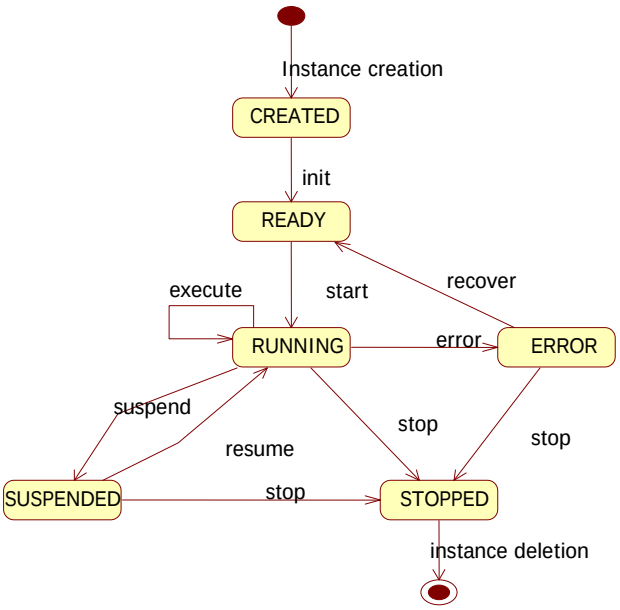

Fig. 6. State transition diagram of an OPRoS component

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

A component becomes CREATED state after the instance of a component is created. In the CREATED state, if the init() method of a component successes, the state changes to READY in which the execution of a component is ready. In the READY state, if the start() function is called, the state changes to the RUNNING state where the execution of a component is possible. In the RUNNING state, if the suspend() function is called, it goes to the SUSPENDED state where the execution of a component is suspended for some time. If the resume() method is called out from the SUSPENDED state, it is transmitted to the RUNNING state again. In the RUNNING state, if an error occurs and the error() function is called out, the component state transits with the ERROR state. And if the recover() function is called from the ERROR state when the error is recovered, the state becomes READY state. In RUNNING, SUSPENED, and the ERROR state, it goes to the STOPPED state if the stop() function is invoked. And the component instance can be removed from the STOPPED state.

3.7 OPRoS component classes structure

Figure 7 shows the UML class diagram of OPRoS component. It shows relations of component base class (IComponent) and other classes (IDataPort, IMethodPort, etc)

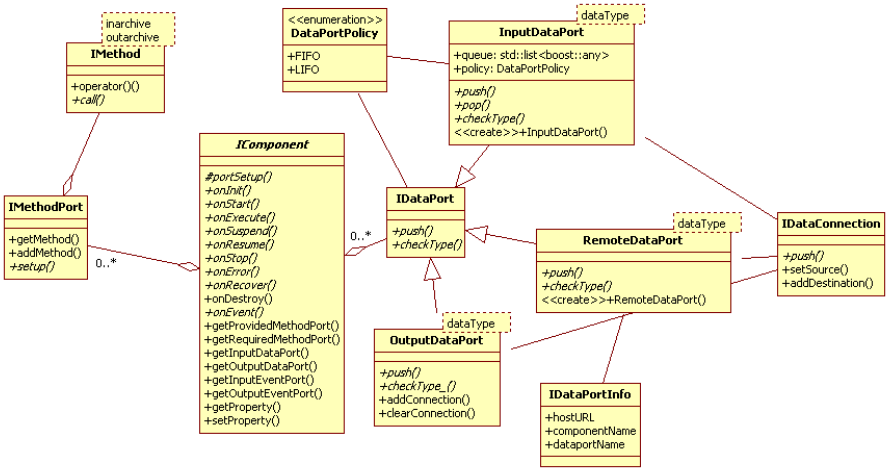

Fig. 7. OPRoS component classes' diagram

The OPRoS component consists of IComponent which is a component itself, IDataPort which supports data flow, IMethodPort which enables RPC call and IDataConnection which provides protocol-independent communications. All user components inherit component base class IComponent and register their method ports which inherit IMethodPort and data ports which inherit IDataPort. The component container makes some data connections between components in runtime.

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

3.8 OPRoS component development process

Figure 8 shows the component development process under the OPRoS environment. The OPRoS component environment is consist of a component editor, a robot simulator, a component composer and a component container.

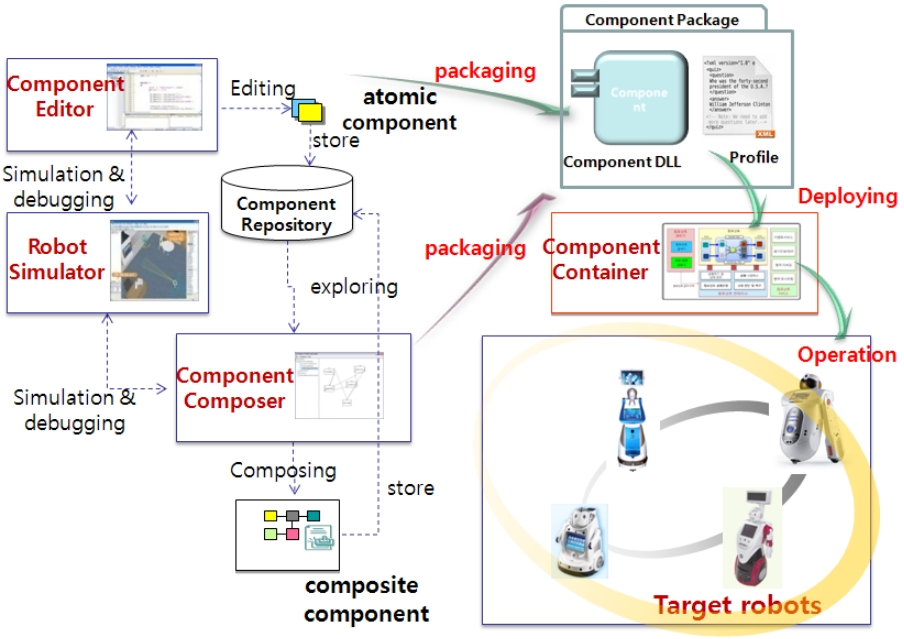

Fig. 8. OPRoS component development process

The component editor helps a developer make an atomic component. If a developer makes a profile for his component, the component editor generates base sources of the component with the profile. The component composer helps a developer make a composite component by exploring the component repository.

The robot simulator helps a developer combine a virtual robot and test the robot in virtual environment. Through this simulation function, the component editor and component composer can debug their components.

When components are made completely, the component editor or the component composer deploys those components to the component container installed in a robot.

The component container loads and executes components deployed and the robot operates tasks or gives services based on deployed components.

4 Conclusion

In this paper, we described the OPRoS component model which provides the component based robot software development model for interoperable, portable and reusable robot applications in distributed, heterogeneous environments. The OPRoS

Workshop Proceedings of SIMPAR 2008
Intl. Conf. on SIMULATION, MODELING and PROGRAMMING for AUTONOMOUS ROBOTS
Venice(Italy) 2008 November,3-4
ISBN 978-88-95872-01-8
pp. 592-603

component supports the method port for RPC model, the data port for dataflow model and the event port for event-driven model.

We are developing a component container and a component IDE including a component composing tool and a simulation tool for the OPRoS component.

We expect the OPRoS component model can contribute to cost reduction and rapid prototyping of robot development. In the future work, we will develop fault detection and recovery technology, real-time QoS supporting technology, remote debugging and monitoring technology.

Acknowledgments. This work was supported by the IT R&D program of MKE/IITA. [2008-S-030-01, Development of RUPI-Client Technology].

References

1. Seung-Ik, Lee. , Choul-Soo, Jang., Sung-Hoon, Kim. , Myung-Chan, Roh., Beom-Su, Seo.: Issues and Implementation of a URC Home Service Robot. 16th IEEE International Conference on Robot & Human Interactive Communication, KOREA (2007)
2. Seung-Ik, Lee., Sung-Hoon, Kim., Woo-Young, Kwon., Joong-Bae, Kim.: uFlow: A Service-Oriented Task Modeling Architecture for Home Service Robots. The 12th IASTED International Conference on Robotics and Applications, Hawaii, USA (2006)
3. The Robotic Technology Component Specification. OMG Adopted Specification, ptc/06-11-07 (2006)
4. OROCOS, http://www.orocos.org/
5. Simulink, http://www.mathworks.com/
6. Miro Manual, Version 0.9.4, Jan 10 (2006)
7. Blum S, A.: From a corba-based software framework to a component based system architecture for controlling a mobile robot. Lecture notes in computer science. vol. 2626, pp. 333--344 (2003)
8. Cabrera-Gamez, J., Dominguez-Brito, A., Hernandez-Sosa, D.: Coolbot: A component-oriented programming framework for robotics. Lecture notes in computer science. vol. 2238, pp. 282--304 (2000)
9. Sukhatme G, S., Matarik M, J.: Robots: intelligence, versatility, adaptivity. Communications of the ACM. vol. 45, issue 3, pp. 30--32 (2002)
10. Seung-Woog, Jung., Seung-Ik, Lee., Sung-Hoon, Kim.: The study on the RUPI client integrated software platform. vol. 25, issue 4, pp. 22--29 (2008)
11. Seung-Woog, Jung., Seung-Ik, Lee., Sung-Hoon, Kim.: The study on the robot software platform for network robots. vol. 26, issue 4, pp.382-47 (2008)