Journal of Institute of Control, Robotics and Systems (2014) 20(12):1252-1258
http://dx.doi.org/10.5302/J.ICROS.2014.14.8024 ISSN:1976-5622 eISSN:2233-4335

# ROS 토픽과 결합 가능한 OPRoS 프레임워크의 이벤트 포트 확장 개발

# Event Port Extension of OPRoS Framework for Inter-connecting with ROS Topic

장 철 수*, 송 병 열, 김 성 훈
(Choulsoo Jang[1,*], Byoungyoul Song[1], and Sunghoon Kim[1])
[1]Intelligent Cognitive Technology Research Department, Electronics and Telecommunications Research Institute

**Abstract:** ROS is based on a graph architecture where processing takes place in nodes. Nodes communicate together by passing messages through topics based on the publish/subscribe model. On the other hand, OPRoS components know each other and are tightly-coupled via port connections, and different coupling schemes make the interoperation between two platforms difficult. This paper describes an extension of OPRoS framework to support the interoperation with the ROS topic.

**Keywords:** OPRoS, ROS, robot component, robot middleware

## I. 서론

로봇은 인식, 지능, 제어 등 다양한 하드웨어 및 소프트웨어들의 결합체이므로 이들을 통합하기 위한 플랫폼이 필수적이다. 특히 컴포넌트 기반의 소프트웨어는 레고 블록을 조립하듯이 로봇 개발에 쉽게 재활용될 수 있어 로봇 개발 생산성을 높일 수 있고, 개발 프로세스를 표준화하여 재사용 가능한 로봇 소프트웨어 개수를 빠른 속도로 늘릴 수 있어서 로봇 산업을 급격히 증대시킬 수 있는 장점이 있다. 현재 여러 로봇 선진국들에서는 각국의 로봇 산업 특성에 맞는 다양한 로봇 플랫폼을 개발하여 보급 확산에 주력하고 있는 실정이며[1-5], 한국에서는 2008년부터 서비스 로봇에 특화된 컴포넌트 기반의 개방형 로봇 소프트웨어 플랫폼인 OPRoS (Open Platform for Robotic Services) [6,7]를 개발하여 로봇 생태계를 조성하고 있다.

OPRoS와 마찬가지로 ROS (Robot Operating System) [8] 또한 컴포넌트 기반 로봇 소프트웨어를 개발하는데 사용되는 플랫폼으로, 2007년 스탠포드 대학에서 AI Robot 프로젝트를 지원하기 위해 개발되었으며 2008년도부터 로봇 벤처 Willow Garage에 의해 지원 및 확장 개발되어 점차 많은 로봇 소프트웨어 컴포넌트들이 ROS 기반으로 만들어지고 있다[9].

컴포넌트 기반 로봇 소프트웨어 개발에서 로봇 서비스는 컴포넌트들을 서로 연결하고 통신을 수행하도록 구성되는데, OPRoS와 ROS 상호간에 컴포넌트가 호환되어 사용될 수 있다면 컴포넌트 기반 로봇 소프트웨어 활성화에 많은 도움을 줄 것으로 예상된다. 하지만 OPRoS에서는 컴포넌트 사이의 연결이 직접적인 연결 방식인 반면, ROS에서는 토픽이라는 중간매개체를 통해 간접적으로 연결하는 방식으로, 서로 다른 결합 방식을 사용하고 있어서 OPRoS와 ROS 상호간에 결합하는데 어려움이 있었다[10,11]. 이를 해결하기 위해 본 논문에서는 ROS와 OPRoS 컴포넌트가 하나의 로봇 서비스를 구성할 때 동일한 개발 방식을 통해 함께 결합이 가능하도록 OPRoS 컴포넌트 프레임워크를 확장한 내용을 설명한다.

본 논문의 구성은 다음과 같다. II 장, III 장에서는 각각 OPRoS와 ROS의 컴포넌트 모델을 설명하고, IV 장에서는 ROS와 연동하기 위한 OPRoS의 확장된 내용을 상세히 살피고, V 장에서는 확장된 OPRoS와 ROS의 연동 시험 결과를 보이고, VI 장에서 결론을 맺는다.

## II. OPRoS 컴포넌트 모델

OPRoS 컴포넌트들은 포트 메커니즘을 통해 상호간의 메쏘드 호출이나 데이터 및 이벤트 교환이 이루어진다. 컴포넌트는 외부와의 인터페이스를 위해 한 개 이상의 포트를 가질 있는데, 그림 1에서와 같이 OPRoS 컴포넌트 모델에서는 서비스 포트, 데이터 포트, 이벤트 포트와 같이 3가지 종류의 포트를 규정하고 있다[12]. 서비스 포트를 통해서는 컴포넌트가 제공하는 메쏘드를 호출하거나 컴포넌트 내부의 멤버 변수를 접근할 수 있다. 데이터 포트는 컴포넌트 사이에 데이터 교환을 위해 사용된다. 컴포넌트의 출력 데이터 포트는 다른 컴포넌트의 입력 데이터 포트에 사전에 정해진 타입의 데이터를 전송할 수 있다. 전송받은 데이터는 입력 데이터 포트의 버퍼에 저장되어 컴포넌트 주기 마다 처리된다. 이벤트 포트는 이벤트 데이터를 전달하는데 사용된다. 이벤트 포트가 정해진 타입의 데이터를 전달하는 측면에서는 데이터 포트와 유사하다. 하지만, 데이터 포트에서는 전달 받은 데이터를 버퍼에 저장하고 수신한 컴포넌트 주기에 처리되는 반면, 이벤트 포트에서는 이벤트 데이터를 수신한 즉시 처리되는 면에서 다르게 동작한다. 즉, 이벤트 포트를 처리하는 방

* Corresponding Author
Manuscript received July 7, 2014 / revised September 10, 2014 / accepted September 15, 2014
장철수, 송병열, 김성훈: 한국전자통신연구원 지능형인지기술연구부
(jangcs@etri.re.kr/sby@etri.re.kr/saint@etri.re.kr)
※ 본 연구는 산업통상자원부에서 시행한 지식경제 기술혁신사업이 지원하여 연구되었음(과제번호: 10044006).
※ 이 논문은 2014 제 29회 ICROS 학술대회에 초안이 발표되었음.

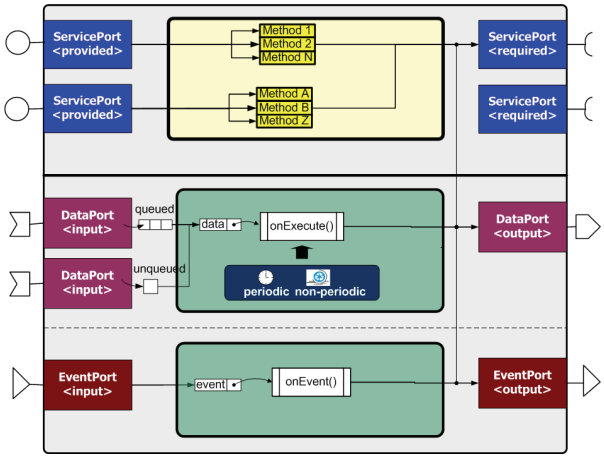



그림 1. OPRoS 컴포넌트 모델.
Fig. 1. OPRoS component model.

법은 수신 컴포넌트의 자체 주기에 따라 처리되는 것이 아니라 외부에서 이벤트 데이터가 도착한 경우에 반응해서 동작하는 것이므로 송신 컴포넌트의 주기에 따라 처리된다.

OPRoS 컴포넌트 모델을 따라 손쉬운 컴포넌트 개발을 지원하는 컴포넌트 개발도구는 저작도구와 조합도구로 구성되는데, 저작도구는 단위 컴포넌트를 손쉽게 저작할 수 있게 해주며, 조합도구는 단위 컴포넌트를 묶어 로봇 응용 패키지를 구성할 수 있도록 해준다. OPRoS 저작도구를 이용하여 컴포넌트를 개발할 때 포트 정보를 포함하여 컴포넌트를 구성하는 다양한 인터페이스 정보는 컴포넌트 프로파일이라는 XML 파일에 기술되며, 프로파일에 따라 컴포넌트의 C++ 탬플릿 파일들이 자동으로 생성되고 사용자 코드를 손쉽게 추가하여 단위 컴포넌트를 만들 수 있도록 해준다. 저작도구에서 완성된 단위 컴포넌트는 컴포넌트 프로파일과 함께 OPRoS 컴포넌트 조합 도구에 전달되고, 로봇 응용 패키지를 만드는데 사용된다. 로봇 응용 개발자는 조합도구를 이용하여 컴포넌트 목록에 나타난 단위 컴포넌트들을 조합도구의 패키지 구성 화면에 끌어다 갖다 놓고 그림 2와 같이 컴포넌트 사이의 포트를 연결하여 로봇 응용 패키지를 만든다.

포트 사이의 연결 정보는 패키지 프로파일에 저장되는데, 패키지 프로파일은 분산 노드들의 네트워크 설정 정보, 패키지에 참여하고 있는 컴포넌트 목록, 컴포넌트 사이의 포트 연결 정보 등을 포함하고 있다. OPRoS 프레임워크는 이들 프로파일 정보를 해석하고, 컴포넌트 간 연결 정보를 이용하여 응용 패키지에 참여하는 컴포넌트들의 통신을 연결하고, 응용 패키지를 구동한다. 이와 같이 OPRoS 컴포넌트들은 연결되는 상대 컴포넌트를 서로 알고 포트를 통해서 직접적인 통신 연결을 맺는다. 이런 결합 방식은 컴포넌트 사이에 직접적인 통신이 가능하게 하여 효율적인 통신 방법을 제공할 수 있다.

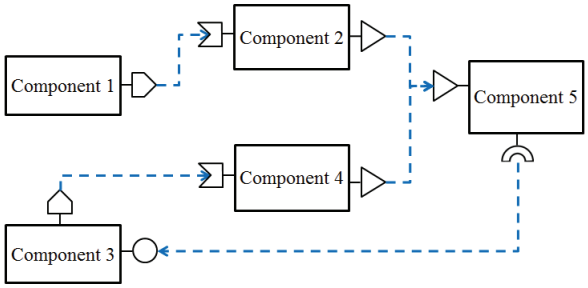



그림 2. OPRoS 컴포넌트 연결 예제.
Fig. 2. An example of OPRoS component connections.

## III. ROS 통신 모델

ROS는 그래프 기반 아키텍처를 채택하고 있는데, 그래프는 노드(Node)들의 연결로 구성된다[8]. 노드는 연산을 수행하는 프로세스로서, 하나의 로봇 시스템은 일반적으로 많은 수의 노드로 구성된다. 그림 3과 같이 ROS에서의 각 노드는 출판/구독(Publish/Subscribe) 방식의 토픽(Topic)이라는 중간 매개체를 이용하여 통신을 수행한다. 즉, 다른 컴포넌트로부터 데이터를 전달받고자 하는 컴포넌트는 통신 채널 역할을 수행하는 관심 토픽에 구독해 놓고, 송신할 데이터가 있는 컴포넌트는 해당 토픽에 데이터를 출판하면, 토픽에 구독되어 있는 컴포넌트에게 데이터가 전달돼서 처리하도록 하는 방식으로 센서 정보나 제어 명령, 상태 정보 등의 메시지를 노드 간에 주고 받을 수 있다. 이와 같이 ROS에서는 서로 데이터를 주고 받는 노드끼리 직접 연결되지 않고 토픽을 통해 간접적으로 연결되는 방식을 사용하고 있다.

ROS에서 토픽을 이용한 연결 방식은 어느 컴포넌트로부터 데이터가 왔는지 혹은 어느 컴포넌트로 데이터가 가는지 신경 쓸 필요가 없어서 OPRoS의 연결 방식 보다 유연한 조합이 가능하다. 즉, 통신의 효율성 측면에서는 OPRoS의 연결 방식이 유리한 면이 있으며, 컴포넌트들의 유연한 조합 측면에서는 ROS의 연결 방식이 유리한 면이 있기 때문에 응용에 특성에 따라 두가지 결합 방법 중 적절한 방법이 선택되어 이용될 수 있어야 한다. 하지만, OPRoS와 ROS는 각자의 연결 방식 만을 지원하므로 각자의 방식으로 만들어진 컴포넌트는 서로 동시에 사용할 수 없는 문제점이 있었다.

서로 다른 로봇 프레임워크를 연동하기 위한 방법으로 기존에는 브릿지 컴포넌트를 만드는 방법에 대한 연구가 수행되었다[13,14]. 즉, ROS 노드와 OPRoS 컴포넌트를 연결하고자 하는 경우에 중간에 ROS 메시지를 OPRoS용 메시지를 변환하는 브릿지 컴포넌트를 만들어 둘 사이를 연결하도록 했다. 하지만, 브릿지 컴포넌트를 둠으로써 추가적인 브릿지 컴포넌트를 구동시켜야 하는 부담이 있으며 전체 응용 패키지 구성도 복잡해지는 문제도 발생한다. 또한, 서로 다른 프레임워크에서 만들어진 기존 컴포넌트들간에는 인터페이스가 상이하여 서로 연결하여 사용하려는 경우는 드문 대신, 다른 프레임워크의 기존 컴포넌트를 활용하기 위해 새로운 컴포

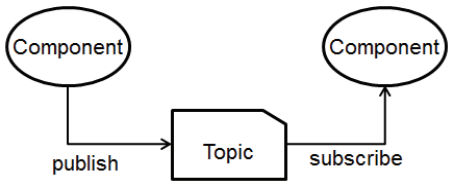



그림 3. ROS 출판/구독 모델.
Fig. 3. ROS publish/subscribe model.

넌트를 개발하고 기존 컴포넌트와 연결하려는 경우가 더 많기 때문에, 기존에 존재하던 컴포넌트간의 연결을 위한 브릿지 방법 대신 새로운 컴포넌트를 만들 때 다른 프레임워크의 기존 컴포넌트와 바로 연결할 수 있는 개발 방법이 요구된다.

## IV. ROS 연동을 위한 OPRoS 확장

본 논문에서는 컴포넌트를 개발할 때 컴포넌트에 추가될 이벤트 포트가 기존 OPRoS의 연결 방식을 사용할지 혹은 ROS와 같은 토픽과 연결할 지 지정할 수 있도록 하여, 기존 OPRoS 컴포넌트 및 ROS 토픽과 연결을 할 수 있도록 OPRoS 컴포넌트의 이벤트 포트를 확장하였을 뿐 만 아니라, OPRoS나 ROS의 통신 방식에 상관 없이 사용자는 컴포넌트 개발시 동일한 형태의 코딩으로 이벤트 데이터를 송수신 할 수 있도록 OPRoS 프레임워크를 다음과 같이 확장하였다.

### 1. 컴포넌트 프로파일 확장

OPRoS 컴포넌트 저작도구는 컴포넌트 프로파일에 따라 컴포넌트 코드를 생성하는데, 확장된 OPRoS 플랫폼의 컴포넌트 저작도구는 컴포넌트의 이벤트 포트가 ROS와 연동하기 위한 포트인지 OPRoS 전용의 포트인지를 개발자가 지정할 수 있도록 하였으며, ROS와 연동하기 위한 포트를 위해 OPRoS 컴포넌트 프로파일의 <event_port> 항목에 이벤트 포트의 타입 태그 및 연결하는 토픽 이름을 추가할 수 있도록 하였다. 즉, 그림 4의 예제에서처럼 OPRoS 전용의 이벤트 포트인 첫번째 및 두번째와는 달리, ROS와 연동되는 세번째와 네번째 이벤트 포트는 기존 프로파일에 topic::ros 라는 타입 태그를 추가로 가질 수 있으며, 해당 타입 태그가 있는 경우에는 연결되는 <topic_name> 항목이 추가된다.

저작도구는 컴포넌트 프로파일에 타입 태그 항목이 없는 이벤트 포트일 때, 출력 이벤트 포트의 경우에는 TCP/IP 기반의 클라이언트 기능을 수행하는 코드를, 입력 이벤트 포트의 경우에는 서버 코드를 포함하는 C++ 컴포넌트 코드를 생성한다. 반면에, 토픽을 이용하도록 설정된 경우에는 출력 이벤트 포트일 때는 토픽에 출판하는 코드를, 입력 이벤트 포트일 때는 주어진 토픽에 구독하는 코드를 ROS API를 이용하여 생성하여 놓는다. 이렇게 ROS 토픽을 이용하는 OPRoS 컴포넌트는 ROS 노드가 되어 다른 ROS 노드와 통신이 가능하게 된다.

```
<ports>
  <event_port>   <!-- opros output event port -->
      <name>SimpleEventOut</name>
      <data_type>int</data_type>
      <usage>output</usage>
  </event_port>
  <event_port>   <!-- opros input event port -->
      <name>SimpleEventIn</name>
      <data_type>int</data_type>
      <usage>output</usage>
  </event_port>
  <event_port type="topic::ros"> <!--ros output event port -->
      <name>RosEventOut</name>
      <topic_name>myRosTopic</topic_name>
      <data_type>int</data_type>
      <usage>output</usage>
  </event_port>
  <event_port type="topic::ros"> <!-- ros input event port -->
      <name>RosEventIn</name>
      <topic_name>yourRosTopic</topic_name>
      <data_type>MyStructType</data_type>
      <usage>input</usage>
  </event_port>
</ports>
```

그림 4. 컴포넌트 프로파일 확장.

Fig. 4. Extension of component profile.

### 2. 동일 인터페이스를 이용한 이벤트 송신 방법

확장된 OPRoS 컴포넌트 개발도구는 지정된 설정에 따라 탬플릿 클래스 형태로 이벤트 포트를 생성하도록 하여, 동일한 인터페이스를 통해 이벤트를 보낼 수 있도록 하였다. 즉, 그림 5와 같이 컴포넌트 개발자는 생성된 이벤트 포트의 탬플릿 클래스 내의 동일한 멤버 메쏘드인 push() 함수를 이용하여 이벤트를 보내도록 하였다.

이를 위해 OPRoS 프레임워크는 그림 6 및 그림 7과 같이 OutputEventPortX라는 탬플릿 클래스를 정의해 놓고 있으며 이를 구체화한 OutputEvnetPortX<ROS>는 ROS의 토픽에 이벤트를 출판하는 메커니즘을, OutputEventPortX<OPRoS>는 OPRoS 전송 메커니즘을 각각 따르도록 하였다.

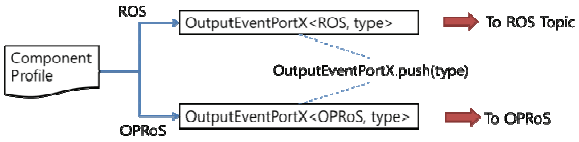



그림 5. 동일 인터페이스를 이용한 이벤트 송신.

Fig. 5. Event transferring through same interfaces.

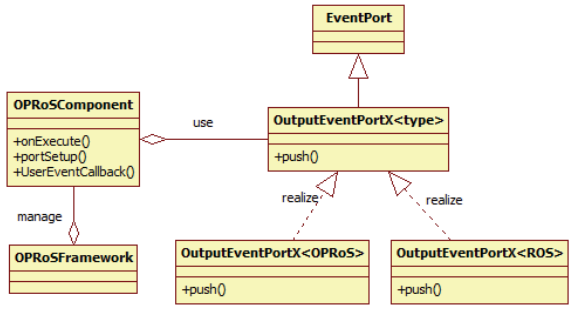



그림 6. 이벤트 송신 클래스다이어그램.

Fig. 6. Class diagram for event transferring through same interfaces.

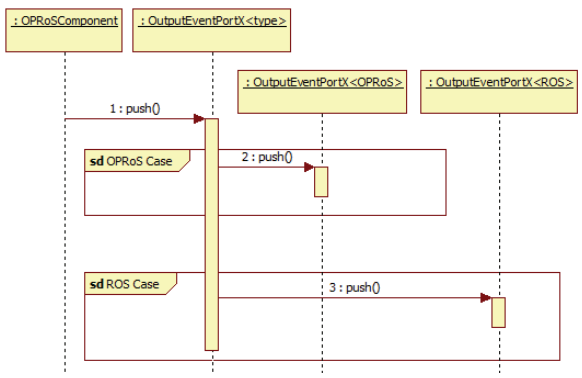



그림 7. 이벤트 송신 시퀀스다이어그램.

Fig. 7. Sequence diagram for event transferring.

### 3. 확장된 이벤트 수신 방법

기존 OPRoS에서는 이벤트를 전달할 때, 송신하는 쪽에서 이벤트 데이터 객체를 생성하고 해당 객체에 이벤트 ID를 직접 추가하여 송신하고, 수신하는 쪽에서는 컴포넌트에 하나만 존재하는 onEvent() 콜백 함수를 통해 모든 이벤트 데이터 객체를 수신한 후 해당 객체로부터 이벤트 ID를 추출하고 비교한 후 해당 데이터를 처리해야 했다. 즉, 개발자가 직접 이벤트 ID를 생성 및 설정하고, 하나의 onEvent() 콜백함수 내에서 개발자가 직접 이벤트 ID를 해석한 후 해당 이벤트 데이터를 처리하여야 하므로 사용에 어려움이 많았다. 반면에, ROS에서는 이벤트 수신시 해당 이벤트를 처리하도록 등록된 콜백함수로 이벤트가 전달되어 해당 콜백함수에서 이벤트를 처리한다. ROS의 콜백함수를 이용한 이벤트 처리는 이벤트를 처리하는 콜백함수에서 해당 이벤트만 집중해서 처리하도록 코딩할 수 있어서 코드가 간결하고 이해하기가 쉬운 장점이 있다.

따라서 보다 쉽게 이벤트 포트를 사용할 수 있도록 그림 8과 같이 OPRoS 컴포넌트에서도 ROS처럼 이벤트를 처리할 수 있는 전용 콜백함수를 등록할 수 있도록 하여, ROS와의 이벤트 교환뿐 만 아니라 OPRoS 전용의 이벤트 교환에서도 동일한 코딩 방식으로 컴포넌트를 개발할 수 있도록 하였다. 즉, 콜백 함수가 ROS 토픽에 연결되어 사용되는 경우에는, 토픽에 구독한 콜백 함수는 ROS 플랫폼에서 직접 호출되며, OPRoS 전용 이벤트 포트를 통해 이벤트 데이터를 수신한 경우, 해당 포트에 등록된 콜백 함수로 이벤트 데이터를 전달하여 처리하도록 하였다.

이를 위해 그림 9 및 그림 10과 같이 OPRoS 전용 이벤트 포트의 경우에는 이벤트 포트 내부에 EventCallback 이라는 함수 포인터를 두어 컴포넌트 포트 초기화 과정(portSetup())에서 컴포넌트 내부의 콜백을 이벤트 포트에 등록하도록 하

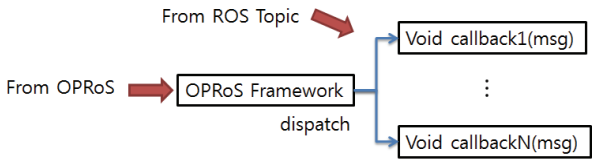



그림 8. 콜백 함수를 통한 이벤트 수신 방법.

Fig. 8. Event receiving through callbacks.

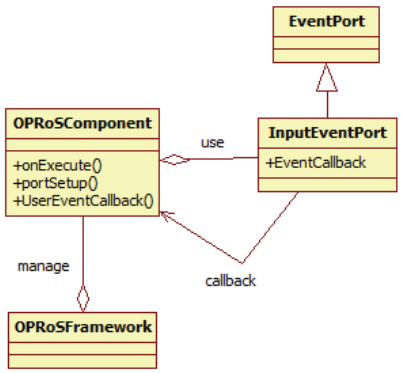



그림 9. 이벤트 수신 클래스다이어그램.

Fig. 9. Class diagram for event receiving via callback.

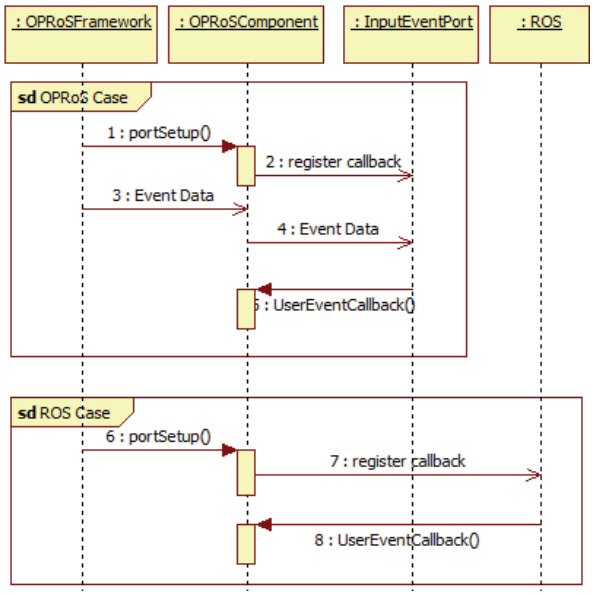



그림 10. 이벤트 수신 시퀀스 다이어그램.

Fig. 10. Sequence diagram for event receiving via callback.

였고, 해당 포트에 이벤트가 전달된 경우에는 참조하고 있는 콜백함수가 호출되도록 하였다. ROS 토픽과 연결되는 경우에는 컴포넌트 포트 초기화 과정에서 ROS 토픽에 콜백함수를 구독시켜 ROS 플랫폼이 직접 콜백을 호출하도록 하였다.

### 4. 콜백함수 탬플릿 생성 및 등록 규칙

컴포넌트 프로파일의 이벤트 포트에 대한 정보를 바탕으로 ROS 연동 컴포넌트 및 일반 OPRoS 컴포넌트에서 표 1과 같은 규칙에 따라 동일한 형태로 입력 이벤트 포트에 대응되는 콜백함수 탬플릿 코드를 생성시키도록 하였다.

또한 생성된 콜백함수를 OPRoS 프레임워크에게 알려주기 위해서, 표 2과 같이 콜백함수를 프레임워크에 등록하는 매퍼 매크로를 정의하고 코드 생성 단계에서 자동으로 추가하도록 하였다. 즉, ROS 수신 이벤트 포트의 경우에는 ROS 토픽에 콜백함수를 등록하는 매크로 코드를 호출하며, OPRoS 수신 이벤트 포트의 경우에는 포트 내의 콜백 참조 객체에 해당 콜백 함수를 등록하는 매크로 코드를 호출하도록 했다.

### 5. 컴포넌트와 토픽의 연결

조합도구는 컴포넌트 프로파일을 해석하여, ROS와 결합이

표 1. 콜백 함수 생성 규칙.

Table 1. Callback generation rule.

| | |
|---|---|
| 리턴값 | void |
| 함수명 | 이벤트 포트 이름+Callback |
| 파라미터 | 컴포넌트 프로파일에 기술된 데이터 타입 (기본 데이터 타입 및 구조체 타입) |

표 2. 콜백 함수 등록 규칙.

Table 2. Callback registration rule.

| | |
|---|---|
| OPRoS 연동 | SET_EVENT_CALLBACK (event_port_name, callbackFunction, this); |
| ROS 연동 | SET_TOPIC_CALLBACK (event_port_name, callbackFunction, this, topicName); |

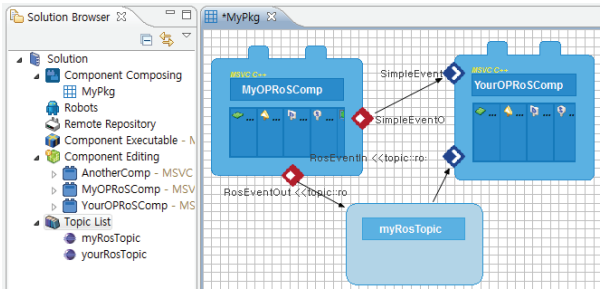



그림 11. OPRoS 컴포넌트와 ROS 토픽의 연결.

Fig. 11. Connection between OPRoS component and ROS topic.

가능하도록 만들어진 컴포넌트의 경우에는, 그림 11처럼 타입 태그에 포함된 토픽 정보를 조합도구의 토픽 목록(Topic List)에 등록하고 사용자가 조합에 사용할 수 있도록 화면에 표시하여 컴포넌트와 토픽이 조합에 사용될 수 있도록 하였다. 특히 ROS 토픽을 사용하는 이벤트 포트의 경우에는 그림에서와 같이 컴포넌트를 패키지 구성 화면에 갖다 놓았을 때 <<topic::ros>>라는 스테레오타입을 표시하여 ROS 토픽과 연결되는 이벤트 포트라는 것을 명확히 표시하고 해당 연결에만 사용될 수 있도록 하였다.

### 6. 패키지 프로파일 확장

포트 사이나 포트와 토픽 사이의 연결 정보는 패키지 프로파일에 저장되는데, 토픽과 컴포넌트 포트의 연결 정보를 나타내기 위해 패키지 프로파일 내의 <port_connection> 항목에 타입 확장 태그를 추가하였으며, 컴포넌트와 토픽이 출판의 방향으로 연결된 경우에는 <source> 항목을 컴포넌트로 명시하고 <target> 항목을 토픽으로 명시하며, 구독의 경우에는 반대로 하여 연결 정보를 패키지 프로파일에 추가하도록 하였다. 그림 12의 예제를 보면, 첫번째 포트 연결은 일반 OPRoS 컴포넌트 사이의 이벤트 포트 연결을 나타내고 있는데 반해, 두번째와 세번째 포트 연결의 경우에는 port_type이 “topic”으로 되어 있어서 토픽과의 연결을 나타내고 있으며, 각각 OPRoS 컴포넌트에서 ROS 토픽으로 이벤트 데이터를 송신하는 연결과, ROS 토픽으로부터 OPRoS 컴포넌트로 수신하는 연결을 나타내고 있다.

```
<package_profile >
    ...
    <port_connections>
      <!-- general opros components -->
      <port_connection port_type=”event”>
          <source component_name="MyOPRoSComp”
            port_name=”SimpleEventOut”/>
          <target component_name="YourOPRoSComp”
            port_name=”SimpleEventIn”/>
      </port_connection>

      <!-- opros to ros -->
      <port_connection port_type="topic">
          <source component_name="MyOPRoSComp”
            port_name="RosEventOut” />
          <target topic_name="myRosTopic”/>
      </port_connection>

      <!-- ros to opros -->
      <port_connection port_type="topic">
          < source topic_name="myRosTopic” />
          <target component_name="YourOPRoSComp"
            port_name= “RosEventIn"/>
      </port_connection>
      ...
    </port_connections>
</package_profile>
```

그림 12. 패키지 프로파일 확장.

Fig. 12. Extension of package profile.

## V. 실험

ROS 배포 패키지에 포함되어 있는 예제인 거북이 시뮬레이터를 활용하여 ROS 토픽(turtle1/command_velocity)과 OPRoS 컴포넌트의 연동을 시험하였다.

이를 위해 확장된 OPRoS 프레임워크를 통해 ROS 토픽과 연동이 가능한 ROSTurtlePublisher와 ROSTurtleSubscriber OPRoS 컴포넌트를 제작하였다. 그림 13의 프로파일에서처럼 ROSTurtlePublisher는 토픽에 제어명령을 출판하고 ROSTurtleSubscriber 컴포넌트는 토픽에 구독하여 제어정보를 모니터링하는 할 수 있도록 컴포넌트 프로파일을 구성하였다. ROSTurtleSubscriber에 대한 콜백은 그림의 소스코드에서처럼 ROSTurtleSubscriber의 멤버메쏘드로 자동으로 생성된다.

저작도구를 통해 만들어진 컴포넌트들을 조합하기 위해 ROSTurtlePublisher 컴포넌트의 출력 이벤트 포트와 ROS 토픽을 연결하여 컴포넌트에서 ROS 토픽으로 출판하기 위한

**ROSTurtlePublisher.xml**

```
<ports>
  …
  <event_port type="topic::ros">
      <name>VelocityOut</name>
      <topic_name>turtle1/command_velocity</topic_name>
      <data_type>turtlesim::Velocity</data_type>
      <usage>output</usage>
  </event_port>
  …
</ports>
```

**ROSTurtleSubscriber.xml**

```
<ports>
  …
  <event_port type="topic::ros">
      <name>VelocityIn</name>
      <topic_name>turtle1/command_velocity</topic_name>
      <data_type>turtlesim::Velocity</data_type>
      <usage>input</usage>
  </event_port>
  …
</ports>
```

**ROSTurtleSubscriber.cpp**

```
void ROSTurtleSubscriber::VelocityInCallback
(turtlesim::Velocity arg)
{
  // user code
}
```

그림 13. 실험에 사용된 컴포넌트 프로파일 및 소스코드.

Fig. 13. Component profiles and codes used in an experiment.

**OPRoSROSTurtle.xml**

```
<package_profile >
    ...
    <port_connections>
      <port_connection port_type="topic">
          <source component_name="ROSTurtlePublisher"
            port_name= "VelocityOut" />
          <target topic_name="turtle1/command_velocity"/>
      </port_connection>
      <port_connection port_type="topic">
        < sourcetopic_name="turtle1/command_velocity"/>
          <target component_name="ROSTurtleSubscriber"
            port_name= "RosEventIn"/>
      </port_connection>
    </port_connections>
    ...
</package_profile>
```

그림 14. 실험에 사용된 패키지 프로파일.

Fig. 14. Package profiles used in an experiment.

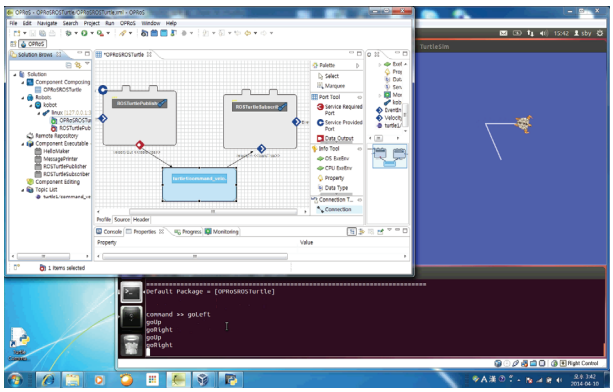


그림 15. ROS와 OPRoS 컴포넌트 연동 실험.

Fig. 15. An experiment of interoperating of OPRoS and ROS.

연결을 짓고, ROS 토픽에서 ROSTurtleSubscriber 컴포넌트의 입력 이벤트 포트 방향으로 구독 연결을 지어 그림 14와 같은 패키지 프로파일이 작성되도록 하였다.

완성된 패키지를 확장된 OPRoS 프레임워크에서 구동하여 그림 15에서처럼 컴포넌트들이 ROS 토픽과 유기적으로 연동되어 거북이 시뮬레이터를 정상적으로 제어하고 모니터링하는 것을 확인할 수 있었다. 보다 자세한 연동 결과는 동영상 공유 사이트(http://youtu.be/m0SVOTP2dHg)에서 확인 가능하다.

## VI. 결론

본 논문에서는 서로 다른 연결방식을 사용하는 OPRoS와 ROS의 컴포넌트들을 함께 사용하여 로봇 서비스를 만들 수 있도록, 컴포넌트 및 패키지 프로파일, 코드 생성 규칙, 개발 도구를 포함한 OPRoS 프레임워크를 확장한 내용을 설명하였다. 확장된 OPRoS 프레임워크를 통해 OPRoS 컴포넌트는 ROS 토픽과 연결할 수 있게 되었을 뿐 만 아니라, 컴포넌트 개발시 동일한 형태의 코딩으로 OPRoS와 ROS의 이벤트 데이터를 송수신 할 수 있도록 개발의 편의성을 제공할 수 있게 되었다.

## REFERENCES


[1] J. Jackson, "Microsoft robotics studio: a technical introduction," *Robotics & Automation Magazine*, vol. 21, pp. 82-87, 2007.

[2] 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, pp. 55-60, 2006.

[3] H. Utz, S. Sablatnog, S. Enderle, and G. Kraetzschmar, "Miro-middleware for mobile robot application," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 4, pp. 493-497, 2002.

[4] N. Ando, T. Suehiro, K. Kitagaki, T. Kotoku, and W. K. Yoon, "RTMiddleware: Distributed component middleware for RT (robot technology)," *IEEE/RSJ International Conference on Robots and Intelligent Systems*, pp. 3555-3560, 2005.

[5] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the Orocos project," *Proc. of the IEEE International Conference on Robotics & Automation*, pp. 2766-2771, 2003.

[6] C. Jang, S.-I. Lee, S.-W. Jung, B. Song, R. Kim, S. Kim, and C.-H. Lee, "OPRoS: A new component-based robot software platform," *ETRI Journal*, vol. 32, no. 5, pp. 646-656, 2010.

[7] S.-H. Kim and H.-S. Park, "Design of a robot-in-the-loop simulation based on OPRoS," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 19, no. 3, pp. 248-255, 2013.

[8] ROS, Available: http://www.ros.org

[9] B. Song, C. Jang, Y. Jung, S. Kim, and H. Choi, "Trends of the interoperating technology for robot software platform," *Korea Robotic Society Magazine (in Korean)*, vol. 10, no. 1, pp. 16-22, 2013.

[10] C. Jang, B. Song, S. Jung, and S. Kim, "A heterogeneous coupling scheme of OPRoS component framework with ROS," *9th International Conference on Ubiquitous Robots and Ambient Intelligence*, pp. 298-301, 2012.

[11] C. Jang, B. Song, and S. Kim, "An Extension of OPRoS framework for interoperating with ROS," *Proc. of the 29th ICROS Annual Conference (in Korean)*, pp. 33-34, 2014.

[12] B. Song, S. Jung, C. Jang, and S. Kim, "An introduction to robot component model for OPRoS," *International Conference on Simulation, Modeling and Programming for Autonomous Robots Workshop*, pp. 592-603, 2008.

[13] OpenRTM-ROS Integration for Humanoid/Mobile Manipulator Robotics, Availavle: http://code.google.com/p/rtm-ros-robotics/

[14] A. Salov, H.-S. Park, S. Han, and D. Lee, "An effective method of sharing heterogeneous components of OPRoS and RTM," *Journal of Electrical Engineering & Technology*, vol. 9, no. 2, pp. 755-761, 2013.

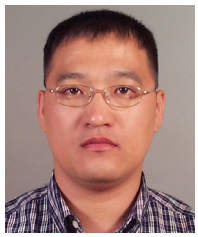


**장 철 수**

1995년 인하대학교 전자계산공학과(공학사). 1997년 광주과학기술원 정보통신공학과(공학석사). 2011년 충남대학교 컴퓨터공학과(공학박사). 1997년~현재 한국전자통신연구원 책임연구원. 관심분야는 지능형 로봇, 분산 시스템, 실시간 시스템.

**송 병 열**

1995년 전북대학교 전자공학과(공학사). 1997년 전북대학교 전자공학과(공학석사). 2013년 충남대학교 컴퓨터공학과(박사수료). 1997년~현재 한국전자통신연구원 책임연구원. 관심분야는 지능형 로봇, 임베디드 시스템, 소프트웨어 공학.



**김 성 훈**

1995년 광운대학교 전자공학과(공학사). 1997년 광운대학교 전자공학과(공학석사). 2007년 한양대학교 전자컴퓨터공학과(박사수료). 1997년~현재 한국전자통신연구원 책임연구원. 관심분야는 로봇 지능, 로봇 소프트웨어 공학, 국방 로봇.