WS-ECA 프레임워크 프로토콜

The Protocol on WS-ECA Framework

이 원 석* 박 종 헌** 이 규 철***
Won-Suk Lee Jong-Hun Park Kyu-Chul Lee

요 약

정보통신 환경이 유비쿼터스 환경으로 발전하면서 이종의 디바이스 간의 연동 기술의 필요성이 크게 증가하고 있으며, 이를 해결하기 위한 하나의 대안으로 웹서비스가 주목 받고 있다. 최근 디바이스 연동을 위한 선언적인 ECA(Event-Condition-Action) 룰 기반의 프레임워크가 제안 되었으며, 이를 WS-ECA(Web Services-ECA) 프레임워크라 한다. 이는 웹서비스를 기반으로 디바이스 간의 연동을 선언적으로 정의 하는 ECA 기반 룰 언어를 제안하고, 정의된 룰을 기반으로 디바이스 간의 연동을 수행할 수 있는 프레임워크를 제안하였다. 본 논문에서는 본 프레임워크가 유비쿼터스 환경의 동적인 특성을 효과적으로 지원하기 위해 반드시 필요한 컴포넌트 간의 프로토콜 및 프레임워크 내의 서비스/이벤트 기술을 위한 WSDL 확장 방법을 제안한다.

Abstract

Importance of coordination among heterogeneous devices is significantly increasing to meet the challenges presented by the evolvement of the ubiquitous computing environment. The web services technology is going up as a promising solution to support the inter-operability between such heterogeneous devices via well-defined protocol. Currently, ECA(Event-Condition-Action) rule based framework for coordinating devices is proposed, it is called as WS-ECA(Web Services-ECA) framework. This framework was basically based on the web services and proposed the ECA based rule language. And this also proposed the framework that could enable the device coordination based on proposed rule language. In this paper, we propose the protocol among the components of WS-ECA framework to support dynamic feature for ubiquitous environment. and we also propose WSDL extension to support service and event descriptions these are provided in components of this framework.

☞ 키워드: WS-ECA Framework, Device Coordination, ECA Rule, Web Services, WS-ECA 프레임워크, 디바이스 연동, ECA 룰, 웹서비스

1. 소 개

최근의 컴퓨팅 환경은 수십억 개의 디바이스가 긴밀하게 네트워크로 연결되는 유비쿼터스 환경으로 빠르게 발전 하고 있으며[1], 이러한 환경은 다양한 응용, 프로토콜, 포맷 등이 상존하는 이종의 시스템 환경을 포함한다[2]. 이와 같은 유비쿼터스 컴퓨팅 환경에서 웹서비스 기술은 이종의 하드웨어와 소프트웨어 플랫폼을 가지고 있는 다양한 디바이스들이 상호운용성을 보장하면서 자연스럽게 연동할 수 있는 효율적인 수단을 제공할 수 있다[3].

이와 같이 웹서비스를 기반으로 디바이스 간의 연동을 수행하기 위한 주요 연구들을 보면, MS의 유비쿼터스 컴퓨팅 프로젝트의 일환으로 초소형 디바이스에 웹서비스 기술을 내장하기 위한 Invisible computing[4], Trento 대학의 웹서비스를 기반으로 모든 디바이스(센서, 가전 등) 간의 연동 방법을 연구하는 스마트 홈 프로젝트[5]와 같은 연구가 있다. 또한, 홈네트워킹의 미들웨어 표

* 정 회 원 : 한국전자통신연구원 표준연구센터 연구원
wslee@etri.re.kr
** 정 회 원 : 서울대학교 산업공학과 교수
jonghun@snu.ac.kr
*** 정 회 원 : 충남대학교 컴퓨터공학과 교수
kclee@cnu.ac.kr(교신저자)
[2007/06/07 투고 - 2007/06/12 심사 - 2007/07/24 심사완료]

준을 개발하고 있는 UPnP 포럼[6]에서는 UPnP 2.0에 웹서비스 기반의 디바이스를 연동하는 기술에 대한 내용을 준비하고 있다.

웹서비스를 기반으로 보다 쉽고, 정확하게 디바이스 연동을 수행하기 위해서는 비즈니스 분야의 WS-BPEL[7]과 WS-CDL[8]과 같은 선언적인 언어가 필요하다. 이러한 언어는 웹서비스를 기반으로 외부의 다양한 사업 파트너들과 내부의 시스템간의 효율적인 통합을 가능하게 한다. 그러나 기존에 정의된 WS-BPEL과 WS-CDL은 기본적으로 유비쿼터스 환경의 동적인 특성을 고려하지 않았고, 언어자체가 비즈니스 분야의 요구사항에 맞도록 방대한 기능을 정의하고 있어 유비쿼터스 환경의 소형 디바이스에 적용하는 것은 적합하지 않다.

WS-ECA 프레임워크[9, 10]는 유비쿼터스 환경의 동적인 특성을 지원하고, 소형 디바이스에 적용 가능한 XML 기반의 디바이스 연동 기술 언어 및 이를 실행하기 위한 프레임워크를 제안하였다. 제안된 기술 언어는 데이터베이스에 이벤트 기반의 동적 처리 메커니즘을 제공하기 위해 고안된 ECA(Event-Condition-Action) 룰을 기반으로 한다. ECA 개념은 현재 분산된 환경에서 가장 효율적인 연동 방법 중의 하나로 인식되어 다양한 분야에서 활용되고 있으며[11, 12], 유비쿼터스 환경에서 특정 이벤트에 대한 대응 로직을 명확히 기술할 수 있는 장점이 있다[13].

WS-ECA 프레임워크는 웹서비스를 기반으로 유비쿼터스 환경에 적합한 동적인 디바이스 연동 환경을 제안하고 있다. 그러나 이러한 동적인 디바이스 연동 환경을 실현할 때 반드시 필요한 프레임워크 구성 컴포넌트들 간의 구체적인 프로토콜이 제안되어 있지 않다. 또한, 사용자가 선언적으로 디바이스의 연동을 정의하기 위해서는 현재 프레임워크 환경에서 제공되는 서비스 및 이벤트 정보가 필요하다. 서비스 정보에 대한 기술은 WSDL(Web Service Description Language)을 이용해서 표현할 수 있으나, 아직까지 WS-Eventing 표준을 위한 이벤트 기술 언어는 없다. 따라서 본 논문에서는 WS-ECA 프레임워크의 동적인 특성을 고려한 프로토콜 및 프레임워크 환경에서 제공되는 서비스 및 이벤트 정보를 모두 기술할 수 있는 WSDL 확장 방법을 제안한다.

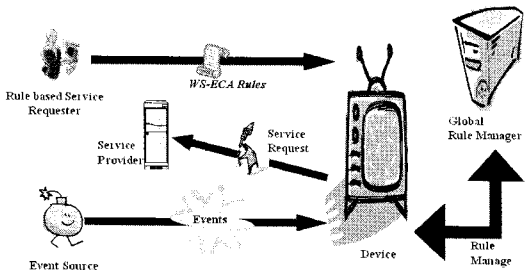
본 논문의 구성은 다음과 같다. 2장에서는 WS-ECA 프레임워크에 대한 소개를 하고, 3장에서는 서비스 및 이벤트 정보 기술을 위한 WSDL 확장 방법을 설명한다. 4장에서는 WS-ECA 프레임워크를 위한 구체적인 프로토콜을 설명하고, 5장에서는 관련연구를 소개한다. 마지막으로 6장에서는 결론을 맺고 향후 연구 방향을 설명한다.

2. WS-ECA 프레임워크

2.1 개요

기본적으로 WS-ECA 프레임워크는 WS-Eventing의 publish/subscribe 메커니즘과 웹서비스 기반의 request/response 메커니즘을 사용하여 기술된 명시적인 룰을 기반으로 디바이스 간의 연동을 가능하게 한다. (그림 1)은 WS-ECA 프레임워크의 전체 구성을 보여준다. 프레임워크는 기본적으로 룰 기반 서비스 요청자(Rule based Service Requester), 디바이스, 글로벌 룰 매니저(Global rule manager), 이벤트 소스(Event source), 서비스 제공자로 구성된다. 룰 기반 서비스 요청자는 자신이 원하는 서비스에 대한 디바이스 간의 연동을 WS-ECA 룰을 이용하여 정의하고 이를 디바이스에 저장한다. 디바이스는 룰을 저장하라는 요청이 오면, 기존에 저장되어 있는 룰과의 충돌 문제가 없는지 글로벌 룰 매니저(Global rule manager)에게 확인 후 문제가 없으면 저장한다. 디바이스에 저장된 WS-ECA 룰은 이벤트를 기반으로 하기 때문에 특정 이벤트가 발생하기를 기다린다. 디바이스가 이벤트 소스가 보낸 이벤트를 받게 되면 수행할 대상이 되는 룰을 찾아 조건을 확인한 후 정의된 액션을 수행하게 된다. 이때,

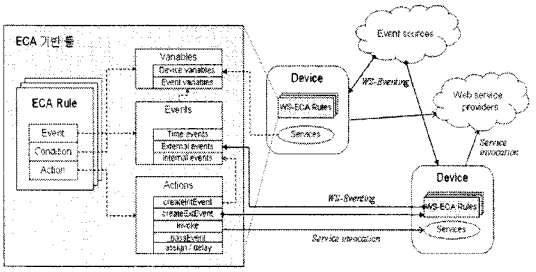
액션은 다른 룰을 실행시키기 위한 이벤트 생성이 될 수도 있고, 웹서비스 제공자가 제공하는 서비스를 호출할 수도 있다. 본 프레임워크에서는 모든 디바이스가 웹서비스 제공자가 될 수 있으며, 인터넷에 존재하는 웹서비스들도 연동의 대상이 된다. 또한, 이벤트 소스도 모든 디바이스가 될 수 있으며, WS-Eventing 표준을 따르는 인터넷의 이벤트 소스도 연동의 대상이 된다.



(그림 1) WS-ECA 프레임워크

2.2 WS-ECA 룰 기술 언어

(그림 2)는 WS-ECA 룰의 구조와 이를 기반으로 연동되는 디바이스, 이벤트 소스, 서비스 제공자의 전체적인 관계를 보여준다. WS-ECA 룰 구조는 이밴트, 조건, 액션으로 구성되며, 여기서 이벤트는 내부 이벤트(internal event), 외부 이벤트(external event), 시간 이벤트(time event) 중 하나의 알림 메시지(notification message)가 된다. 내부 이벤트는 같은 디바이스 내에서 다른 기능을 호출 할 때 사용된다. 즉, 일반적으로 함수를 모듈별로 구성하여 필요시 호출하는 것과 같이, WS-ECA 룰을 정의 할 때도 기능을 이벤트 기반으로 모듈화 한 후 내부 이벤트를 통해서 필요한 기능을 호출할 수 있다. 외부 이벤트는 디바이스 간의 연동을 수행할 때 관련된 디바이스로 이벤트를 전달하기 위한 것이다. 그리고 시간 이벤트는 시간과 관련되어 발생하는 이벤트를 정의한다. 즉, 주기적으로 발생하거나, 특정 시간에 발생하는 이벤트를 정의할 수 있다. 조건은 논리 표현식(boolean expression)으로 표현할 수 있으며, 이벤트 메시지에 있는 이벤트 변수나 디바이스에 의해서 유지되는 디바이스 변수를 활용하여 조건을 정의할 수 있다. 액션은 디바이스에 의해서 실행되는 명령을 의미하며, 내부 이벤트를 생성하는 액션인 createIntEvent, 외부 이벤트를 생성하는 액션인 createExtEvent, 웹서비스를 호출하는 액션인 invoke, 이벤트를 포워딩하는 액션인 passEvent, 변수의 데이터를 처리하는 액션인 assign, 일정 시간 지연을 시키는 액션인 delay에 대한 명령을 포함한다.



(그림 2) WS-ECA 룰의 구조

(그림 3)은 WS-ECA 룰 기술 언어의 전체 구조를 보여준다. ECARule은 ECA 기반의 룰 기술 언어의 루트 엘리먼트로 name, targetNamespace, issuedDate, expiredDate 속성을 갖는다. name 속성은 WS-ECA 룰 이름을 정의하기 위한 것이고, targetNamespace는 WS-ECA 룰의 네임스페이스를 지정한다. issuedDate와 expiredDate는 각각 WS-ECA 룰이 효력을 발휘하기 시작하는 시각과 끝나는 시각을 지정한다. issuedDate와 expiredDate 두 속성의 값이 모두 없으면 그 룰 문서는 영원히 유효하며, issuedDate만 있으면 issuedDate 이후로 영원히 유효하고, expiredDate만 있으면 현재부터 expiredDate 까지만 유효하다. WS-ECA 룰 기술 언어에는 ECA 룰을 정의하는데 필요한 변수(variables), 이벤트(events), 액션(actions)을 정의하는 부분과 ECA 룰을 정의하는 부분으로 나뉜다.

```
〈ECARule name="ncname" targetNamespace="xsd:anyURI" issuedDate="xsd:dateTime"
expiredDate="xsd:anyURI"〉

 〈variables〉 variable+ 〈/variables〉
 〈events〉 event+ 〈/events〉
 〈actions〉 action+ 〈/actions〉
 〈rules〉
  〈rule name="NCName"〉
    〈documentation〉 comments 〈/documentation〉
    〈event name="NCName" /〉
    〈condition expression="XPath Expression" /〉
    〈action name="NCName" /〉
  〈/rule〉
  ...
 〈/rules〉
〈ECARule〉
```

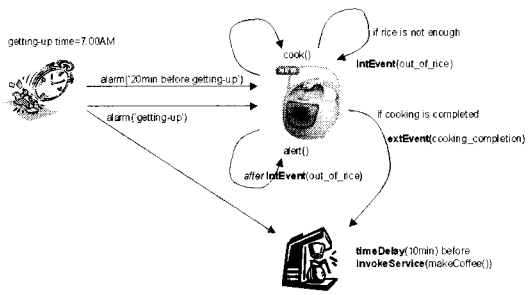
(그림 3) WS-ECA 룰 기술 언어의 전체구조

본 절의 내용은 참고문헌 [9]의 내용을 기반으로 하였음으로, 자세한 사항은 참고문헌 [9]에 설명되어 있다.

2.3 WS-ECA 프레임워크 기반 시나리오

이 절에서는 디바이스 연동을 위해 WS-ECA 룰이 어떻게 활용될 수 있는지를 "morning cooking service" 시나리오를 통하여 설명한다.

(시나리오) Jack은 자기 전에 7:00AM에 자명종 기상 알람을 맞춰놓는다. 아침이 되면, 기상 시간 20분 전이라는 alarm('20min before getting-up') 메시지를 전기밥솥에게 보낸다. 그러면 밥솥은 밥을 짓기 시작하고, 만약 쌀이 없다면 기상 시간에 alarm('getting-up')에 알림 서비스를 호출하여 (alert(out_of_rice)) 사용자에게 알린다. 밥솥은 밥짓기가 성공적으로 끝나면, 커피메이커에게 밥짓기 성공 이벤트 (intEvent(cooking_completion))를 전달하여 10분 후에 커피를 준비시킨다.



(그림 4) Morning Cooking Service 시나리오

2.3.1 시나리오에 대한 ECA 룰 정의

"morning cooking service" 시나리오에 대한 각 개체별 WS-ECA 룰은 아래와 같이 간결한 형태로 표현하였다.

● 알람시계(Alarm-Clock)

알람시계는 사용자 정의된 기상시각(7:00am)에 대하여 아래와 같은 두 가지 룰을 저장하고 있으며, 이 두 개의 룰은 정의된 기상시각을 기준으로 기상 20분전과 기상시각 정각에 시간 이벤트가 발생하면 외부로 이벤트 메시지를 보낸다.

```
on timeEvent(at 20min before getting-up time)
if today is not holiday
do createExtEvent(alarm(contents='20min
    before getting-up'))
```

```
on timeEvent(at 7:00 AM everyday)
if today is not holiday
do createExtEvent(alarm(contents='getting-up'))
```

● 전기밥솥(Rice-Cooker)

전기밥솥은 "morning cooking service"와 관련하여 아래와 같이 다섯 개의 룰을 저장하고 있다. 첫 번째 룰은 외부로부터 알람 메시지를 받으면 이 메시지의 내용이 '20min before getting-up' 혹은 '30min before dinner' 인지를 확인하고 맞으면 내부 이벤트로 'cooking'을 생성한다. 두 번째와 세 번째 룰은 내부 이벤트로 'cooking' 이벤트가 발생하면 밥솥에 쌀을 확인하여 충분하면 밥짓기(cook())를 시작하고, 쌀이 부족하면 내부 이벤트로 'out_of_rice'를 생성한다. 네 번째 룰은 1시간 내에 'out_of_rice' 내부 이벤트가 발생한 후 'alarm' 외부 이벤트가 발생하면, 알람 메시지의 내용이 'getting-up'인지 확인하고 맞으면 사용자에게 경고 메시지를 보낸다. 이 룰의 의미는 쌀이 없으면, 사용자의 기상 시각에 경고 메시지를 보내라는 의미이다. 다섯 번째 룰은 'cooking-completion' 내부 이벤트가 발생하면 'cooking-completion' 외부 이벤트를 생성하라는 의미이다. 즉, 밥짓기가 완료되면, 커피 메이커에게 이를 알려준다.

```
on extEvent(alarm)
if alarm.contents='20min before getting-up' or
    '30min before dinner'
do createIntEvent(cooking)
```

```
on intEvent(cooking)
if rice is enough
do invokeService(cook())
```

```
on intEvent(cooking)
if rice is not enough
do createIntEvent(out_of_rice)
```

```
on extEvent(alarm) after
    intEvent(out_of_rice) within 1hr
if alarm.contents='getting-up'
do invokeService(alert("out of rice"))
```

```
on intEvent(cooking_completion)
if cooking is succeeded.
do passEvent(cooking_completion)
```

● 커피 메이커(Coffee-Maker)

커피 메이커는 "morning cooking service"와 관련하여 아래와 같이 한 개의 룰을 저장하고 있다. 이 룰은 30분 안에 'alarm'과 'cooking_completion' 두 개의 외부 이벤트가 발생하면 alarm 메시지의 내용이 'getting-up'인지 확인하고 맞으면 10분 후에 커피를 조제하라는 의미이다. 즉, 밥짓기가 완료되었고 사용자가 기상 했으면 10정도 후에 커피를 준비하라는 것이다.

```
on extEvent(cooking_completion) and
    extEvent (alarm) within 30min
if alarm.contents='getting-up'
do timeDelay(10min) before invokeService
    (makeCoffee())
```

3. 서비스/이벤트 기술을 위한 WSDL 확장

유비쿼터스 환경은 기본적으로 디바이스의 출현과 사라짐이 반복되는 특성이 있으며, 추가적으로 새로운 디바이스의 출현도 고려해야 한다. 이러한 동적인 특성을 지원하기 위해서, WS-ECA 프레임워크는 새로운 디바이스의 출현 시 디바이스가 제공하는 새로운 서비스와 새로운 이벤트에 대한 정보를 요청하여 관리해야 한다. 즉, 새로운 디바이스가 출현하는 동적인 상황에도 룰 기반 서비스 요청자가 새로운 디바이스가 제공하는 서비스와 이벤트를 포함하여 디바이스 연동을 정의하고 서비스를 받을 수 있도록 지원해야 한다. 이를 위해 WS-ECA 프레임워크는 현재 프레임워크

환경에서 제공되는 서비스와 이벤트에 대한 표준화된 기술 방법이 필요하다.

WS-ECA 프레임워크에서 서비스는 웹서비스를 의미하므로, 서비스에 대한 정보는WSDL(Web Services Description Language) 형태로 기술 한다. WSDL에는 서비스를 이용하기 위해 필요한 입력 인자 및 결과 값에 사용할 복합 타입의 정의, 다른 웹서비스들과 주고받을 메시지, 제공하는 원격 프로시저, 서비스를 호출하는데 사용해야 하는 프로토콜, 서비스 시스템의 endpoint 정보 등을 기술한다. 하지만 이 정보들은 웹서비스의 클라이언트 애플리케이션이 서비스를 호출하는 관점에서 기술되기 때문에 이벤트에 대한 정보를 기술하는 것은 부적합하다. 따라서 본 논문에서는 서비스와 이벤트 정보 모두를 기술하기 위한 방법으로 다음과 같이 WSDL을 확장하는 방법을 제안한다.

(그림 5)는 WSDL에서 메시지에 사용할 데이터 타입 및 메시지를 정의하는 문법을 보여준다. types 엘리먼트는 메시지에 들어갈 complex type을 정의할 때 사용되며, 호환성을 극대화하기 위해서 XML 스키마(XML Schema)를 이용하여 정의한다. message 엘리먼트는 웹서비스들 간에 주고받을 메시지를 정의한다. 이와 같은 types와 message 엘리먼트는 이벤트 알림 메시지를 정의할 때도 WSDL의 변경 없이 그대로 사용이 가능하다. 즉, 이벤트 알림 메시지에 대한 표현은 types 엘리먼트를 이용해서 complex type으로 정의하고, message 엘리먼트에서 이를 이용해서 메시지를 정의 할 수 있다.

```
〈definitions .... 〉
 〈types〉
  〈xsd:schema .... /〉 *
 〈/types〉
 〈message name="nmtoken"〉 *
  〈part name="nmtoken" element="qname"?
type="qname"?/〉 *
 〈/message〉
〈/definitions〉
```

(그림 5) WSDL의 데이터 타입 및 메시지 정의 문법

웹서비스의 경우 구체적인 서비스에 대한 부분은 WSDL의 portType과 operation 엘리먼트를 이용하여 정의한다. operation 엘리먼트는 서비스의 이름, 입력/출력 매개변수를 정의하고, portType 엘리먼트는 operation들의 집합을 하나의 인터페이스로 제공한다. 그러나 이벤트의 경우 이벤트 알림 메시지는 operation 엘리먼트에 출력 매개변수만 사용하여 기술은 가능하지만, 이벤트 알림 메시지의 식별을 위한 action URI 기술과 일반적인 출력 매개변수를 갖는 웹서비스 프로시저와의 구분을 위한 식별 정보가 추가적으로 필요하다. 따라서 operation 엘리먼트는 다음과 같이 확장이 필요하다.

```
〈wsdl:definitions .... 〉
 〈wsdl:portType .... 〉 *
  〈wsdl:operation name="nmtoken"
eca:actionURI="xsd:anyURI"
   eca:eventNotification="(yes|no)"〉
    〈wsdl:output name="nmtoken"?
message="qname"/〉
    〈/wsdl:operation〉
 〈/wsdl:portType 〉
〈/wsdl:definitions〉
```

(그림 6) 이벤트 정의를 위한 WSDL operation 엘리먼트 확장

이벤트 정의를 위해서 WSDL의 operation 엘리먼트에 eca:actionURI와 eca:eventNotification 속성을 추가하여 확장하였다. eca:actionURI는 이벤트의 고유한 ID로, WS-Addressing의 message properties에서 action 부분에 사용된다. 또한, eca:eventNotification 속성은 일반적인 웹서비스와 이벤트 메시지를 구분하기 위한 것으로, 이벤트 메시지이면 yes, 웹서비스이면 no 값을 갖는다.

또한, 웹서비스의 경우 WSDL에서 서비스 호출에 사용할 endpoint의 기술을 위해 service와 port 엘리먼트를 이용하는 반면, 이벤트는 클라이언트 애플리케이션이 이벤트 등록을 요청 시 필요한

이벤트 소스(event source) endpoint의 URI와 이벤트의 ID 정보가 필요하다. 따라서 아래와 같이 WSDL을 확장이 필요하다.

```
〈wsdl:definitions  .... 〉
 〈wsdl:service .... 〉 *
  〈wsdl:port name="nmtoken"
binding="qname"〉 *
    〈eca:event sourceURI= "xsd:anyURI"
     eventID= "xsd:anyURI" /〉
  〈/wsdl:port〉
 〈/wsdl:service〉
〈/wsdl:definitions〉
```

(그림 7) 이벤트 정의를 위한 WSDL port 엘리먼트의 자식 엘리먼트 확장

이와 같은 WSDL 확장을 통해서 서비스 제공자와 이벤트 소스 모두가 WSDL을 이용하여 자신의 서비스와 이벤트를 정의할 수 있다.

4. WS-ECA 프레임워크 프로토콜

WS-ECA 프레임워크는 룰 기반 서비스 요청자, 서비스 제공자(Service Provider), 이벤트 소스(Event Source), 디바이스, 글로벌 룰 매니저(Global rule manager)의 5개 컴포넌트로 구성되어 있다. 룰 기반 서비스 요청자는 WS-ECA 룰을 이용하여 서비스 제공자, 이벤트 소스, 디바이스가 어떤 로직을 가지고 연동되어야 하는지 기술하고, 이 룰에 의해서 제공되는 서비스를 받는다. 서비스 제공자는 서비스를 제공하고, 이벤트 소스는 이벤트 알림 메시지(event notification message)를 등록된 이벤트 싱크들에게 제공한다. 디바이스는 자신이 처리할 WS-ECA 룰을 저장하고 있다가, 이벤트를 받으면 조건을 검사한 후 필요한 액션을 수행한다. 글로벌 룰 매니저(Global rule manager)는 자신의 네트워크에 연결되어 있는 디바이스들을 관리한다.

본 장에서는 WS-ECA 프레임워크의 각 컴포넌트들이 자신의 역할을 올바르게 수행하기 위해 다른 컴포넌트들과 연동하는 각 과정에서 필요한 프로토콜에 대해 기술한다.

4.1 프로토콜의 기본 정보 소개

WS-ECA 프레임워크는 컴포넌트 간의 메시지 교환을 하는데 기본적으로 W3C의 WS-Addressing 표준을 따른다. 이 표준은 웹서비스가 전달하는 메시지가 올바르게 처리되기 위해 가지고 있어야 하는 최소한의 정보들을 정의하고 있다. 따라서 본 절에서는 WS-ECA 프레임워크의 모든 메시지가 포함해야하는 이들 기본 정보에 대한 소개와 이들이 SOAP 메시지에 어떻게 포함되어야 하는지 설명한다.

(1) Destination

Destination은 메시지를 받는 endpoint의 주소로, IRI (Internationalized Resource Identifier)를 이용해 기술한다. 메시지가 전달되어야 하는 주소를 명시하고 있기 때문에 모든 프로토콜에서 반드시 기술되어야 한다.

(2) Source endpoint

Source endpoint는 메시지가 생성된 endpoint를 endpoint reference로 표현한다. 이 정보는 메시지가 어디에서 생성되었는지 확인하기 위한 정보로 모든 프로토콜 메시지에서 반드시 필요로 하는 것은 아니다. 메시지를 처리하는 웹서비스에서 필요로 하거나, source endpoint 정보를 기반으로 메시지 내용을 처리할 필요가 있는 경우에 기술한다. 필요 여부는 서비스나 프로토콜 핸들러(protocol handler)의 구현에 따라 다를 수 있다.

(3) Reply endpoint

Reply endpoint는 자신이 보낸 메시지가 처리된 후 메시지가 처리된 결과 메시지를 받기 위한

endpoint 정보를 endpoint reference로 기술한다. WS-ECA 프레임워크에서 정의되는 프로토콜은 요청-응답 (request-reply) 모드를 사용하기 때문에, reply endpoint 정보는 요청 메시지들이 반드시 기술해야 한다.

(4) Fault endpoint

Fault endpoint는 메시지 처리 과정에서 생기는 오류 메시지를 받는 endpoint 정보를 endpoint reference로 기술한다. Fault endpoint는 오류 메시지를 처리할 경우에만 필요하기 때문에, 프로토콜 핸들러(protocol handler) 혹은 웹서비스가 오류 사항을 어떻게 처리하는가의 방법에 따라 메시지에 포함 여부가 결정된다.

(5) Action

Action은 메시지가 어떤 작업을 수행하기 위한 것인지 나타내기 위한 것으로, IRI를 이용해 기술한다. 이를 기반으로 메시지의 처리가 결정되기 때문에 모든 메시지는 action을 반드시 기술해야 한다. 메시지 별로 action 정보에 기술되어야 할 사항들은 각각의 메시지에 대한 세부 정보에서 다시 설명한다.

(6) Message id

Message id는 메시지를 구분하기 위한 고유의 ID로, IRI를 이용해 표현된다. WS-ECA 프레임워크에서는 다양한 컴포넌트들 사이에서 메시지가 전달되며, 여러 개의 컴포넌트로부터 다양한 메시지가 동시에 전달될 수 있기 때문에 메시지들을 구분하기 위해 message id는 반드시 기술되어야 한다.

(7) Relationship

Relationship은 현재 보내지는 메시지가 다른 메시지와 어떤 관계를 가지고 있는지 IRI로 표현한다. 이 정보는 두 개의 IRI를 필요로 하는데, 하나는 이 메시지가 관계를 맺고 있는 다른 메시지의 ID (message id)이고, 다른 하나는 둘 사이의 관계를 지정해주는 것이다. WS-ECA 프레임워크에서 사용하는 프로토콜은 요청-응답 (request-reply) 모드를 사용하고 있기 때문에, 응답 메시지들은 이 정보를 반드시 기술해야 한다.

위에서 정의한 정보 모델을 XML 포맷으로 표현하면 다음과 같다.

```
〈wsa:To〉 xs:anyURI 〈/wsa:To〉 ?
〈wsa:From〉 wsa:EndpointReferenceType
〈/wsa:From〉 ?
〈wsa:ReplyTo〉 wsa:EndpointReferenceType
〈/wsa:ReplyTo〉 ?
〈wsa:FaultTo〉 wsa:EndpointReferenceType
〈/wsa:FaultTo〉 ?
〈wsa:Action〉 xs:anyURI 〈/wsa:Action〉
〈wsa:MessageID〉 xs:anyURI 〈/wsa:MessageID〉 ?
〈wsa:RelatesTo RelationshipType="xs:anyURI"?〉
xs:anyURI 〈/wsa:RelatesTo〉 *
〈wsa:ReferenceParameters〉 xs:any*
〈/wsa:ReferenceParameters〉 ?
```

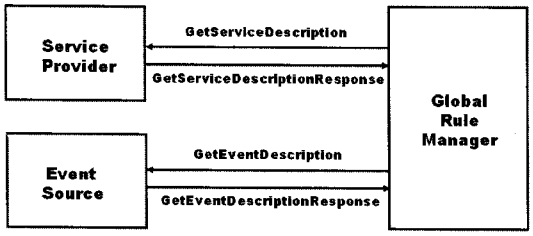
(그림 8) WS-ECA 프레임워크 프로토콜의 기본 정보

이 정보들은 프로토콜에서 사용하는 SOAP 메시지들의 헤더에 기술된다.

4.2 서비스/이벤트 기술 정보 교환 프로토콜

본 절에서는 3장에서 제안한 WSDL 확장 방법을 이용하여 기술된 서비스와 이벤트 정보를 WS-ECA 프레임워크의 해당 컴포넌트 간에 교환하기 위한 프로토콜을 정의한다. 기본적으로 서비스와 이벤트 정보는 서비스 제공자, 이벤트 소스에 의해서 제공되고 글로벌 룰 매니저(Global rule manager)가 관리하므로, 이에 대한 프로토콜을 그림으로 표현하면 (그림 9)와 같다.

GetServiceDescription 메시지는 글로벌 룰 매니저(Global rule manager)가 서비스 제공자에게 서비스 정보를 요청하는 것이고, GetServiceDescriptionResponse는 서비스 제공자가 요청에 대한 응답으로 전달하는 메시지이다. 서비스 기술 정보 교환 프로토콜과 유사하게 이벤트 기술 정보 교환을 위한 요청 및 응답 메시지는 각각 GetEventDescription, GetEventDescriptionResponse로 정의한다.



(그림 9) 서비스/이벤트 상세 정보 교환 프로토콜

(표 1) 서비스/이벤트 상세 정보 교환 프로토콜별 Action

Protocol	Action
GetServiceDescription	http://scsr.etri.re.kr/2007/eca/GetServiceDescription
GetServiceDescriptionResponse	http://scsr.etri.re.kr/2007/eca/GetServiceDescriptionResponse
GetEventDescription	http://scsr.etri.re.kr/2007/eca/GetEventDescription
GetEventDescriptionResponse	http://scsr.etri.re.kr/2007/eca/GetEventDescriptionResponse

GetServiceDescription, GetServiceDescriptionResponse, GetEventDescription, GetEventDescriptionResponse 메시지의 헤더 부분은 프로토콜의 기본 정보들을 포함하고, (표 1)의 해당 action을 지정하여 기술한다. 또한, GetServiceDescription와 GetEventDescription 메시지는 body에 포함할 내용이 없고, GetServiceDescriptionResponse, GetEventDescriptionResponse의 body 부분의 구성은 아래에서 설명한다.

GetServiceDescription 메시지는 헤더에 프로토콜의 기본 정보들과 함께, (그림 10)과 같이 (표 1)의 해당 action을 지정해야 하며, body 부분에 기술할 내용은 없다.

```
〈s:Envelope  ...〉
 〈s:Header ...〉
   〈wsa:Action〉

http://scsr.etri.re.kr/2007/eca/GetServiceDescrip
tion
   〈/wsa:Action〉
  ...
 〈/s:Header〉
 〈s:Body /〉
〈/s:Envelope〉
```

(그림 10) GetServiceDescription 메시지 문법

GetServiceDescriptionResponse 메시지는 헤더에 프로토콜의 기본 정보들과 함께, (표 1)의 해당 action을 지정해야 한다. body에는 GetServiceDescriptionResponse 엘리먼트에 서비스 제공자가 제공하는 서비스들의 정보를 기술한다. 서비스 정보를 기술하는 방법은 두 가지가 있는데, 첫 번째 방법은 서비스 제공자가 제공하는 WSDL의 URI를 전달하여 글로벌 룰 매니저(Global rule manager)가 이를 참조하도록 하는 것이고, 두 번째 방법은 메시지 안에 직접 WSDL 문법을 이용하여 기술하는 것이다. 첫 번째 방법을 사용할 경우, GetServiceDescriptionResponse 엘리먼트는 ServiceDescriptionURI 엘리먼트를 하위 엘리먼트로 갖는다. 두 번째 방법은 WSDL을 정책, 보안 등의 문제를 이유로 모두 공개하고 있지 않거나, 필요한 부분만 전달하고 싶은 경우에 사용한다. 이때는 GetServiceDescriptionResponse 엘리먼트는 ServiceDescription 엘리먼트를 하위 엘리먼트로 갖는다.

GetEventDescription 메시지는 헤더에 프로토콜의 기본 정보들과 함께, (표 1)의 해당 action을 지정해야 하고, body 부분에 기술할 내용은 없다.

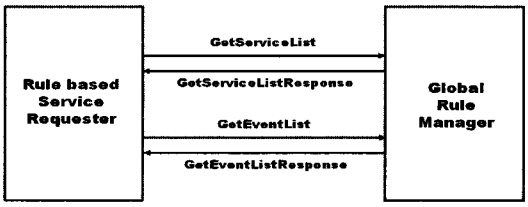
```
〈s:Envelope ...〉
  ...
 〈s:Body〉
  〈GetServiceDescriptionResponse〉

〈ServiceDescriptionURI〉xs:anyURI〈/ServiceDesc
riptionURI〉?
    〈ServiceDescription〉
     〈wsdl:definitions ...〉
     〈/wsdl:definitions〉
    〈/ServiceDescription〉?
  〈/GetServiceDescriptionResponse〉
 〈/s:Body〉
〈/s:Envelope〉
```

(그림 11) GetServiceDescriptionResponse 메시지의 body 부분 문법

GetEventDescriptionResponse 메시지는 헤더에 프로토콜의 기본 정보들과 함께, (표 1)의 해당 action을 지정해야 한다. GetEventDescriptionResponse 메시지 역시, GetServiceDescriptionResponse 메시지와 유사하게 이벤트 정보를 전달함에 있어 WSDL의 URI를 전달하거나 WSDL의 내용을 직접 기술할 수 있도록 한다. 각각의 경우 GetServiceDescriptionResponse 엘리먼트는 EventDescriptionURI 엘리먼트와 EventDescription 엘리먼트를 하위 엘리먼트로 갖는다.

4.5 서비스/이벤트 목록 교환 프로토콜

기본적으로 룰 기반 서비스 요청자는 WS-ECA 룰을 작성할 때 자신이 접속하고 있는 네트워크에서 어떤 서비스와 이벤트가 존재하고 있는지 알아야 한다. 글로벌 룰 매니저(Global rule manager)가 각각의 서비스 제공자와 이벤트 소스가 제공하는 서비스와 이벤트에 대한 정보들을 관리하고 있으므로, 룰 기반 서비스 요청자가 WS-ECA 룰을 작성할 때 글로벌 룰 매니저(Global rule manager)에게 이들 정보들을 얻을 수 있도록 다음과 같은 메시지를 정의한다.



(그림 12) 서비스/이벤트 목록 교환 프로토콜

(표 2) 서비스/이벤트 목록 교환 프로토콜별 Action

Protocol	Action
GetServiceList	http://scsr.etri.re.kr/2007/eca/GetServiceList
GetServiceListResponse	http://scsr.etri.re.kr/2007/eca/GetServiceListResponse
GetEventList	http://scsr.etri.re.kr/2007/eca/GetEventList
GetEventListResponse	http://scsr.etri.re.kr/2007/eca/GetEventListResponse

(표 2)에서 보여주는 각 프로토콜 메시지의 헤더 부분은 프로토콜의 기본 정보들을 포함하고, (표 2)의 해당 action을 지정하여 기술한다. 또한, GetServiceList와 GetEventList 메시지는 body에 포함할 내용이 없고, GetServiceListResponse, GetEventListResponse의 body 부분의 구성은 아래에서 설명한다.

```
〈s:Envelope ...〉
  ...
 〈s:Body〉
  〈GetServiceListResponse〉
 〈ServiceDescriptionURI〉 list of xs:anyURI
〈/ServiceDescriptionURI〉
 〈Definitions〉
  〈wsdl:definition〉
  〈/wsdl:definition〉
 〈/Definitions〉
〈/GetServiceListResponse〉
 〈/s:Body〉
〈/s:Envelope〉
```

(그림 13) GetServiceListResponse 메시지의 body 부분 문법

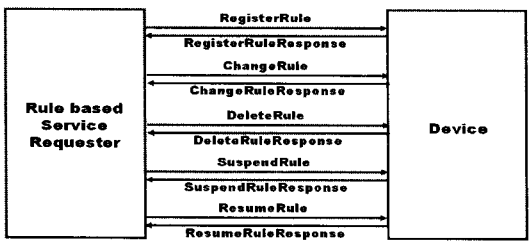
GetServiceListResponse 메시지는 글로벌 룰 매니저(Global rule manager)가 가지고 있는 서비스

정보를 기술하기 위해 body에 GetServiceListResponse 엘리먼트를 갖는다. 서비스 제공자가 서비스 정보를 WSDL의 URI만 제공한 경우와 WSDL의 내용을 직접 기술한 두 가지 경우를 모두 처리하기 위해 GetServiceListResponse 엘리먼트는 하위 엘리먼트로 ServiceDescriptionURI와 Definitions를 가질 수 있다. ServiceDescriptionURI는 그 값으로 xsd:anyURI의 리스트를 갖는데, 이것은 서비스 제공자들이 제공한 WSDL의 URI 리스트들이다. 또한 Definitions에는 서비스 제공자들이 제공한 WSDL의 내용을 기술한다.

GetEventListResponse 메시지는 기본적으로 GetServiceListResponse와 거의 유사하다. body에 이벤트 정보를 기술하기 위해 GetEventListResponse 엘리먼트를 갖는다. GetEventListResponse 엘리먼트는 EventDescriptionURI 엘리먼트와 Definitions 엘리먼트를 하위 엘리먼트로 갖는다.

4.4 WS-ECA 룰 관리 프로토콜

룰 기반 서비스 요청자는 자신이 작성한 WS-ECA 룰을 디바이스에 전달하여, 디바이스가 정의된 룰을 기반으로 다른 디바이스들과 동적으로 연동되기를 기대한다. 이 과정에서 룰 기반 서비스 요청자는 WS-ECA 룰을 디바이스에게 전달해야 하고, 이후 필요에 따라 룰을 바꾸거나, 삭제, 혹은 룰의 적용을 중지하거나 재개시킬 수 있어야 한다. 본 절에서는 룰 기반 서비스 요청자와 디바이스간의 이러한 룰을 관리하기 위한 프로토콜에 대해서 정의한다. 이 과정을 그림으로 표현하면 다음과 같다.



(그림 14) WS-ECA 룰 관리 프로토콜

(표 3) WS-ECA 룰 관리 프로토콜별 Action

Protocol	Action
RegisterRule	http://scsr.etri.re.kr/2007/eca/RegisterRule
RegisterRule Response	http://scsr.etri.re.kr/2007/eca/RegisterRule Response
ChangeRule	http://scsr.etri.re.kr/2007/eca/ChangeRule
ChangeRuleResponse	http://scsr.etri.re.kr/2007/eca/ChangeRule Response
DeleteRule	http://scsr.etri.re.kr/2007/eca/DeleteRule
DeleteRuleResponse	http://scsr.etri.re.kr/2007/eca/DeleteRuleResponse
SuspendRule	http://scsr.etri.re.kr/2007/eca/SuspendRule
SuspendRule Response	http://scsr.etri.re.kr/2007/eca/SuspendRule Response
ResumeRule	http://scsr.etri.re.kr/2007/eca/ResumeRule
ResumeRule Response	http://scsr.etri.re.kr/2007/eca/ResumeRule Response

(표 3)에 정의된 각 프로토콜 메시지의 헤더 부분은 프로토콜의 기본 정보들을 포함하고, (표 3)의 해당 action을 지정하여 기술한다.

룰 기반 서비스 요청자는 자신이 사용할 WS-ECA 룰을 작성하여 이를 디바이스에게 배포한다. RegisterRule 메시지의 body에는 WS-ECA 룰을 참조할 URI나 그 내용을 직접 기술하기 위해 RegisterRule 엘리먼트가 ECARuleURI 엘리먼트와 ECARules를 하위 엘리먼트로 갖는다. ECARuleURI 엘리먼트는 WS-ECA 룰이 위치한 곳의 URI를 전달해서 디바이스가 참조할 수 있도록 하며, ECARules 엘리먼트는 직접 WS-ECA 룰을 기술하는데 사용한다.

```
〈s:Envelope  ...〉
  ...
 〈s:Body〉
  〈RegisterRule〉
    〈ECARuleURI〉 xsd:anyURI 〈/ECARuleURI〉 ?
    〈ECARules〉
    ...
    〈/ECARules〉 ?
  〈/RegisterRule〉
 〈/s:Body〉
〈/s:Envelope〉
```

(그림 15) RegisterRule 메시지의 body 부분 문법

디바이스는 룰 기반 서비스 요청자로부터 RegisterRule 메시지를 받으면, 해당 WS-ECA 룰을 처리한 후 결과에 따라 body의 RegisterRuleResponse 엘리먼트는 "success" or "fail" 값을 기술하여 보낸다.

ChangeRule 메시지의 body에는 WS-ECA 룰을 변경하기 위한 내용들이 ChangeRule 엘리먼트에 기술된다. ChangeRule 엘리먼트의 ruleName은 변경 대상이 되는 룰의 이름을 뜻한다. ChangeRule 엘리먼트는 하위 엘리먼트로 ECARuleURI와 ECARules 두 가지 중 하나를 가질 수 있다. ECARuleURI 엘리먼트는 WS-ECA 룰을 참조하기 위한 URI를 기술하고 있으며, ECARules는 변경되어야 할 사항들을 직접 기술한다.

ChangeRuleResponse 메시지는 처리 결과에 따라 body의 ChangeRuleResponse 엘리먼트에 "success" or "fail" 값을 포함할 수 있다.

DeleteRule 메시지는 body의 DeleteRule 엘리먼트의 ruleName 속성에 삭제할 룰의 이름을 지정한다.

DeleteRuleResponse 메시지는 처리 결과에 따라 body의 DeleteRuleResponse 엘리먼트에 "success" or "fail" 값을 포함할 수 있다.

디바이스에게 배포된 룰을 일시적으로 사용하고 싶지 않은 경우, 룰 기반 서비스 요청자는 디바이스에게 다음의 SuspendRule 메시지를 보낸다. SuspendRule 메시지는 body에 SuspendRule 엘리먼트를 갖고, 이 엘리먼트의 하위 엘리먼트로 name과 until 속성을 포함하는 rule 엘리먼트를 갖는다. name 속성에는 사용을 중지할 WS-ECA 룰의 이름을 기술하고, until에는 언제까지 해당 룰을 사용하지 않을 것인지, xsd:dateTime 형으로 기술한다. 만약 이 값이 생략되어 있다면, 해당 룰은 룰 기반 서비스 요청자가 ResumeRule 메시지로 다시 사용하도록 요청하기 전까지 계속 사용되지 않는다.

SuspendRuleResponse 메시지는 처리 결과에 따라 body의 SuspendRuleResponse 엘리먼트에 "success" or "fail" 값을 포함할 수 있다.

룰 기반 서비스 요청자가 디바이스에게 룰을 사용 중지할 것을 요청한 이후, 다시 그 룰을 사용해야 하는 경우, 즉, SuspendRule 메시지에 사용 중지 기간을 명시하지 않았거나, SuspendRule 메시지에 기술한 사용 중지 기간보다 더 빨리 룰을 사용하기 시작해야 하는 경우에 룰 재사용 요청 메시지인 ResumeRule를 보내야 한다.

ResumeRule 메시지는 body에 ResumeRule 엘리먼트를 갖고, 이의 하위 엘리먼트로 재사용할 룰 이름을 기술할 ruleName 엘리먼트를 갖는다.

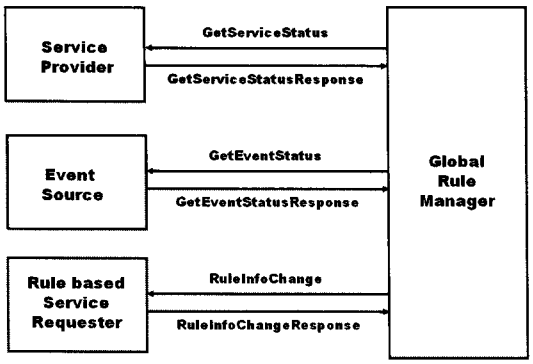
ResumeRuleResponse 메시지는 처리 결과에 따라 body의 ResumeRuleResponse 엘리먼트에 "success" or "fail" 값을 포함할 수 있다.

4.5 서비스/이벤트 상태 확인 프로토콜

WS-ECA 프레임워크에서 디바이스에 배포된 룰은 서비스 제공자에 의해 제공되는 서비스, 혹은 이벤트 소스에게서 전달받는 이벤트 알림 메시지의 내용들을 참조하여 정의된다. 하지만 유비쿼터스 환경에서 처음에 룰이 배포 될 때 사용 가능했던 서비스와 이벤트들이 지속적으로 사용 가능하다고 가정할 수는 없다. 기본적으로 유비쿼터스 환경에서는 애드혹 네트워크(Ad-hoc network)가 가능하기 때문에 서비스나 이벤트들은 필요에 따라 네트워크에서 사라지거나, 일시적으로 사용 가능하지 않을 수도 있다.

따라서 WS-ECA 프레임워크에서는 글로벌 룰 매니저(Global rule manager)가 서비스 제공자와 이벤트 소스의 상태를 점검할 수 있는 프로토콜을 정의하고, 이 정보를 바탕으로 룰 기반 서비스 요청자가 정의한 룰에 기술된 서비스나 이벤트가 유효하지 않으면, 룰 기반 서비스 요청자에게 이 사실을 알려 룰을 수정하거나 필요한 조치를 취하도록 해야 한다.

이 과정의 프로토콜을 그림으로 표현하면 다음과 같다.



(그림 16) 서비스/이벤트 상태 확인 프로토콜

(표 4) WS-ECA 룰 관리 프로토콜별 Action

Protocol	Action
GetServiceStatus	http://scsr.etri.re.kr/2007/eca/GetServiceStatus
GetServiceStatus Response	http://scsr.etri.re.kr/2007/eca/GetServiceStatusResponse
GetEventStatus	http://scsr.etri.re.kr/2007/eca/GetEventStatus
GetEventStatusResponse	http://scsr.etri.re.kr/2007/eca/GetEventStatusResponse
RuleInfoChange	http://scsr.etri.re.kr/2007/eca/RuleInfoChange
RuleInfoChange Response	http://scsr.etri.re.kr/2007/eca/RuleInfoChangeResponse

글로벌 룰 매니저(Global rule manager)는 서비스의 상태를 확인하기 위해 서비스 제공자에게 상태 확인 요청 메시지 GetServiceStatus를 보낸다.

```
〈s:Envelope ...〉
...
 〈s:Body〉
  〈GetServiceStatus〉
  〈service portType="qname" operation="qname" /〉
  〈/GetServiceStatus〉
 〈/s:Body〉
〈/s:Envelope〉
```

(그림 17) GetServiceStatus 메시지의 body 부분 문법

GetServiceStatus 메시지는 body에 GetServiceStatus 엘리먼트를 갖고, 이 엘리먼트는 상태를 알고자 하는 서비스를 기술하기 위한 portType과 operation 속성을 포함하는 service 엘리먼트를 갖는다.

```
〈s:Envelope ...〉
  ...
 〈s:Body〉
  〈GetServiceStatusResponse〉
   〈service portType="qname"
            operation="qname"〉
    〈status〉 "ready | not exist | suspended"
            〈/status〉
     〈reactive〉 xsd:dateTime 〈/reactive〉
  〈/service〉
 〈/GetServiceStatusResponse〉
 〈/s:Body〉
〈/s:Envelope〉
```

(그림 18) GetServiceStatusResponse 메시지의 body 부분 문법

서비스 제공자가 글로벌 룰 매니저(Global rule manager)에게 보내는 GetServiceStatusResponse 메시지는 body의 GetServiceStatusResponse 엘리먼트에 서비스의 상태를 기술한다. GetServiceStatusResponse 엘리먼트는 각각의 서비스의 상태를 기술하기 위한 service 엘리먼트를 하위 엘리먼트로 갖는다. service 엘리먼트는 WSDL의 portType과 operation 속성을 갖으며, 서비스의 상태를 기술하기 위한 status와 reactive 엘리먼트를 하위 엘리먼트로 갖는다. status 엘리먼트는 서비스의 현재 상태를 기술하는 것으로 "ready", "not exist", "suspended" 중 하나를 가질 수 있다. "ready"는 사용할 수 있는 상태임을 의미하며, "not exist"는 글로벌 룰 매니저(Global rule manager)가 요청한 서비스가 존재하지 않음을 뜻하고, "suspended"는 서비스를 일시적으로 사용할 수 없는 경우를 뜻한다. 이때, status의 값이 "suspended"이면 서비스가 언제 다시 사용 가능한지 reactive 엘리먼트에 xsd:dateTime 형으로 기술해준다.

GetServiceStatus와 마찬가지로, 이벤트 소스가 제공하는 이벤트의 상태를 확인하기 위해 글로벌 룰 매니저(Global rule manager)는 이벤트 소스에

게 이벤트 상태 확인 요청 메시지 GetEventStatus를 보낸다.

GetEventStatus 메시지는 body의 GetEventStatus 엘리먼트에 상태 확인을 요청할 이벤트에 대해 기술한다. GetEventStatus 엘리먼트의 하위 엘리먼트인 event 엘리먼트는 상태를 검사할 이벤트들에 대한 정보를 기술하는 것으로, WSDL에 정의된 URI 값을 eventID 속성에 기술하고, eventSink 속성은 현재 이 이벤트를 받고 있는 이벤트 싱크(event sink)들을 나타낸다.

GetEventStatusResponse는 body의 GetEventStatus Response 엘리먼트에 각 이벤트들의 상태를 기술한다. GetEventStatusResponse는 event 엘리먼트를 하위 엘리먼트로 가지며, 이 엘리먼트가 하나의 이벤트에 대한 상태를 기술한다. event 엘리먼트는 이벤트를 구분하기 위한 eventID 속성을 갖는다. event 엘리먼트는 하위 엘리먼트로 status, reactive와 NoSubscriptionEventSink 엘리먼트를 가질 수 있다. status 엘리먼트는 이벤트의 현재 상태를 나타내며, "ready", "not exist", "suspended", "no subscription" 중 하나의 값을 가질 수 있다. "ready"는 새로운 이벤트가 발생하면 이벤트 알림 메시지를 전달할 수 있음을 의미한다. "not exist"는 명시한 이벤트가 현재 이벤트 소스에 존재하지 않음을 뜻하며, "suspended"는 이벤트 소스가 일시적으로 중단되어 있는 상태를 뜻한다. 마지막으로 "no subscription"은 이벤트가 존재하고 문제는 없으나, 전달 받은 이벤트 싱크(event sink)들 중 이벤트를 받지 않는 것이 존재하고 있음을 뜻한다.

status 엘리먼트의 값이 "suspended"인 경우, 이 이벤트가 언제 다시 사용 가능하게 되는지를 기술하기 위해 reactive 엘리먼트를 사용한다. reactive 엘리먼트는 xsd:dateTime 형으로 이벤트가 다시 사용 가능한 시기를 기술한다. 또한 status의 값이 "no subscription"인 경우 NoSubscriptionEventSink 엘리먼트에 전달 받은 이벤트 싱크(event sink)들 중, 이 이벤트를 받지 않는 이벤트 싱크(event sink)들의 URI들을 기술한다.

글로벌 룰 매니저(Global rule manager)가 서비스와 이벤트의 상태를 확인 한 후, 상태가 변경되어 룰이 제대로 실행될 수 없을 경우 이 사실을 룰 기반 서비스 요청자에게 RuleInfoChanged 메시지를 통해 알리게 된다.

```
〈s:Envelope  ...〉
  ...
 〈s:Body〉
  〈RuleInfoChanged〉
   〈service portType="qname"
            operation="qname"
    status="( not exist | suspended |service
            provider not exist )"〉
     〈RelatedRules〉
      〈rule〉
      〈/rule〉
   〈/RelatedRules〉
   〈reactive〉xsd:dateTime〈/reactive〉
    〈/service〉
  〈event eventID="xsd:anyURI"
  status="( not exist | suspended | no
       subscription | event source not exist )"〉
      〈RelatedRules〉
       〈rule〉
       〈/rule〉
   〈/RelatedRules〉
      〈reactive〉xsd:dateTime〈/reactive〉
     〈/event〉
〈/RuleInfoChanged〉
 〈/s:Body〉
〈/s:Envelope〉
```

(그림 19) RuleInfoChange 메시지의 body 부분 문법

RuleInfoChange 메시지는 body의 RuleInfoChanged 엘리먼트에 상태가 변경된 서비스와 이벤트의 상태들을 기술한다. RuleInfoChanged 엘리먼트는 service와 event 하위 엘리먼트를 가질 수 있으며, 각각 서비스와 이벤트의 상태 변화 내용을 기술하고 있다.

service 엘리먼트는 서비스의 상태 변화를 기술하는 엘리먼트이다. service 엘리먼트는 portType,

operation, status 속성을 가지고 있는데, portType과 operation은 서비스를 구분하기 위한 정보이고, status 속성은 서비스의 상태를 기술하는 속성으로, "not exist", "suspended", "service provider not exist"의 세 가지 값을 가질 수 있다. "not exist"와 "suspended"는 GetServiceStatusResponse 메시지에서 전달된 서비스의 상태 값과 동일한 의미로 사용되며, "service provider not exist"는 GetServiceStatusResponse 메시지를 받지 못해서 서비스 제공자 자체가 네트워크에 존재하지 않는다고 판단되는 경우를 뜻한다.

service 엘리먼트의 하위 엘리먼트인 RelatedRules는 상태가 변한 해당 서비스와 관련된 WS-ECA 룰들을 기술한다. 이와 같이 관련된 룰 정보를 룰 기반 서비스 요청자에게 알려주는 것은 관련된 룰의 삭제, 수정 등의 조치가 가능하도록 하기 위한 것이다. 또한 서비스의 상태가 "suspended"인 경우엔 reactive 엘리먼트에 언제 서비스가 다시 사용 가능한지 xsd:dateTime 형으로 기술해줘서 해당하는 룰들을 서비스가 사용 가능할 때까지 사용을 중지시킬 수도 있도록 한다.

event 엘리먼트는 하나의 이벤트의 상태 변화를 나타낸다. event 엘리먼트는 eventID 속성과 status 속성을 가지고 있는데, eventID 속성은 하나의 이벤트를 식별하기 위한 URI이다. 또한 status 속성은 GetEventStatusResponse 메시지에서 받은 이벤트의 상태를 나타내는 것으로, "not exist", "suspended", "no subscription", "event source not exist" 중 하나의 값을 가질 수 있다. "not exist", "suspended", "no subscription"은 GetEventStatusResponse 메시지에서 얻은 이벤트의 상태와 같은 의미로 사용된다. "event source not exist"는 글로벌 룰 매니저(Global rule manager)가 GetEventStatusResponse 메시지를 받지 못해 이벤트 소스가 네트워크에 존재하지 않다고 판단되는 경우를 뜻한다.

event 엘리먼트는 하위 엘리먼트로 RelatedRules와 reactive를 가질 수 있으며, 이것은 service 엘리먼트와 동일한 의미로 사용한다.

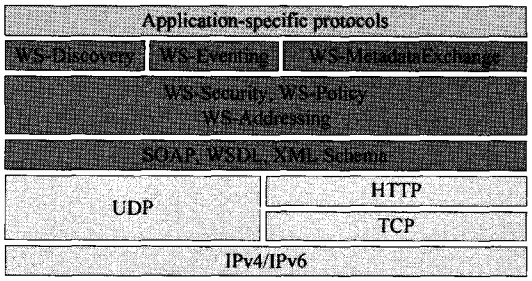
RuleInfoChangedResponse 메시지는 body의 RuleInfoChangedResponse 엘리먼트에 메시지를 잘 받았다는 의미로 "ack"를 포함할 수 있다.

5. 관련연구

본 장에서는 웹서비스 기반의 디바이스 연동과 관련된 기존의 연구들에 대해서 살펴보고, 본 논문에서 제안하는 프레임워크 기반을 두고 있는 웹서비스 및 관련 표준에 대해서 살펴본다.

5.1 웹서비스 기반의 디바이스 연동

마이크로소프트의 Invisible Computing 프로젝트는 웹서비스가 e-비즈니스 환경에 있는 상호운용성 문제를 해결하기 위해 고안되었던 점을 착안하여, 유비쿼터스 컴퓨팅 환경에 있는 상호운용성 문제에 대한 실마리를 찾기 위한 목적으로 진행되었다. 이는 실제로 웹서비스를 소형 디바이스에 적용할 때의 발생할 수 있는 설정, 성능, 보안 등의 다양한 문제점들에 대해서도 검토하였다. 결과적으로 웹서비스는 디바이스에 적용 가능성이 있으나 디바이스에 적합한 추가적인 표준의 필요성과 효율적인 메시지 교환 방법 등 보완이 필요한 부분이 상당히 있다[16]. Invisible Computing 프로젝트는 웹서비스를 디바이스 연동에 적용하였을 경우에 대한 가능성 및 추가적으로 보완이 필요한 이슈들에 대한 내용에 중점을 두었다. (그림 20)은 Invisible Computing 프로젝트의 DPWS 프레임워크 구조를 보여준다. 서로 다른 디바이스에서 제공하는 서비스를 찾는 것은 WS-Discovery 프로토콜을 이용하며, 서비스가 변경된 경우에 대한 부분은 WS-Eventing 프로토콜을 이용하여 알려준다. 또한, 디바이스 간의 메시지 교환 시 기본적으로 WS-Addressing 표준을 따라 메시지를 표현한다.



(그림 20) Invisible Computing의 DPWS(Device Profile for Web Services) 프레임워크 구조

Trento 대학의 Smart Home 프로젝트는 홈 내의 모든 디바이스, 센서, 가전들을 연결하여 보다 편리한 가정 내의 서비스 제공을 위한 SOA 기반의 아키텍처와 웹서비스 기반의 서비스 생성 방법을 제안하였다. 홈 안의 모든 디바이스는 표준 웹서비스 스택을 내장하고 있으며, SOAP 메시지를 교환한다. 가전에 의해서 제공되는 서비스는 WSDL로 기술되고, 사용 정책은 WS-Policy로 정의되며, 서비스는 UDDI 레지스트리를 통해서 등록 및 검색된다. 그리고 모든 홈 디바이스들의 인터액션(Interaction)을 조율하기 위해 WS-Notification을 사용한다. 본 연구는 웹서비스를 기반으로 WS-Notification을 활용하여 센서와 관련된 응용 간의 연동을 지원하는 프레임워크를 제안하고 있다. 즉, 센서와 가전 사이에 이벤트 서버(Event Server)가 위치하는 구조로 되어 있으며, 이벤트 서버는 센서로부터 이벤트를 전체적으로 관리하고, 가전으로 부터는 이벤트 서비스에 대한 등록(subscription)을 받아 센서에서 이벤트가 발생하면 등록된 가전에 이벤트를 전달해 준다. 응용은 이벤트를 받아 사용자에게 지능화된 서비스를 제공할 수 있다[5]. 그러나 이러한 방식은 센서는 이벤트를 생성하는 역할을 하고, 가전은 이벤트를 이용하는 역할의 구조로 되어 있으므로 가전들 간의 연동에 대한 한계를 가지고 있다.

5.2 웹서비스 개념 및 관련 표준

5.2.1 웹서비스 개념

웹 서비스는 세 개의 구성요소로 이루어진다. 우선 웹 서비스를 구현하고 서비스를 제공하는 서비스 제공자와 구현된 서비스를 이용하는 서비스 이용자, 그리고 웹 서비스를 등록하고 검색할 수 있는 서비스 중개자로 이루어진다. 서비스 제공자는 웹 서비스를 구현하고 서비스를 제공하는 것으로 서비스 이용자가 자신의 서비스를 이용할 수 있도록 서비스에 대한 상세 정보를 서비스 중개자에게 공개해야 하며, 특히 자신의 서비스에 대한 인터페이스 정보를 정의하고 있는 WSDL을 제공해야한다. 서비스 중개자는 웹 서비스를 등록하고 검색할 수 있는 저장소로 웹 서비스에 대한 검색 엔진과 같은 역할을 한다. 서비스 이용자는 서비스 중개자로부터 자신이 이용하고자 하는 웹 서비스를 검색하고, 원하는 서비스의 인터페이스 정보인 WSDL을 기반으로 서비스 제공자가 제공하는 웹 서비스를 이용한다. 웹서비스는 기본적으로 XML로 표현되는 SOAP(Simple Object Access Protocol)을 사용하므로, 플랫폼에 독립적인 특성을 갖는다. 따라서 웹서비스는 이종의 시스템들로 구성되는 인터넷이나 유비쿼터스 환경에서 시스템 간의 상호운용성을 보장해 줄 수 있는 표준 기술이다.

5.2.2 WS-Eventing(Web Services Eventing)

웹서비스들은 다른 서비스들이나 응용에서 발생되는 이벤트를 활용하여 개선된 서비스를 제공할 수 있으며, 이를 위해 웹서비스가 관심 있는 이벤트가 있으면 등록하고 이를 기반으로 이벤트를 전송하는 메커니즘이 필요하다. 이것은 하나의 웹서비스("subscriber" 역할)가 이벤트 메시지("notification" 또는 "event messages")를 발생하는 다른 웹 서비스("event source" 역할)에 등록("subscription")하는 프로토콜을 필요로 한다. WS-Eventing은 Subscribe, Renew, GetStatus, Unsubscribe, Subscribe End 등 이벤트 등록, 운영

및 관리에 필요한 기본적인 프로토콜을 정의하고 있다[14].

ECA의 개념을 웹서비스 기반으로 적용하기 위해서 WS-Eventing 표준의 활용이 필요하다.

5.2.3 WS-Addressing (Web Services Addressing)

일반적으로 통신이 가능한 환경에서 종단 간의 정보 교환을 하기 위해서는 전송 프로토콜과 메시징 시스템에 대한 정보가 필요하다. WS-Addressing은 웹서비스 Endpoint가 정보를 교환할 때 트랜스포트나 응용과는 독립적으로 처리할 수 있는 두 개의 표준화된 포맷을 정의한다. 하나는 종단 참조(Endpoint Reference)이고 다른 하나는 정보 헤더(Information Header)이다. 종단 참조는 웹서비스 종단을 식별하기 위해 필요한 정보를 전달하며, 정보 헤더는 정보의 전송지와 목적지에 대한 주소를 포함한 종단간의 메시지 특성 정보를 전달한다[15].

WS-Addressing은 응용들이 트랜스포트 및 응용의 메시징 시스템과 독립적으로 정보를 교환 할 수 있는 메커니즘을 제공하므로, 유비쿼터스 환경과 같이 이종의 다양한 디바이스들이 존재하는 환경에서 상호운용성 보장을 위한 중요한 표준이다.

6. 결론

최근 컴퓨팅 환경이 수십억 개의 디바이스가 긴밀하게 네트워크로 연결되는 유비쿼터스 환경으로 빠르게 발전하면서, 이종의 하드웨어와 소프트웨어 플랫폼을 가지고 있는 다양한 디바이스들이 상호운용성을 보장하면서 자연스럽게 연동할 수 있는 대안으로 웹서비스가 대두되고 있다. 그러나 아직까지 웹서비스를 기반으로 디바이스 간의 연동을 쉽게 정의하고, 이를 기반으로 연동을 수행하는 프레임워크에 대한 연구는 없다.

비즈니스 분야의 WS-BPEL과 WS-CDL과 같은 서비스 연동을 기술하기 위한 표준은 웹서비스를 기반으로 외부의 다양한 사업 파트너들과 내부의 시스템간의 효율적인 통합을 가능하게 한다. 그러나 이들은 기본적으로 유비쿼터스 환경의 동적인 특성을 고려하지 않았고, 스펙이 방대한 양의 기능을 정의하고 있어 이러한 기능을 유비쿼터스 환경의 소형 디바이스에 적용하는 것은 적합하지 않다. WS-ECA 프레임워크는 유비쿼터스 환경의 동적인 특성을 지원하고, 소형 디바이스에 적용 가능한 ECA 기반 룰 기술 언어 및 이를 실행할 수 있는 프레임워크를 제안하였다. 그러나 이러한 동적인 디바이스 연동 환경을 실현할 때 반드시 필요한 프레임워크 구성 컴포넌트들 간의 구체적인 프로토콜이 제안되어 있지 않다. 또한, 서비스 정보에 대한 기술은 WSDL(Web Service Description Language)을 이용해서 표현할 수 있으나, 아직까지 WS-Eventing 표준을 위한 이벤트 기술 언어는 없다.

그러므로 본 논문에서는 WS-ECA 프레임워크 환경의 서비스/이벤트 정보를 모두 기술할 수 있는 WSDL 확장 방법을 제안하였고, 프레임워크에 필요한 서비스/이벤트 기술 정보 교환 프로토콜, 서비스/이벤트 목록 교환 프로토콜, WS-ECA 룰 관리 프로토콜, 서비스/이벤트 상태 확인 프로토콜을 개발하였다.

향후 WS-ECA 룰을 쉽게 정의할 수 있도록 기존의 WS-BPEL 에디터로 생성한 WS-BPEL 스크립트를 WS-ECA 룰 언어로 변환하는 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] Ha, W.K., Kim, D.H., and Choi, N.H., The Ubiquitous IT Revolution and the Third Space, The Korea Electronic Times, 2002.

[2] Shankar R. Ponnekanti, Brian Lee, Armando Fox, Pat Hanrahan, and Terry Winograd

ICrafter: "A Service Framework for Ubiquitous Computing Environments" Proceedings of Ubicomp 2001, September 30-October 2, 2001

[3] Z. Maamar, On coordinating personalized composite web services, Information and Software Technology 48 (7) (2006) 540 - .548.

[4] Microsoft, The Microsoft invisible computing project web site. <http://research.microsoft.com/invisible/>.

[5] Aiello, M., Marchese, M., Busetta, P., and Calabrese, G. (2005a). Opening the Home: a Web Service Approach to Domotics. In Applied Computing 2005, volume I:271-279.

[6] UPnP, The UPnP forum web site. <http://www.upnp.org/>.

[7] A. Alves, et al., Web Services Business Process Execution Language Version 2.0, OASIS, 2007.

[8] N. Kavantzas, et al., Web services choreography description language Version 1.0, W3C Candidate Recommendation, 2005.

[9] Jaeyoon Jung, Jonghun Park, Seung-Kyun Han, and Kangchan Lee, "An ECA-Based Framework for Decentralized Coordination of Ubiquitous Web Services", To Appear in Information and Software Technology

[10] Kangchan Lee, Wonsuk Lee, Jonghong Jeon, Seungyun Lee, Jonghun Park, "Event-driven Coordination Rule of Web Services enabled Devices in Uniquitous environments", W3C Ubiquitous Web Workshop, March 2006

[11] G. vonBulltzingsloewen, A. Koschel, P.C. Lockemann, H.-D. Walter, ECA functionality in a distributed environment, in: N.W. Paton(Ed.), Active Rules in Database Systems, Springer, 1999, pp. 147 - .175.

[12] M. Cilia, A. Buchmann, An active functionality service for e-business applications, ACM SIGMOD Record 31 (1) (2002) 24 - .30.

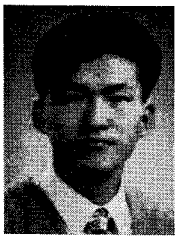
[13] U. Dayal, B. Blaustein, A.P. Buchmann, S. Chakravarthy, D. Goldhirsch, M. Hsu, R. Ladin, D. McCarthy, A. Rosenthal, The HiPAC project: combining active databases and timing constraints, ACM SIGMOD Record 17 (1) (1998) 51 - .70.

[14] D. Box, et al., Web Services Eventing (WS-Eventing), W3C Member Submission, 2006.

[15] M. Gudgin, et al., Web Services Addressing 1.0 - Core, W3C Recommendation, 2006.

[16] Yong Xiong, Johannes Helander, Alessandro Forin, Gideon Yuval, Secure Invisible Computing, Microsoft Technical Report, October 7, 2003.

◐ 저 자 소 개 ◑



이 원 석(Won-Suk Lee)

1996년 배재대학교 전자계산학과 졸업(학사)
1998년 충남대학교 대학원 컴퓨터공학과 졸업(석사)
1998년~2000년 교육부산하 한국교육학술정보원 연구원
2000년~2002년 해동정보통신㈜ 기술연구소 전임연구원
2003년~현재 한국전자통신연구원 표준연구센터 연구원
2005년~현재 W3C 대한민국 사무국 코디네이터
2006년~현재 TTA 지정 국제표준전문가
2006년~현재 ITU-T SG13 WP3 Q.2 에디터
관심분야 : 데이터베이스, XML, 웹서비스, 모바일 웹, 유비쿼터스 웹서비스
E-mail : wslee@etri.re.kr

박 종 헌(Jong-Hun Park)

1990년 서울대학교 산업공학과 졸업(학사)
1992년 서울대학교 대학원 산업공학과 졸업(석사)
2000년 Georgia Institute of Technology 산업 및 시스템공학과 졸업(박사)
현재 서울대학교 공과대학 산업공학과 교수
관심분야 : Internet Business, Mobile Services
E-mail : jonghun@snu.ac.kr

이 규 철(Kyu-Chul Lee)

1984년 서울대학교 컴퓨터공학과 (학사)
1986년 서울대학교 컴퓨터공학과 (석사)
1990년 서울대학교 컴퓨터공학과 (박사)
1994년 미국 IBM Almaden Research Center 초빙연구원
1996년 미국 Syracuse University, CASE Center 초빙 교수
1997년~1998년 학술진흥재단 부설 첨단학술정보센터 파견 교수
2001년~현재 한국정보과학회 논문편집위원
현재 한국 ebXML 전문위원회 위원장
현재 충남대학교 공과대학 컴퓨터공학과 교수
관심분야 : 데이터베이스, XML, 정보 통합, 멀티미디어 시스템, e-비즈니스 시스템, 유비쿼터스 웹 서비스
E-mail : kclee@cnu.ac.kr