공 학 석 사 학 위 논 문

다양한 RFID 태그의 지원을 위한 EPCglobal Reader Protocol 이벤트의 확장

2010년 8월

부 경 대 학 교 대 학 원

정 보 공 학 과

최 승 혁

공 학 석 사 학 위 논 문

다양한 RFID 태그의 지원을 위한 EPCglobal Reader Protocol 이벤트의 확장

지도교수 송 하 주

이 논문을 공학석사 학위논문으로 제출함.

2010년 8월

부 경 대 학 교 대 학 원

정 보 공 학 과

최 승 혁

최승혁의 공학석사 학위논문을 인준함.

2010년 8월 25일

주 심 공학박사 권 오 흠 (인)

위 원 공학박사 신 봉 기 (인)

위 원 이학박사 신 상 욱 (인)

목 차

표 차 례

PUKYONG NATIONAL UNIVERSITY 부경대학교

그림 차례

다양한 RFID 태그의 지원을 위한 EPCglobal Reader Protocol 이벤트의 확장

최 승 혁

부경대학교 대학원 정보공학과

요 약

RFID의 리더 관련 표준인 EPCglobal Reader Protocol (RP) 1.1의 태그 이벤트 모델은 판독된 태그식별자 정보만을 기준으로 한다. 따라서 태그식별자가 아닌 Class 1 Gen 2 태그의 사용자 메모리 데이터, 능동형 태그 데이터, 센서태그의 데이터 변화에 대해서는 이벤트 형식으로 처리 되지 않는다. 본 논문에서는 이와 같은 문제를 해결하고자 다양한 태그데이터가 이벤트 형식으로 처리되도록 RP 이벤트 모델을 확장하였다. 따라서 기존의 RP 이벤트 모델을 사용하는 응용프로그램을 크게 변경하지 않고서도 다양한 RFID 태그 데이터를 이벤트 형식으로 처리할 수 있다.

Extending EPCglobal Reader Protocol Events for Supporting Various Tag Events

Seung Hyuk Choi

Derpartment of Information Engineering, The Graduate School,
Pukyoung National University

Abstract

EPCglobal Reader Protocol (RP) 1.1 which defines the standard protocol between RFID readers and hosts provides a tag event model that deals with only scanned tag identifiers. Therefore the RP tag event model cannot support events associated with changes in tag data other than tag identifiers such as user memory data of class 1 generation 2 tags, active tag data, and sensor tag data. In this paper, we propose a new tag event model that supports various tag events by extending RP tag event model. Since the proposed event model is based on RP tag model, it can be used in existing RFID applications with little modifications.

1. 서 론

1.1. 연구의 필요성

전파를 이용한 무선인식(Remote Frequency Ientification, RFID) 기술의 응용범위가 무선식별이라는 응용분야에 집중됐으나, 최근에는 유저메모리(User Memory)를 이용하거나 혹은 여러 종류의 센서 태그에서의 센싱 값에 대한 응용이 점차 많아 지고 있다[1]. 예를 들어 이동하지 않고 일정장소에 계속 정지되어 있는 센서 태그의 경우 ,리더 내에서 동작하는 스무딩(Smoothing) 기능에 의해 태그가 새로 발견되었을 때는 이벤트(Event)로 보고되지만, 다시 이동하기 전에는 센서 값이 변경되더라도 이벤트가 발생하지 않게 된다. 또한 하나의 태그가 다수의 리더에 의해 읽히고 있는 상황에서 만약 특정 리더가 Class 1 Gen 2 Tag[2]의 메모리 값을 변경하면 나머지 리더는 태그 메모리가 변경된 사실을 인지하고 만약 연결된 응용프로그램이 있다면 메모리 값의 변경을 알려주어야 한다.

실제 세계에서 RFID 리더는 60~70%정도의 인식률을 보이는 것으로 알려져 있으며[3], 미세전파를 사용하여 통신하므로 각종 동작환경에 따라 태그의 읽기 동작이 불안정하게 이루어진다. 이에 EPCglobal에서 제시하고 있는 표준 Reader Protocol인 Reader Protocol 1.1[4](이하 RP)에서는 태그 이벤트를 리더수준에서 스무딩하는 이벤트 서브시스템(Event Subsystem)을 포함하고 있다. Event Subsystem은 태그의 존재 유무에 대한 이벤트를 발생시키고, 스무딩한다. 각각의 Read Cycle들이 완료될 때 마다 리더 서브시스템(Reader Subsystem)은 여러 번 같은 태그가 나타날 수 있으며, 이벤트 서브시스템은 이것을 이벤트라는 것으로 정제하여 데이터의 양을 줄이며, 매번 Read Cycle이 완료될 때 마다 이벤트를 발생시킨다. RFID 리더는 불안정하여, 읽힐 때가 있고 안 읽힐 때가 있는데, 이럴 경우, 원하지 않는 이벤트가 발생하게 된다. 이것을 보정하기 위해 스무딩을 하게 되며, 이 스무딩은 보통 인식되느냐 인식하지 모하느냐 얼마나 인식했느냐등의 태그의 인식여부에만 초점을 맞춘다.

이런 식의 이벤트 처리는 센서 태그의 센싱 값이나 혹은 클래스 1 젠2 태그에서의 유저 메모리에 대한 처리가 불가능하다. 이런 부가적인 정보가 계속 변화하면서 읽힌다면, 부가적인 정보의 변화가 되더라도 이벤트는 올라가지 않는다.
본 논문에서는 RP의 이벤트 모델을 확장하여 센싱값의 변화, 태그 메모리 값의 변경, 능동형 태그 이벤트의 처리를 통합적으로 수행할 수 있는 이벤트 모델을 제안하고자 한다.

1.2. 연구의 목적 및 방법

본 연구는 EPCglobal에서 제시하고 있는 표준 리더 프로토콜과 이벤트 모델에 대해 조사한다. 센서 태그 및 클래스 1 젠2 태그와 능동형 태그, 등에 있는 EPC가 아닌 부가적인 정보를 취득할 수 있고, 이 취득한 정보를 기반으로 이벤트를 일으킬 수 있는 일괄적인 이벤트 모델로 확장하였으며, 이에 대한 구현하였다. 이 이벤트의 메시지는 EPCglobal에서 제시하는 메시지 규격과 일치한다. 따라서 표준에 부합하는 메시지 형식을 통해 다양한 부가적인 정보를 넣을 수 있게 된다.

1.3. 논문의 구성

본 논문은 다음과 같은 내용으로 구성된다. 2장에서는 관련 연구를 하였고, 3장에서는 이벤트 확장 모델에 대한 제시, 4장에서는 이벤트 확장 모델을 구현을 하였으며, 5장에서는 적용했을 시에 시나리오에 대해 알아보며, 6장에서는 결론을 지었다.

2. 관련연구

2.1. RFID

RFID(Radio Frequency IDentification)는 어떤 사물에 무선 주파수 또는 자기장의 변화를 이용하는 전자장치를 부착하여 사물을 인식하거나 식별하는 시스템을 뜻한다[5]. RFID시스템은 바코드 시스템이나 마그네틱 시스템의 단점을 극복하기 위해 개발된 것으로 인식, 추적, 식별의 기능을 할 수 있는 기술이다[6]. RFID의 구성요소로는 태그, 리더, 미들웨어, 이를 상위 응용 프로그램 등으로 이루어 져 있다.

먼저 RFID 태그는 크게 안테나, 태그의 ID, 쓰고 읽고 할 수 있는 메모리를 포함한 장비이다. 리더로부터 전원을 공급받아 자신의 정보를 담아 다시 리더로 보내는 수동형 태그(Passive Tag)와 리더로부터 전원을 공급받지 않고 자신의 전원을 가지고 보내는 능동형 태그(Active Tag)로 나누어진다. 수동형 태그는 객체의 인식 쪽에 많이 쓰이며, 능동형 태그는 객체 인식뿐만 아니라, 센싱값이나 객체의 위치 등의 부가적인 정보를 취득하고자 할 때 많이 쓰인다.

리더는 비접촉하는 RFID태그에서 전파를 받아 호스트로 보내는 장치이다. 수동형 태그를 읽는 리더에서 전원을 태그에 공급하여 태그의 정보를 받아 온다. 하지만, 능동형 태그를 읽는 리더의 경우 태그에 전원을 공급하지 않고, 태그로부터 올라오는 정보를 받아서 처리한다.

RFID미들웨어는 리더로부터 읽힌 태그는 의미 있는 정보만을 걸러 응용 프로그램에 전달하는 소프트웨어이다. RFID리더로부터 수집된 데이터는 다양한 응용 프로그램에서 쓰기엔 방대하여, 필터링 및 스무딩, 그리고 그룹핑등을 거쳐 정제된 데이터를 만들어 내어, 서로 상이한 환경에서 원할한 통신을 하게 해준다.

2.2. Reader Protocol

RP는 EPCglobal에서 제안한 표준을 지키는 미들웨어와 태그를 읽고 쓸 수 있는 리더와 미들웨어사이의 프로토콜이다. 그림1과 같이 RP는 Reader Layer, Message Layer, Transport Layer로 이루어져 있다.

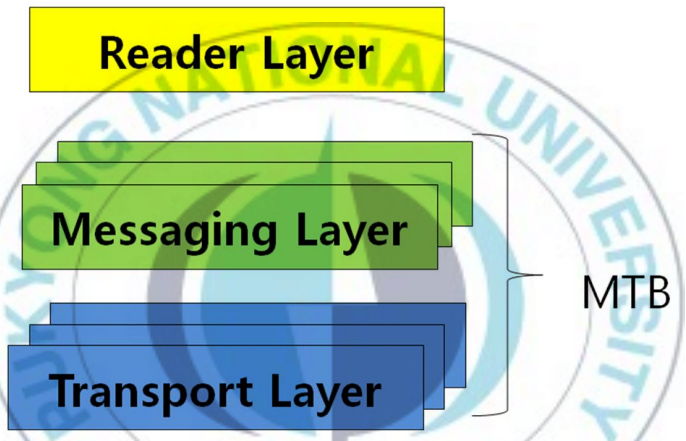

그림 1 RP의 계층

● Reader Layer

리더와 호스트 사이에서 교환하는 추상적인 메시지와 이 메시지에 대한 문법을 기술한다. 리더 프로토콜에서 핵심적인 부분으로서 리더가 어떻게 동작하고 무엇을 의미하는지를 정의한다.

● Message Layer

Reader Layer에 의해 정의한 추상적인 메시지를 실체하는 구조적인 메시지로 변환시킨다. 핸드쉐이킹 시 이 메시지 레이어가 결정되며, 최종적인 메시지가 만들어진다. (XML/TXT)

● Transport Layer

운영체제 혹은 동급에 종속적인 네트워크 기술을 이용하여 리더와 호스트 사이의 통신을 한다. (TCP, HTTP, UDP, Serial)

Message Layer와 Transport Layer를 합쳐 Messaging Transport Binding(MTB)라 부르며, 리더는 여러 개의 MTB를 가질 수 있으나, 연결 시 MTB를 정하고 기타 형식을 정하는 핸드쉐이킹(Hand Shaking)을 하고 난 이후에는 정해진 MTB만 쓸 수 있다. HTTP의 경우에는 비 연결 지향적인 프로토콜 특성 상, 호스트에서 명령을 보낼 때 마다 MTB를 바꿀 수 있다.

PUKYONG NATIONAL UNIVERSITY
부경대학교

다음의 그림2는 RP의 객체모델로서 ReaderDevice안에 여러 가지 RP관련 객체들이 들어갈 수 있다.

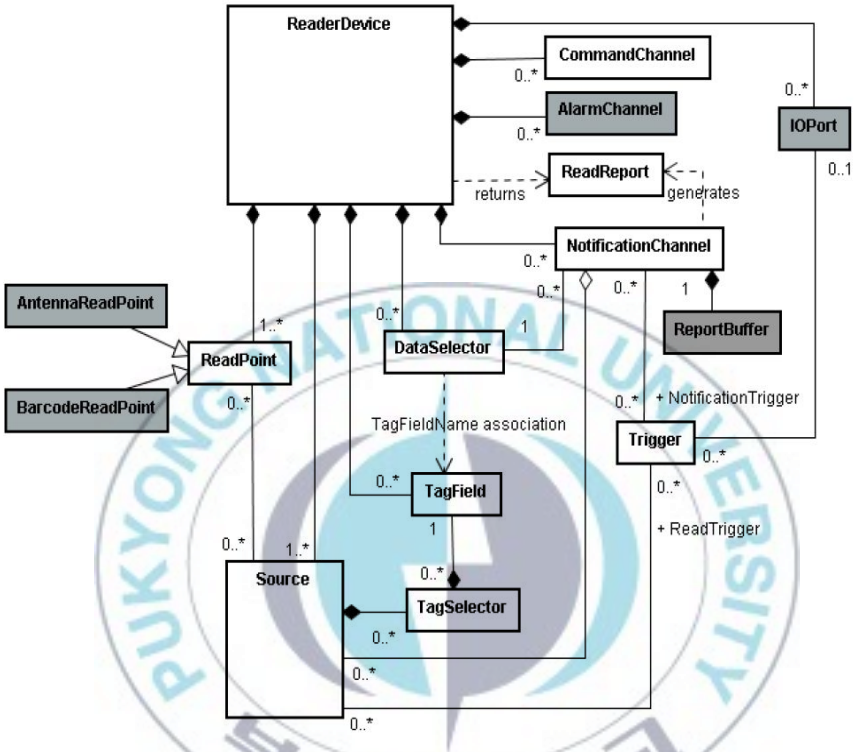

그림 2 RP의 객체 모델

ReaderDevice Object

리더디바이스(ReaderDevice) 는 RP에서 정의하는 객체의 컨테이너가 되는 객체로서 모든 RP에서 리더를 대표하는 객체이다. 리더디바이스는 적어도 한 개의 정해진 커맨드 채널(CommandChannel)이 필요하며 추가적으로 리더를 관리하는 필요하거나 혹은 네트워크 통신 시 필요한 속성과 객체를 가진다.

Sources

소스(Source) 객체는 리더디바이스객체와 노티피케이션 채널(Notification Channe) 객체와 연결되어 있으며, 태그 쓰기 및 읽거나, 이벤트 관련 인자를 관리

하는 등의 태그와 밀접한 관련이 되어있다.

Triggers

트리거(Trigger) 객체는 리드트리거(ReadTrigger)일 때는 소스와 관련된 태그를 계속 읽어 이벤트를 만드는 작업을 하고, 노티피케이션 트리거(Notification Trigger)일 때는 노티피케이션 채널(Notification Channel)과 관련된 호스트로의 리포팅 작업을 한다. 트리거의 상태가 Continuous일 때는 가능한 빠르게 처리해야하며, 리드트리거일 때는 쉬지 않고 리더가 태그를 계속 읽어야 하며, 노티피케이션 트리거일 때는 이벤트가 발생하면 바로 호스트로 전달하게 만든다.

트리거의 상태가 Timer일 때는 호스트에서 정한 일정한 인터벌로 리드트리거 혹은 노티피케이션 트리거가 동작하게 한다.

TagSelectors

태그셀렉터(TagSelector) 객체는 소스와 연관되며, 이 객체의 필터링 로직(Filterling logic)은 비트와이즈 패턴(bitwise pattern)에 기반하여 간단하게 이루어져 있다. 태그셀렉터는 태그필드와 연관될 수 있으며, 이 태그 필드는 태그셀렉터를 지원하는 데이터 필드를 정의한다. 리더는 Tag의 RF protocol의 singulation part를 실행하는 시간을 제거하기 위해 호스트에 의해 설정된 태그셀렉터를 쓴다.

Channels

채널(Channel) 객체는 리더와 호스트간의 통신을 유지하는 책임이 있다. 채널은 커맨드 채널(Command Channel)과 노티피케이션 채널로 나누어진다. 커맨드 채널과 노티피케이션 채널은 리더디바이스와 관련되어 있으며, 한 개의 채널은 특별한 MTB로 구현된다.

커맨드 채널은 호스트로부터 오는 메시지를 받는 것에 대한 책임이 있으며, 커맨드 채널의 접속은 호스트에 의해 시작된다.

노티피케이션 채널은 비동기적으로 호스트와 통신을 유지하는 책임이 있으며, 노티피케이션 채널은 커맨드채널의 요청으로 만들어지며, 이 요청에는 주소가 들

어가게 된다.

DataSelectors

데이터셀렉터(DataSelector) 객체는 노티피케이션과 관련되어있거나, 혹은 태그 읽기 명령의 인자로서 제공된다. 데이터셀렉터는 데이터필드 선택과 이벤트 필터링을 제공한다.

TagFields and TagFieldValues

태그 필드(TagField)는 태그의 특별한 메모리필드에 읽기나 쓰기를 하기위하여 정의된다. 태그필드는 메모리 레이아웃으로부터 상위 응용이 사용하는 논리적인 태그 필드 명을 대신 하기 위하여 물리적인 태그 추상적인 개념으로 사용되어진다. 태그필드밸류(TagFieldValue)는 메모리 관련 쓰기를 할 때 쓰인다.

2.3. RP의 이벤트 모델과 문제점

그림3은 RP에서의 이벤트 서브시스템에서 사용하는 이벤트 처리를 위한 상태도를 보인 것이다.

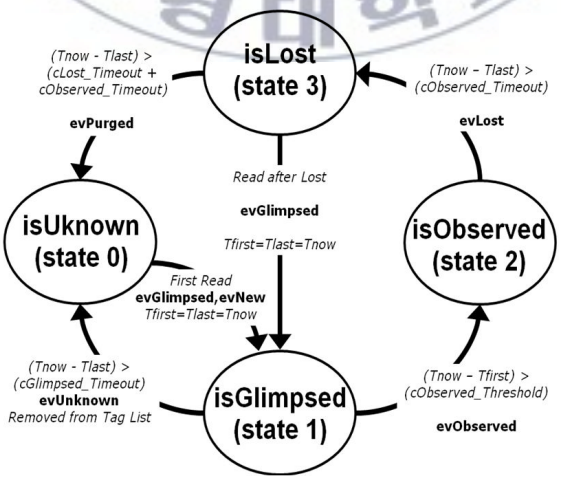

그림 3 이벤트 생명주기

RP에서 정한 이벤트는 evNew, evGlimpsed, evUnknown, evObserved, evLost, evPurged로 나누어진다. 다음 그림3과 같이 표시된 이벤트 생명 주기(Event Life cycle)에 의해 일어나게 된다.

태그는 안테나로부터 멀리 떨어져 있는 것에서부터 있는 것이 일반적인 초기 상태이다. 이 태그가 점점 다가와서 안테나로부터 처음으로 인식되었다면, evNew와 evGlimpsed가 같이 발생된다. 리더가 처음으로 태그를 인식하였기 때문에, 처음 읽힌 시각(Tfirst)와 마지막으로 읽힌 시각(Tlast)은 현재 읽힌 시각(Tnow)으로 입력되고 이벤트 서브시스템에서 유지하는 태그를 일으켜야 하는 목록에 추가된다. evNew이벤트는 처음 읽혔을 때 발생하는 이벤트이며, evGlimpsed 이벤트는 태그가 간헐적으로 읽히고 있음을 알리는 것으로서 발생한다.

태그의 상태가 isGlimpsed일 경우에는 간헐적으로 리더가 태그를 인식하고 있는 상태이다. 이 경우엔 안테나와의 교신상태가 안정적이라고 볼 수 없는 상태라고 볼 수 있다. isGlimpsed인 태그 중에 잘 안 읽혀 사용자가 설정한 cObserved_Threshold보다 안 읽힌 태그는 리더가 전파 왜곡으로 인해 전혀 다소 읽으면 안 되는 태그를 읽을 수 있기 때문이다. 어쩌다 한번 읽힌 태그의 이벤트를 발생시키는 것은 시스템의 자원에 부담을 줄 수 있다. 이런 이유로 사용자는 cGlimpsed_Timeout을 설정하여 isGlimpsed인 태그 중에서 읽히지 않는 태그를 제거하고 evUnknown이벤트가 발생시킨다.

간헐적으로 읽히는 태그(isGlimpsed)는 시간이 지나면서 안테나에 다가올 것이며 이 안테나에서 읽힐 것이다. cGlimpsed_Timeout보다 작은 시간동안 태그 안 읽혔을 경우에나, 리더가 isGlimpsed인 태그를 계속 읽어서 지금 읽힌 시각과 처음 읽힌 시각의 차가 cObserved_Threshold보다 클 경우, evObserved라는 이벤트가 발생한다. 이 이벤트가 발생했다는 것은 간헐적으로 읽히던 태그가 이제 리더 입장에선 안테나와 가까워져 안정적인 리딩을 하는 상태인 isObserved가 된다.

isObserved인 태그가 안테나와 멀어지기 시작하면, 점점 읽히는 빈도가 떨어진다. 이렇게 계속 멀어지면, 리더가 태그를 안 읽혀진 시간(Tnow - Tlast)이 cObserved_Timeout보다 더 커지면 evLost를 발생시키고, isLost로 태그의 상태는 전환된다. 이 evLost가 발생한 태그는 거의 읽히지 않을 상태일 수 있다. 이 상태

가 지속되어 evLost가 발생한 시간보다 cLost_Timeout이 더 안 인식하지 못하게 되면 이 evPurged를 발생한다.

evPurged는 더 이상 이벤트 서브시스템에서 유지하는 목록을 지우고 이벤트를 일으키지 않는다. evPurged가 발생하고 다시 읽히게 되면, 처음 읽혔던 대로 evNew와 evGlimpsed가 발생하여 다시 이벤트 서브시스템의 목록에 추가한다. 그리고 evLost일 때 다시 읽히면 일단 evGlimpsed이벤트를 발생시키고 isGlimpsed 상태로 변화된다.

이렇게 일어난 이벤트는 그림4와 같은 전이과정을 통해 호스트로 보고된다.

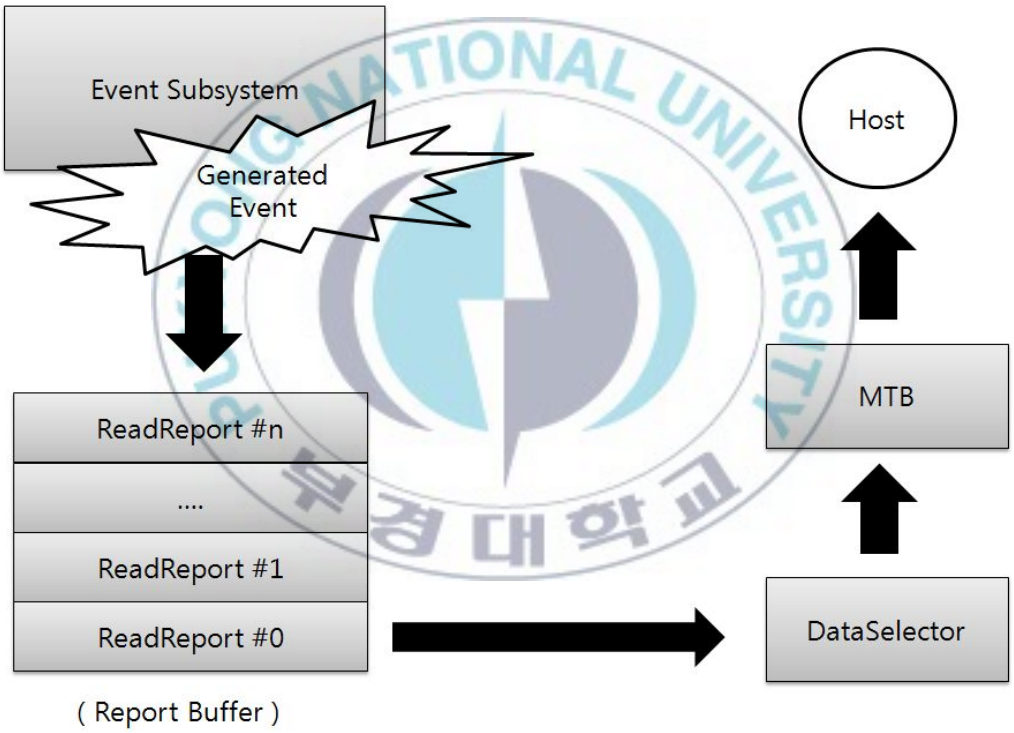

그림 4 이벤트가 일어나고 난 뒤 과정

이벤트 서브시스템에서 일어난 이벤트는 ReadReport형태로 Output Subsystem의 Report Buffer에 순서대로 쌓이게 된다. Notification Trigger에 의해 Report Buffer에서 Data Selector에서 지정된 필드나 태그 필드에 대한 추출을 한 뒤 커뮤니케이션 서브시스템(Communication Subsystem)에서 MTB를 거친다. 이

ReadReport는 호스트에서 접속 시 핸드쉐이킹(HandShaking)할 때 지정되어진 XML 혹은 Text로 만들어진다. 이후 만들어진 XML이나 TXT는 Message Layer, Transport Layer에서 처리하여 호스트로 알려진다.

이와 같이 태그 인식 여부에만 초점을 맞춘 이벤트 처리는 태그식별자 이외에 태그 메모리 또는 센싱값을 포함하는 태그를 사용하는 경우 이것을 알려줄 수 있는 적절한 이벤트가 생성되지 않으므로 문제를 야기할 수 있다. 그림 5과 그림6은 이러한 예를 보인 것이다.

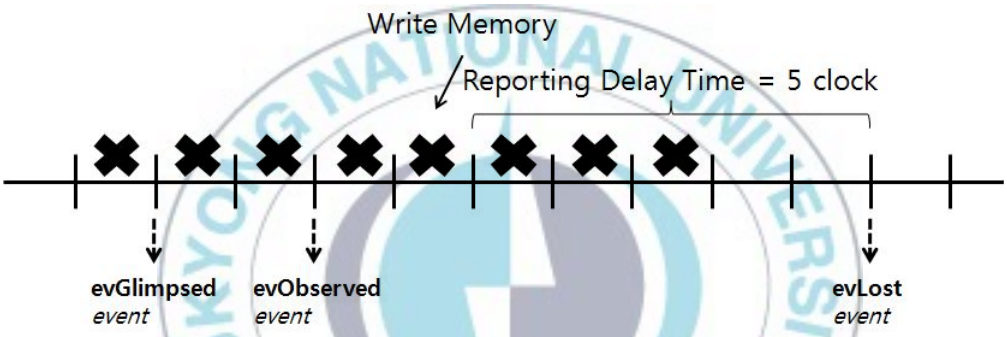

그림 5 Class 1 Gen 2 태그의 메모리 쓰기 시 이벤트

evGlimpsed 이벤트가 발생한 후, 계속 태그를 리더가 인식하여 evObserved가 발생하였다면, 이 후 태그를 계속 인식할 경우 더 이상 이벤트 서브시스템에서는 이벤트를 발생하지 않는다. 예를 들어, 그림3 과 같이 Class 1 Gen 2 Tag의 메모리 쓰기나 그림4와 같이 센서 태그를 이용하여 센싱 값을 측정하고자 할 경우 이벤트는 더 이상 일어나지 않는다.

그림 5의 경우 evObserved 발생한 이후에 호스트는 이제 태그가 안테나에 근접하였다는 것으로 판단하고, 이 태그의 유저메모리에 Write를 하였다. 하지만, 호스트에 태그의 유저메모리가 변경되었음을 알리는 메시지가 오는 것은 태그가 사라지고 난 이후, 즉 evLost를 통해 전달된다. 따라서 그 동안에는 메모리에 변경에 대한 보고가 되지 않는 지연시간이 발생한다. 이 지연시간은 evLost가 발생될 때까지이며, 그 기간 동안의 메모리 쓰기는 마지막 값으로 대체되어진다. 하지만 언

제 evLost가 될지도 모르고, 이런 지연시간으로 인해 호스트가 메모리 작업을 하는데 있어서 제약이다.

호스트가 즉각적으로 알아보기 위하여 메모리쓰기를 하고 난 뒤에 읽기 명령을 통해 업데이트를 받을 수 도 있으나, 상이한 두 개의 데이터가 올라가는 경로가 만들어지므로 일관성이 없고, 호스트에서는 서로 다른 두 개의 경로를 거쳐 전달되는 데이터를 받아 처리하여야 한다.

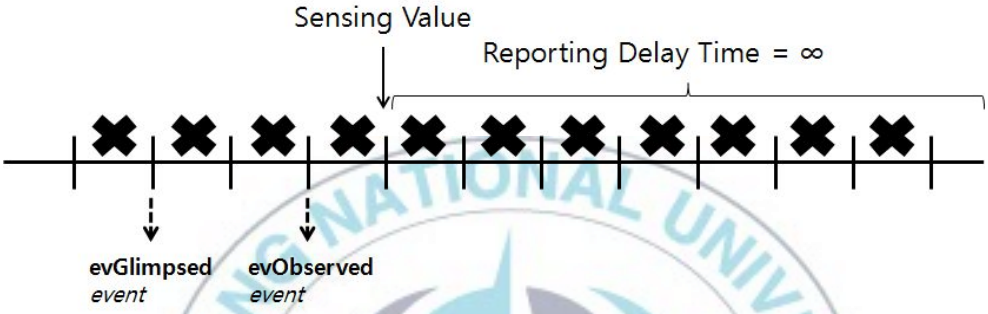

그림 6 센서 태그의 센싱 시 이벤트

그림 6의 경우에는 evLost일 때 올라왔었던 그림4의 문제보다 더 심각하다.

처음 evObserved가 일어난 후에 변화된 센싱값이 태그로부터 리더가 읽었음에도 불구하고, 더 이상 태그가 계속 인식하는 한 이벤트는 더 이상 보고되지 않는다. 즉, 태그인식이 계속되고 있는 동안에는 센싱값이 전달되지 않을 수 있다. 호스트에서 계속 리드 명령을 리더로 보내 값을 가져갈 수 있으나, 호스트에서 RP의 이벤트모델을 쓰지 않고 호스트내부에서 이벤트 처리해야 한다.

2.4. 기존연구의 문제점

RFID이벤트에 관련된 연구로 먼저 Adaptive Cleaning for RFID Data Stream[3]을 들 수 있는데, [3]은 태그판별을 위한 고정된 시간 윈도우 사이즈를 가지는 것이 아니라, 판별의 정확성을 위해 '올바른' 윈도우 사이즈를 자동적이며 연속적으로 찾는다. 하지만, 고정된 윈도우 사이즈보다 적합한 윈도우 사이즈를 계속

RFID Data에서 찾아야 하는 오버헤드와 센싱값 변동에 대한 이벤트처리가 없다.

그리고 PEEX[7]는 반복적으로 읽힌 것 중에 확률적인 판단에 따라 이벤트 발생에 대한 연구이다. PEEX에 경우, Low 이벤트와 High 이벤트로 나누어서 처리하는데, Low 이벤트는 리더에서 태그가 읽히고 안 읽히는 것 대한 이벤트이며 High 이벤트는 실제 응용에서 의미 있는 이벤트라고 정의하는데, 많은 Low 이벤트를 따라 확률에 근거한 판단을 통해 High 이벤트로 변경한다. 하지만, 식별에 대한 판단만 있을 뿐, 유저메모리나 혹은 센싱값에 대한 고려가 없다.

마지막으로 ESN[8]에 경우, USN과 RFID를 통합하기 위한 연구이다. RFID와 USN간의 차이로 인해 이벤트를 RFID 이벤트와 Sensor 이벤트로 나뉘어 처리한다. RFID 이벤트에 경우 ID, Location, Timestamp로 이루어져 있고, Sensor 이벤트에 경우 ID, Location, Timestamp, Sensor Data로 이루어져 있고, 추가적으로 Life Time을 정할 수 있다. 하지만, 이렇게 두 가지의 이벤트를 따로 처리함으로서 일관적인 구조를 가질 수 없다.

[3][7][8]모두 능동형 리더나 Class 1 Gen 2 태그의 유저메모리 값 변경에 대한 이벤트 처리를 하는데 부적합 구조이다. [3][7]에 경우 태그 존재 여부의 이벤트 처리를 위한 효율적인 처리 방안을 제시하였을 뿐, 능동형 리더나 Class 1 Gen 2 태그의 메모리 처리에 대한 고려가 없었다. [8]은 두 가지의 서로 다른 RFID이벤트와 Sensor 이벤트로 처리하여 일관적인 접근이 힘들다.

종합하면, 현재까지 제안된 각종 이벤트 모델은 태그식별자에 대한 인식만을 기준으로 한 것이므로 유저메모리 또는 센싱값 등의 변경을 고려한 이벤트 모델을 제안한 연구는 전무하다. 따라서 현실적으로는 리더 수준의 이벤트 처리 기능사용하지 않고, 리더에서 판독된 모든 태그정보를 RFID 미들웨어(호스트)에 전달한 다음, 응용프로그램 수준에서 태그메모리 및 센싱값을 처리하는 경우가 많다. 이에 본 연구에서는 리더내의 RP의 이벤트 시스템을 확장하여 이러한 기능을 지원하고자 한다. 제안하는 확장 이벤트 모델을 적용 리더와 미들웨어사이의 데이터 전송량을 줄일 수 있고 미들웨어의 부하를 감소시키며, 유저메모리에 대한 쓰기를 미들웨어 수준에서 효율적으로 수행할 수 있게 할 수 있다.

3. 이벤트 서브시스템의 확장과 구현

3.1. 이벤트 서브시스템의 확장

RP의 Object Model에서 제시된 리더디바이스는 전반적으로 RP에서 쓰는 객체를 관리하는 객체이다. 리더디바이스는 RP에서 생성 시 필요한 인자들을 받는다. 이 인자 중에 이벤트 서브시스템 확장에 관련된 인자를 넣어 생성한다. 그리고 이벤트 주체인 소스가 만들어질 때 리더디바이스로부터 인자를 받아 이벤트 서브시스템 을 생성한다.

인자가 RPStandard이면 위와 같은 evNew와 같이 RP에서 제시한 이벤트만 발생시킨다. RPExtension은 일 경우 RP에 맞는 이벤트 처리 이외에 isObserved에서 태그의 가 수정되었을 때 이벤트가 발생한다. RPExtension은 필드를 지정하며, 이 필드가 변경되었다면 이벤트를 발생시키며 사용자가 선택하지 않은 필드에서의 데이터 변경에 대한 이벤트 생성을 막는다.

그리고 태그 데이터가 지속적으로 올라오긴 하지만 벤더필드 혹은 태그필드가 잦은 변경 혹은 이미 이벤트 처리가 된 것으로부터 오는 메시지를 처리하기 위해 NoSmooth처리도 존재해야 한다. 이것은 이벤트 처리가 무의미하거나 이미 처리한 것이므로 이벤트 서브시스템을 거치지 않고 아우풋 서브시스템(Output Subsystem)으로 가게 된다. 그리고 설정파일에서 설정된 리더디바이스의 이벤트 서브시스템관련 설정은 표1과 같은 이벤트를 발생시켜, 호스트로 보내어질 수 있다.

표 1 리더디바이스 설정에 따른 대표적인 이벤트

설정	대표적인 이벤트
RPStandard	evObserved
RPExtension	LIT:evValue
noSmooth	LIT:Alarm

리드트리거로 읽혀진 태그들은 이벤트 서브시스템을 처리하는데, 표준모델에선 읽혀진 태그 데이터 갱신 이후 이벤트가 발생되어야 한다. 하지만 확장 모델에선 리드트리거로 발생 시 태그 데이터는 표2와 같이 처리된다.

표 2 RP확장모델의 의사코드

```
Read TagData at Physical Tag
Merge TagData
Select TagData
IF EventSub is Smooth THEN
  IF EventSub is RPExtension THEN
    Generate RP Extension;
  ENDIF     TagData Update EventSub
  Generate RP Standard Event
ELSE
   Generate AlarmEvent.
ENDIF
```

실제 물리적 태그라면 자신을 구분해주는 EPC와 태그 읽힌 안테나를 ReadPoint는 가져와야 한다. 태그필드나 혹은 벤더필드를 사용하여 메모리 값이나 혹은 센싱값에 대한 호스트로 보내려면, 센서태그일 경우 센싱값을 가져와야 한다. 그리고 Class 1 Gen2 태그일 경우 메모리 값을 태그 읽을 때 시에 같이 해당하는 메모리 값을 그림7과 같이 되어 있는데, 메모리 값을 모두 가져와야 한다.

그림7과 같이 Class 1 Gen 2 Tag의 논리적인 메모리 구조는 Reserved 메모리, EPC 메모리, TID 메모리, 유저 메모리로 이루어져 있다.

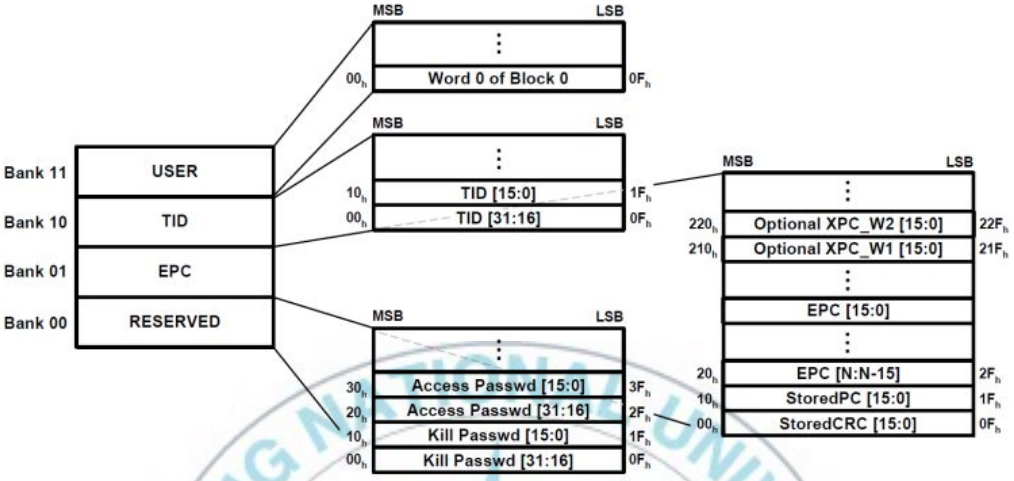

그림 7 Class 1 Gen 2의 Logical Memory Map

Reserved Memory는 Kill 및 Access 패스워드로 되어 있고, EPC 메모리는 CRC값과 EPC값이 들어가야 한다. EPC값은 Tag Data Standard[9]를 준수하거나, ISO/IEC 15961[10]을 준수해야 한다. TID 메모리는 태그를 만드는 제조사에서 발급하는 물리적인 태그에 대한 고유한 숫자(Serial Number)를 저장하는 곳이며, 유저 메모리는 EPC를 제외한 물리적 태그에 추가적으로 저장해서 읽고 싶을 경우 쓰는 곳으로 User specific한 구조로 저장할 수 있다.

이렇게 올라온 TagData는 Bitwise패턴에 기초한 태그셀렉터를 지나 정제된 TagData가 되어진다.

최종적으로 정제된 TagData는 이벤트를 일으킬 수 있는 기준이 된다. 만약 태그가 없어 TagData를 생성 못하였거나 혹은 타임아웃이 되었을 시엔 현재 기준 시각으로 넣어 이벤트를 발생시킬 수 있도록 하여야 한다.

이벤트 서브시스템이 RPStandard으로 설정될 경우 RP의 표준에 맞는 이벤트 모델의 태그 인식만으로 이벤트를 발생시킨다. RP의 이벤트 모델은 태그의 상태와 태그가 리더에 처음 및 마지막에 읽힌 시간에 의해 결정된다. 그리고 현재의 읽힌 데이터가 갱신되고 난 이후, 이벤트가 발생한다.

다음 표3은 본 논문에서 제시하는 RP일 때 하는 이벤트 의사코드이다.

표 3 RPStandard일 때 이벤트 의사 코드

```
Set Event Reference Time
cPurged_Timeout=cObservered_Timeout+cLost_Timeout
For each Tag of the Tag Data
   Get Event Tag at Event Subsystem's TagList
   IF Event Tag is Existed Then
      Set Event Tag's Last Time to Event Reference Time
   ELSE
      Create Event Tag
      Set both Event Tag's First and Last Time to Event Reference Time
   Set Event Tag's State to IsUnknown
   Add TagList
   ENDIF
ENDFOR
Init RemoveTagList
FOR each Event Tag Information of TagList
    CASE Event Tag's Status OF
       IsUnknown:
          Generate evNew and evGlimpsed
          Set Tag's State to IsGlimpsed
       IsGlimpsed:
          IF Event Reference Time - LastTime > cGlimpsed_Timeout Then
            Generate Unknown Event
            Add Tag At RemoveTagList
          ELSE IF Event Reference Time - FirstTime > cObserved_Threshold Then
            Generate evObserved
            Set Tag's State to IsObserved
          ENDIF
       IsObserved:
          IF Event Reference Time - LastTime > cObserved_Timeout Then
             Generate evLost
```

```
            Set Tag's State to IsLost
        ENDIF
    IsLost:
        IF Event Reference Time - LastTime > cPurged_Timeout Then
            Generate evPurged
            Add Tag At RemoveTagList
        ELSEIF now = LastTime Then
            Generate evGlimpsed
            Set Tag's FirstTime to Event Reference Time
            Set Tag's State to isGlimpsed
ENDCASE
for each RemoveTag of RemoveTagList
    Remove removeTag At TagList
```

이벤트 서브시스템의 설정이 RPStandard일 경우 처리절차는 2.3에서 설명한 RP의 이벤트 모델과 비슷하다. 가장 처음 처리 하는 것이 태그가 읽힌 시간으로 이벤트를 발생시킬 이벤트 서브시스템의 기준 시각(Event Reference Time)을 정하는 것이다. 현재 리드트리거에 의해 읽혀진 태그가 있다면, 읽혀진 시각을 기준 시각으로 정해지며, 태그가 전혀 읽혀지지 않았다면 현재 시각(now)로 정한다.

이렇게 정해지고 나면, 스펙에서는 나오지 않는 Purged_Timeout을 Observed_Timeout + Lost_Timeout을 통해 구한다. 이것은 어차피 evPurged를 할 때는 Observed_Timout + Lost_Timeout만큼 안 보여야 하므로, 먼저 구하고 들어가 연산을 조금이라도 줄인다.

이벤트 서브시스템이 RP에서 지정한 이벤트를 발생시키기 위해선, 이벤트 서브시스템의 태그리스트(TagList)를 최신의 상태로 유지시켜야 한다. 리더에서 읽힌 올라온 개개의 태그 정보로 이벤트 서브시스템에서 유지하는 태그리스트의 태그들의 마지막에 읽힌 시간을 이벤트 기준 시간으로 갱신한다. 이벤트 서브시스템의 태그리스트에서 찾지 못 하는 태그들은 새롭게 읽힌 태그이며, 이 태그들을 보관하는 리스트에 추가하고 처음 읽힌 시각과 마지막에 읽힌 시각을 이벤트 기준 시각으로 갱신한다. 처음의 이렇게 읽혀진 태그데이터들이 이벤트 서브시스템의 태그리스트에 갱신되고 난 뒤에 이벤트 서브시스템의 태그리스트를 순회하면서 RP에서 지정한 이벤트를 발생시킨다. 태그리스트를 순회할 시에 태그의 상태와 처음

읽힌 시각과 마지막으로 읽힌 시각, 이벤트 기준 시간을 이용하여 발생시킨다.

이후는 앞서 설명한 그림 2와 비슷하여 생략하며, 다른 것은 evUnkwon이나 evPurged시에 isUnknown으로 변경하지 않고 지울 목록에 추가하고 이 목록을 통해 일괄적으로 이벤트 서브시스템내의 태그리스트에 있는 지워져야 할 태그를 지워야 한다.

이벤트 서브시스템이 RPExtension일 경우 RP의 이벤트와 값 비교 이벤트인 LIT:evValue가 발생하여야 한다. LIT:evValue 이벤트는 전에 이전 데이터와의 값 비교를 통해 이벤트를 발생시켜야 하므로, 이벤트 서브시스템의 유지하는 태그리스트를 갱신하지 않고 이 태그리스트와 비교하면 일으킨다. 다음 표4는 이벤트 서브시스템이 RPExtension일 때 값 비교를 통해 이벤트 발생하는 것을 기술하였다.

표 4 LIT:evValue를 일으키기 위한 Pseudo Code

```
IF Tag is Existed THEN
  FOR each tag of TagData
    storedTagData = getStoredTag(tag)
    IF storedTagData is Existed THEN
      IF storedTagData is Observed THEN
        FOR each tagfiled of chooseTagFields
          ACCESSS Memory through TagField
          IF storedTagDataMemory NOT Equals ReadTagDataMemory THEN
            Generated LIT:eValue
          ENDIF
        ENDFOR
        IF IsNot Generated LIT:evValue Event THEN
          FOR each vendorField of chooseVendorFields
            IF storedTagDataVendorValue is NOT Equals
              ReadTagDataVendorValue THEN
              Generated LIT:evValue
            ENDIF
          ENDFOR
        ENDIF
IF LIT:evValue is generated THEN
```

```
Send LIT:evValue to Notification Channel
          ENDIF
        ENDIF
      ENDIF
   ENDFOR
ENDIF
```

LIT:evValue를 원하는 사용자는 RPExtension로 리더디바이스에서 설정한 후 태그필드나 벤더필드를 설정한다. 그리고 이러한 태그필드와 벤더필드는 각각 단수 또는 복수가 될 수 있으며, 복수의 태그필드와 복수의 벤더필드를 같이 설정할 수 있다. 이후 값 비교를 통해 LIT:evValue를 발생시킬 수 있다. 그리고 값 비교 시 복수의 벤더 필드 혹은 태그 필드를 선택하였다면, 복수의 벤더 필드 혹은 태그 필드에서 한 개라도 변경이 되었다는 것이 확인되면, 더 이상 값 비교를 하지 않고, LIT:evValue를 발생시킨다.

LIT:evValue는 RP의 이벤트 모델에서 태그가 계속 읽혀지고 있다는 것으로 여기는 상태가 isObserved로 되어있을 때, 이 상태에서 센서 태그에서 측정된 센싱 값이나 클래스 1 젠2 태그의 메모리 값이 변경되었다면 이벤트가 일어나야 하는데, RP에서 제시하는 이벤트 모델로는 일으키지 못하므로, ReaderDevice를 최초 생성할 시에 설정된 태그의 태그필드나 혹은 벤더 필드의 값이 변경되었을 시 이벤트를 발생시킨다. 선택된 필드가 태그필드일 경우 바이너리 타입인 태그필드를 위해 비트단위의 값 비교를 해야 하며, 태그필드에서 정의하고 있는 뱅크, 오프셋, 길이를 통해 값을 추출해야 한다. 그 값을 비교하여 LIT:evValue를 일으킨다. 벤더 필드의 경우 문자열 비교만으로도 충분히 값을 비교할 수 있기 때문에 비트 연산은 하지 않고, 벤더 필드에 담겨 있는 값 또한 문자열 비교 연산을 통해 값이 변경되었다면 이벤트가 발생된다.

그림 8,9는 각각 LIT:evValue가 발생했을 발생할 수 있다.

그림 8은 확장된 이벤트 모델을 통해 지연시간 없이 메모리 변경 시, 이벤트가 발생할 수 있다.

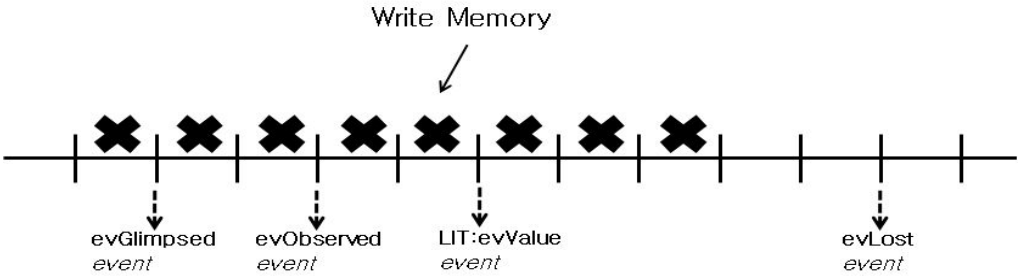

그림 8 Class 1 Gen 2에서의 메모리 변경에 대한 LIT:evValue발생의 예

이를 통해 태그가 안테나로부터 떨어졌을 때 발생하는 evLost가 발생하기 전에 호스트에선 태그의 메모리 값이 변경되었음을 알 수 있다. 즉 다시 말해 태그가 안테나로부터 멀리 떨어지기 전에, 태그의 메모리 변경을 했을 시 알 수 있다.

그림9는 확장된 이벤트 모델을 통해 evObserved이후 이벤트가 발생하지 않아 센싱값이 올라가지 않았던 센서 태그의 문제점을 LIT:evValue를 통해 해결한 예상도이다.

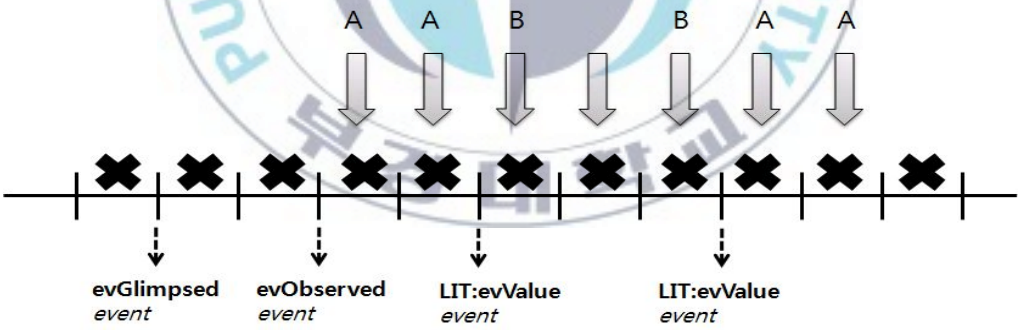

그림 9 센서태그에서의 LIT:evValue발생의 예

evObserved이후 값 A가 2번 읽혀 이 값은 계속 Observed일때와 다르지 않으므로, 발생하지 않는다. 하지만 3번째 리드 사이클에 읽혀진 값B는 evObserved일 때 발생된 값과 다르므로, 이벤트가 발생되어야 한다. 그리고, 다음번에 리드 사이클에 의해 읽혀진 값이 없으므로, 변경이 되었는지 않되었는지 이것에 대한 명확한 판단근거가 없으므로, 이벤트를 발생시키지 않는다. 이후 값B가 읽혔기 때문에 이전 읽혀졌던 값B와 같기 때문에 이벤트가 발생하지 않는다. 다음번 리드 사이클

에서 값A로 읽힌다면, 이벤트는 센싱값이 변경었으므로 발생하게 된다.

그림10처럼 실제 물리적 리더 혹은 태그에서 이벤트가 이미 처리되어 해당 데이터는 태그의 마지막 읽힌 시간과 태그의 처음 읽힌 시간으로만 비교하는 RP의 이벤트 모델로 처리하는 것은 다소 맞지 않다. 혹은 호스트 측에서 RP의 이벤트 모델에 따라 스무딩되지 않은 Raw Data를 원할 때, 호스트는 LIT:Alarm을 통해 받을 수 있다.

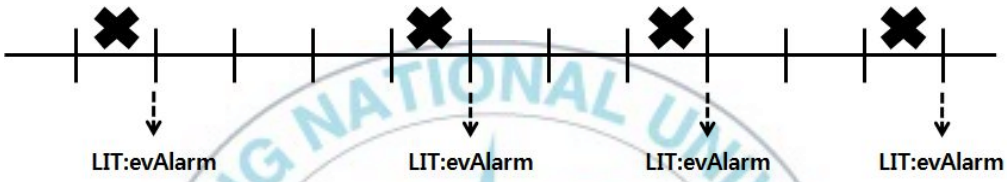

그림 10 LIT:evAlarm이 일어났을 경우

그림 10과 같이 능동형 태그나 센서 태그에 경우 일정 조건에 맞으면 태그와 값을 같이 보낼 수 있다. 이 경우 RP에 경우, evNew, evGlimpsed가 일어났다가, evUnknown이 계속해서 반복적으로 일어날 것이다. 이 경우에 LIT:evAlarm으로 이러한 오버헤드 없이 구현할 수 있다.

RP의 태그 인식만으로 하는 이벤트 모델을 LIT:evValue와 LIT:evAlarm을 쓰는 모델로 확장하여 센서 태그 및 능동형 태그, 메모리를 쓰고 읽고 할 수 있는 Class1 Gen 2 태그들의 값 변화에 맞춰 이벤트를 확장하였다.

3.2. RP지원 리더 에뮬레이터의 구현

실제 구현은 인증 받지 않은 실제 물리적 리더를 RP로 지원하게 보여주는 RP 리더 에뮬레이터를 구현하면서 진행하였다. 실험에 쓰인 실제 물리적인 리더는 Alien사의 ALR 9800[11]이다. 이 RP 리더 에뮬레이터는 리더디바이스를 생성하면서 설정파일로부터 리더디바이스 객체를 초기화하기 위한 인자를 받게 되며 이 인자를 통해 리더디바이스를 초기화하며, 여러 개의 부수적인 객체들이 만들어진다. 이 부수적인 객체를 만들 때 Source도 만들어지며, 이 Source에서 EventSub를 만든다. 이 EventSub는 이벤트 서브시스템을 구현한 객체이다.

그림11은 RP 리더 에뮬레이터의 전체적인 모습이다.

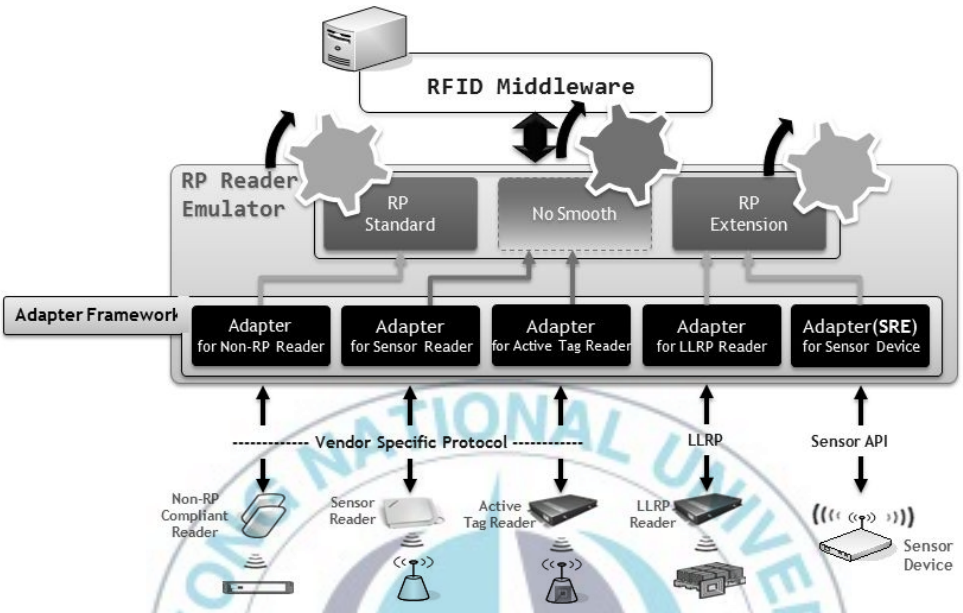

그림 11 RP 리더 에뮬레이터의 개괄도

RP 리더에뮬레이터는 Alien 9800과 같은 RP를 지원하지 않는 리더를 RP리더 에뮬레이터가 가지는 RP 객체들을 통해 RP를 지원하는 리더처럼 보이게 해준다. 그리고 쉽게 확장할 수 있는 Adapter Framework를 가지고 있다. Adapter Framework은 물리적 리더와의 상호 인터페이스를 하는 CustomReaderDevice가 있으며, CustomReaderDevice는 상속받아 벤더에 맞게 각각 구현한다. 이 CustomReaderDevice는 RP객체인 리더디바이스와 일 대 일 매칭된다. RP와 물리적 리더간의 상호간의 인터페이스를 하는 CustomReaderDevice는 각각의 벤더에 맞게 구현한다. 이 CustomReaderDevice는 각각의 리더디바이스와는 1:1로 매칭되며 CustomReaderDevice는 CustomTagData라는 통일된 자료구조를 가진다.

CustomTagData는 태그와 그이외의 태그별로 저장하는 구조인 CustomTagInfo를 가지며, CustomTagInfo는 표5와 같이 가진다.

표 5 CustomTagInfo의 구조

CustomTagInfo(Type)	실제 값
TagID(string)	EPC
ReadPoint(string)	ReadPoint
VendorFields(Map<String,String>)	Vendor Values
TagFields(Map<String,String>)	Tag Memory

TagID에는 상위응용에서 태그를 구분하기 위해 쓰일 EPC가 저장되며, 상위로 보내기 전까지 RP에선 계속 문자열로 취급되기 때문에 TagID는 String으로 저장된다. ReadPoint는 태그가 읽혀진 안테나로 안테나에 할당된 문자열로써, 태그가 읽혔다면 올라온다. VendorFields는 벤더필드의 이름과 그것에 해당하는 값들의 쌍이 들어가며, 문자열형식으로 들어간다. 태그로 부터 오는 값은 정수 혹은 실수로 올 수 있으나, 이것을 문자열화하여 보내어진다. TagFields는 그림2과 같은 구조로 되어 있는 클래스 1 젠2 태그에 경우, Bank0, Bank1, Bank2, Bank3에다 각각 뱅크에 맞는 전체 값이 16진수 문자열으로 매치 된다. 태그의 정보가 다 들어간 CustomInfo와 EventSub에서 쓸 태그가 읽힌 시간이 저장되어 CustomTagData를 이루어진다.

물리적 리더에 대해 태그를 읽으라는 명령이 내려가면 세팅되어진 뱅크마다 가지는 메모리를 읽을 수 있도록 명령이 내려가야 한다. 이렇게 하기위해 Vendor Protocol에서 지정된 명령이 있어야한다. 실험에 쓰인 ALR-9800리더는 태그가 읽혀졌을 때 미들웨어로 전송하는 포맷을 지정하는 명령인 TagListCustomFormat과 포맷을 지정할 때 메모리 값을 반환할 수 있게 뱅크, 오프셋, 길이를 정해주는 명령인 AcqG2TagData를 통해 정할 수 있어 반환할 수 있으며 이때 오프셋 및 길이를 0으로 주면 해당 뱅크의 전체 메모리 값이 반환 된다.

물리적 리더에서 네트워크 혹은 리더 자체의 문제로 인하여 사용자가 세팅한 타임아웃 시간보다 훨씬 늦게 태그가 읽혀 커스텀리더디바이스에 들어올 수 있다.

그림 12과 같이, 일정 시간마다 트리거를 통해 발동해야 하는 타이머 모드일 경우 일정시간은 지켜져야 한다.

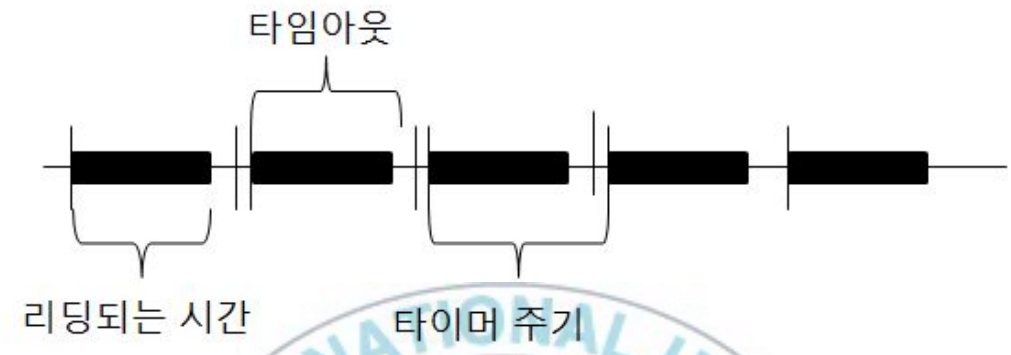

그림 12 타이머 주기와 타임아웃

이 주기를 벗어나 들어오는 타임아웃에 걸릴 경우, 사용자는 타임아웃시간과 리드트리거의 주기를 증가시켜 주거나 물리적 리더의 설정을 바꿔야 한다.

그림 13 타임아웃 발생

그림13과 같은 타임아웃을 처리하기위해 Source내부에서는 커스텀리더디바이스에서 리딩하는 부분과 이벤트를 처리하는 부분을 다른 쓰레드로 나누어 처리하였다. 타임아웃이 발생하여도 피지컬 리더에 오는 메시지를 무시 하지 말고 받아야 한다. 그러기 위해 리드 트리거에 의해 커스텀 리더디바이스을 통해 물리적 리더에 태그를 가져 오기 전에 쓰레드를 만들어 보낸다. 쓰레드를 통해 커스텀리더디바이스는 태그 읽기 명령을 수행하며, 이 수행하는 중에는 다시 읽기 명령을 나면 무시한다. 수행 중에는 읽기 명령을 무시하는 이유는 주기적으로 리드트리거는 읽기 명령을 커스텀 리더 디바이스에 보내는데, 이 CustomReaderDevice는 소스나 리더디바이스로부터 오는 트랙잭션을 수행하기 위해 실제 물리적 리더에 내리기 전에 락을 거는데, 계속적으로 타임아웃이 발생하는 상황이라면 읽기 명령은 메모리에 계속 쌓일 것이다. 이렇게 되면 RP의 주기적으로 읽지 않고, 바로바로 동작

하는 Continuos모드처럼 물리적 장치는 동작할 것이다.

물리적 리더로부터 CustoTagInfo를 구성할 수 있는 문자열을 받으면 파싱한다. 이렇게 해서 올라오는 CustomTagData를 큐에 넣는다. 이 큐에 대해 Source단에서 Timeout을 걸고 리드트리거의 원래 쓰레드는 값을 가져온다.
물리적 리더에 리딩하는 쓰레드는 계속 만들어질 것이므로, 소스에 고정된 한 개의 쓰레드만 가지는 쓰레드풀을 두어 쓰레드 생성 및 파괴로 인한 자원 손실을 줄일 수 있다.

이렇게 큐로부터 CustomTagData를 가져온 뒤, TagSelector에서는 이벤트가 필요 없는 태그가 비트와이wm 패턴으로 제거되어진다. 이렇게 선택 되어진 태그는 EventSub에 들어가 이벤트처리를 하게 된다.

EventSub은 RP에서 제시된 태그의 처음 읽힌 시각과 마지막으로 읽힌 시간으로 찾는 이벤트를 일으켜서 Report Buffer로 보낼 것인지, RP에 추가적으로 덧붙여 CustomTagData내의 벤더영역과 메모리영역의 비교를 통해 이벤트를 보낼 것인지, 스무딩 없이 바로 Report Buffer로 보낼 것인지가 정해진다. 만약 RPExtension일 경우 EventSub가 만들어질 때 설정파일에 설정해놓은 태그필드리스트와 벤더필드리스트를 보내어 진다. 모든 이벤트들은 Report Buffer에 쌓이고 난 뒤에 노티피케이션 채널에 의해 호스트로 전달된다.
EventSub는 확장된 이벤트를 일으켜서, RP에서 나온 태그의 처음 읽힌 시각과 마지막으로 읽힌 시간으로 찾는 이벤트를 일으켜서 Report Buffer로 보낼 것인지, 스무딩없이 바로 Report Buffer로 보낼 것인지가 정해진다. 만약 RPExtension일 경우 Event Sub가 만들어질 때 설정파일에 설정해놓은 태그필드리스트와 벤더필드리스트를 보내어 진다. 모든 이벤트는 Report Buffer에 쌓이고 난 뒤에 노티피케이션 트리커에 의해 호스트로 전달된다.

표6은 해당 미들웨어에서 설정하는 XML파일의 일부이다.

표 6 설정파일의 예

```
<ConfigReaderData>
<RPManager>
<RMIServerPort>9097</RMIServerPort>
</RPManager>
  <ReaderData>
   <ReaderName>Alien9800</ReaderName>
   <CustomReaderClass>
<ClassName>
Alien9800ReaderDevice
</ClassName>
<CustomReaderAddress>
210.125.112.100
</CustomReaderAddress>
<CustomReaderCommandPort>
23
</CustomReaderCommandPort>
</CustomReaderClass>
      <UserID>alien</UserID>
      <Password>password</Password>
      <TCPCommandChannel>
9000
</TCPCommandChannel>
      <HTTPCommandChannel>
9009
</HTTPCommandChannel>
      <ReaderEPC>
3500000000AABBCCDDEEFF0
</ReaderEPC>
<AvailableReadPointCount>
2
</AvailableReadPointCount>
      <ReaderSerialNumber>
```

```
ALR 9800 KOR
</ReaderSerialNumber>
    <Manufacturer>
Alien Technology Corp.
</Manufacturer>
    <EventSub>
<EventSubType>
RPExtension
      </EventSubType>
      <UserDefinedFieldList>
     <TagField>tf0</TagField>
    </UserDefinedFieldList>
   </EventSub>
</ReaderData>
</ConfigReaderData>
```

표6과 같이 EventSub에 설정될 인자들을 EventSub의 하위 노드에 설정해 둔다. EventSubType은 RPExtension, RPStandard, NoSmooth로 설정할 수 있다. RPStandard로 EventSub가 설정이 되었다면 RP 에서 제시된 이벤트 처리를 하면 된다. EventSub에서는 RP에서 제시된 이벤트처리를 하기 위해선, 태그에서 단순히 읽힌 정보만이 아닌 다른 정보들이 필요하다.

EventSub에서 이벤트를 처리하기 위해 쓰는 자료구조인 EventTagInfo 를 설명한 표7이다.

표 7 EventTagInfo Class

항목	타입	설명
Info	CustomTagInfo	태그에서 읽힌 EPC, 태그가 읽힌 안테나 번호, 읽힌 태그의 전체 메모리 정보값, 벤더필드의 값을 유지함
state	State(enum)	RP에서 이벤트를 일으킬 때 쓰이는 태그의 상태
Tfirst	long	처음 태그가 읽혀졌을 때 시각을 정수화함
Tlast	long	마지막 태그가 읽혀졌을 때 시각을 정수화함.

RPStandard 시에는 EventSub에서 유지하고 있는 데이터를 현재 읽힌 데이터로 갱신해야 한다. Timeout시 발생하지 않고 읽혀져 정상적으로 올라왔다면, 현재의 이벤트를 일으킬 기준 시각(Tnow)을 현재 읽힌 CustomTagData의 읽힌 타임스탬프로 (CustomTagData.getTimestamp)로 설정한다. 하지만, Timeout시 발생하였다면, 현재 읽힌 시각에 대한 정보가 없으므로, 현재 시각을 이벤트 기준 시각으로 설정한다.

올라오는 태그의 데이터는 안테나 단위로 읽혀진 데이터로 올라온다. 보통의 리더설정이 안테나와 태그간의 중복이 아닌 리더와 태그의 중복 제거이다. 하지만, 상위 응용에선 어느 리더에서 읽혔는가 보단, 리더의 어느 안테나에 대한 이벤트에 더 접근하고자 할 수 있다. 이런 경우를 위해 리더와 태그의 중복제거가 아닌 안테나와 태그간의 중복제거로 리더 설정을 바꾸었다.(ALR-9800의 경우 TagListAntenaCombine = Off) 이렇게 함으로써, 태그와 안테나간의 중복이 없는 데이터를 가져올 수 있다. RP에서는 안테나와 소스간의 맵핑을 하고, 소스는 이벤트를 일으킬 때 태그만 이벤트 생성의 단위로 해야 한다. 하지만 현재 구현은 원소스로 고정되어 있고, 이 원소스에 여러개의 물리적인 안테나가 기본적으로 붙어 있다고 가정한다.

현재 구현에서의 이벤트의 생성단위는 현재 읽힌 TagID와 현재 읽혀진 ReadPoint이다. 읽혀져 올라온 CustomTagData내의 CustomTagInfo에서 TagID와 ReadPoint를 키로 잡고, EventTaginfo를 값으로 구성된 Map형식으로 EventSub에

서 유지된다. 이 맵에 있는 값들이 변화하면서 이벤트가 발생한다. 일단 EventSub에 TagID와 ReadPoint로 된 값이 없다면, Tlast와 Tfirst를 Tnow로 설정한 후. CustomTagInfo를 저장한다. 처음 들어온 것은 상태가 isUnknown이며, 만약 있다면 원래 있던 CustomTagInfo를 현재 읽힌 CustomTagInfo로 변경하고, Tlast를 Tnow로 변경한다. RP에서 정해진 대로 처리하여 이벤트를 발생시킨다. 이후 evPurged가 발생하였나 evUnknown이 발생한 태그는 제거되어야 할 목록으로 추가시킨 뒤에 따로 이벤트가 다 발생하고 난 뒤에 태그를 제거 한다.

RPExtension로 EventSub이 설정되었다면 설정파일에서 사전에 선택해 놓은 태그필드리스트 혹은 벤더필드리스트를 이용하여 이벤트를 발생시킨다. 만약 설정파일에 선택해놓지 안으면 RPExtension은 RPStandard와 똑같은 방식으로 이벤트를 발생시킨다. 설정파일에 설정되어진 태그필드나 혹은 벤더필드가 있다면, LIT:evValue이벤트를 추가적으로 발생시켜야 한다.

RP에서 제시된 이벤트의 경우 태그가 실제 물리적 리더로부터 읽혀져 온 CustomTagData가 전해지면, 유지하고 있는 태그의 목록을 최신의 데이터로 갱신하고 난 후, 이에 맞게 이벤트를 발생시킨다. 하지만 RPExtension에 경우 이전에 읽혀진 태그의 벤더필드나 태그필드를 활용해야 하기 때문에 최근에 읽혀진 데이터로 변경하기 이전의 데이터와 최근 데이터를 비교해야 한다. 비교할 시에는 갱신하기 전 태그의 상태가 isObserved이고, 선택된 벤더 필드나 태그필드가 해당되는 실제 물리적 리더로부터 실제 값을 가져온다면, 저장해 놓은 태그 목록에서 이전 태그의 값을 비교하여 이벤트를 발생시킨다.

하지만 태그의 목록에서 유지하는 태그필드에 관련된 선택되어진 값이 없고, 리더로부터 읽혀진 데이터가 있으며, 이벤트를 발생시켜야 한다고 생각할 수 있으나, 실제 태그의 값이 변경이 되었다고는 볼 수 없다. 이 경우는 전에 읽으려고 했던 메모리영역을 못 읽었을 뿐이다. 그러므로 LIT:evValue이벤트를 발생시키면 안 된다. 하지만 실제 값이 올라왔기 때문에 최신 값으로 변경한다. 태그의 목록에 값을 유지하고 있는 값이 있고 최신의 값이 없다고 해도, 이것 역시 동일하게 보고 이벤트를 발생시키면 안 되며 유지하고 있는 태그의 목록에서 해당 태그의 값을 변경하면 안 된다. 벤더 필드 역시 태그필드와 마찬가지로 값을 가지지 못하는 경우가 발생할 수 있다. 이 경우도 역시 LIT:evValue이벤트를 발생시키면 안 된다.

태그필드로 메모리의 값을 비교할 시엔, CustomTagInfo를 비트단위(1,0)의 문자열로 만들어 비교하였다. CustomTagInfo의 TagFields에 있는 메모리값은 0~F까지의 16진수 문자열로 구성된다. 이 메모리 값을 리틀엔디언 형식의 비트 문자열로 바꾼다. 바꿀 때, 맨 앞자리가 0이더라도 4개의 0을 넣어서 자릿수를 맞춘다. 그리고 다음 표8과 같이 첫 자리수가 나왔을 경우 앞의 자리에 보정한다.

표 8 이진 변환 후 보정

첫 번째로 나오는 수	추가하는 0의 개수
1	3
2, 3	2
5, 6, 7	1
8, 9, A, B, C, D, E, F	0

이렇게 보정한 뒤에 비트단위로 문자열로서 추출한다. 이렇게 추출한 비트문자열을 다시 16진수로 바꾸어, 비교를 할 수 있다.

NoSmooth의 경우, EventSub의 스무딩과정을 거치지nav 않고, 물리적 리더에서 올라오는 Raw Data, 혹은 물리적 리더에서 이미 이벤트 처리가 되어, 스무딩하는 것이 의미가 없는 경우 RP에 맞게 리더로부터 올라온 CustomTagData를 ReadReport형식으로 바꾸어 이벤트를 발생시킨다.

이렇게 EventSub에서 발생한 이벤트는 Report Buffer에 쌓여 지고, 이 Report Buffer에서 Notification Trigger에 의해 호스트측으로 전송된다. 이때 Data Selecting을 하기 전에 벤더 이벤트 필터인 LIT:evValue나 LIT:evAlarm을 추가해 데이터 선택 시 이벤트가 누락되지 않도록 추가해야 한다.

4. 적용 사례 및 미들웨어 연동 방법

4.1. 실제 적용 사례

실제 물리적인 리더 (ALR-9800)에서 태그의 메모리값을 변경하여 LIT:evValue를 발생시켜 보았다.

표 9 실제 LIT:evValue의 이벤트 메시지

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<notification>
    <id>1</id>
    <reader>
        <readerEPC>3500000000AABBCCDDEEFF3</readerEPC>
        <readerName>Alien9800</readerName>
        <readerRole/>
        <readerType>Passive</readerType>
    </reader>
    <notifyTriggerName/>
    <notifyChangeName/>
    <readReport>
        <sourceReport>
            <sourceInfo>
                <sourceName>s0</sourceName>
            </sourceInfo>
        <tag>
          <tagID>
            3500003200000C8000000003
          </tagID>
            <tagIDAsPureURI>
                urn:epc:id:gid:800.200.3
            </tagIDAsPureURI>
            <tagIDAsTagURI>
                urn:epc:tag:gid-96:800.200.3
            </tagIDAsTagURI>
```

PUKYONG NATIONAL UNIVERSITY 부경대학교

```
            <tagEvent>
            <eventType>LIT:evValue</eventType>
            <time>
               <eventTimeUTC>
                  2010-04-20T23:36:29.359KST
               </eventTimeUTC>
               </time>
            </tagEvent>
            <tagFields>
                <Temperature>33330000</Temperature>
            </tagFields>
            <other>
                <LIT:ReadPoint>0</LIT:ReadPoint>
            </other>
         </tag>
      </sourceReport>
   </readReport>
</notification>
```

표9는 리더 설정이 RPExtension으로 설정되어져 있고, 설정된 태그필드는 tf0이다. 실제 태그의 상태가 isObserved가 된 이후 태그 쓰기로 인해 값이 변경이 되어, 이벤트가 일어나 호스트에 전달된 LIT:evValue 메시지이다. 선택된 tf0는 내부 객체관리를 위한 이름이며, 실제 호스트로 전달할 때는 tf0와 같은 객체의 이름이 아니라, 외부 표현을 위한 이름인 Temperaturre가 사용되어 호스트로 전달되어진다. evValue라는 이 메시지를 통해 호스트 측에서는 유저메모리에서 메모리 값이 변경이 되었다는 이벤트를 받아 통합적인 이벤트 처리가 가능해진다.

4.2. RFID 미들웨어 연동 방법

RP에서 처리하는 이벤트를 상위 호스트는 Application Level Event(ALE)[12]이라 하는 미들웨어가 관리한다. ALE는 RP에서 일어나는 태그의 이벤트를 필터링 및 그룹핑하는 EPCglobal에서 제시하는 EPCglobal의 미들웨어 규격이다. ALE1.0에선 태그의 EPC 리딩(ECSpec) 기능에 초점이 맞추어 제안된 규격이었다, ALE1.0은 유저 메모리등의 접근에 대한 고려가 없었다. ALE 1.1에서는 태그의 EPC 리딩뿐만 아니라 EPC쓰기(CCSpec)도 지원할 뿐만 아니라, 유저 메모리에 접근하여(TMSpec) 읽기,쓰기 기능을 확장하였다. RP만 아니라 LLRP[13]도 지원하며, Lock과 Kill도 추가되었다.

ALE와 RP는 RP의 커맨드채널로 명령을 주고 ALE의 ECSpec관련된 이벤트 처리는 RP의 Notification Channel로 올라오는 이벤트로 처리하는 것이 보통이다. 하지만, 사용자가 유저메모리에 쓰려고 한다면 다소 문제가 발생한다. 사용자가 TMSpec을 등록하고, 이것을 이용한 CCSpec으로 메모리 쓰기 하고 나면 RP는 메모리 쓰기에 대한 성공여부에 대해선 에러체크를 하지 않는다. 태그를 읽어 보아야 하는데, RP 표준 이벤트 모델로는 isObserved가 지속되고 있는 한 노티피케이션으로 올라오지 않는다. 그렇다고 쓰기 이후 바로 읽게끔 할 수 도 있으나, 이것 역시 ALE의 상에선 두 개의 채널을 통해 값을 받아야 하므로, 일괄적이지 못하다. 그리고, 센서 태그의 경우 RP에서는 계속 읽히는 것으로 찍혀 센싱값이 변화하여도 이 값은 RP 표준 이벤트 모델로는 evObserved발생 이후에는 태그가 사라지거나 혹은 다시 사라졌다 나타나지 않는 이상 최신 값은 노티피케이션을 통해 올라가지 않는다. 그렇다고 ALE에서 계속 원타임 리드를 할 수 있지만, 이렇게 되면 ALE에서 태그의 이벤트를 발생시켜야 한다.

하지만, RP 확장 이벤트 모델을 통해 이런 ALE가 가지는 문제점을 해결 할 수 있다. 표9의 메시지가 evObserved 이후에 온다는 것을 안다면 ALE는 태그의 태그필드가 바뀌었다는 것을 알 수 있다.

5. 결 론

태그의 최근 읽힌 시각과 태그의 마지막 읽힌 시각으로 판단하는 RP의 이벤트 모델은 유저 메모리 변경이나 센서태그에 대한 이벤트를 처리할 수 없다. 본 논문에서는 센싱값 또는 태그 메모리 값을 RP 표준의 태그필드나 벤더필드 값으로 대응시키고 그 값이 변경되는 경우에 이벤트를 발생하도록 하였다. 그리고 RP의 표준을 확장한 메시지를 사용하여 호스트에 전달할 수 있도록 하였다. 따라서 RFID 응용프로그램의 경우 태그식별자 이외에 태그 내의 각종 센싱값 또는 메모리 값에 기반한 이벤트 처리를 용이하게 한다. 따라서 리더와 미들웨어사이의 데이터 전송량을 줄일 수 있고 미들웨어의 부하를 감소시키며, 유저메모리에 대한 쓰기를 미들웨어 수준에서 효율적으로 수행할 수 있게 할 수 있다. 또한 RP를 확장하였으므로 RFID 미들웨어를 크게 수정하지 않고서도 추가된 이벤트를 사용할 수 있다.

PUKYONG NATIONAL UNIVERSITY 부경대학교

참고문헌

[1] J. R. Smith, A. P. Sample, P. S. Powledge, S. Roy, and A. Mamishev. A wirelessly powered platform for sensing and computation. In 8th International Conference on Ubiquitous Computing (Ubicomp 2006),pp. 495 - 506, 2006.

[2] EPCglobal, Class 1 Generation 2 UHF Air Interface Protocol Standard "Gen 2", 2008

[3] S. Jeffery, M. Garofalakis, and M. Franklin. Adaptive cleaning for RFID data streams. In Very Large Database, pp. 163 - 174, 2006.

[4] EPCglobal, RP Standard Version 1.1 Ratified Standard, 2006

[5] 빌 글로버, 히만슈 바트 공저, 서환수 역, 실무자를 위한 RFID이해와 활용, 한빛미디어, 2007년 2월

[6] EPCglobal, Object Naming Service(ONS) Version 1.0 Ratified Specification, 2005

[7] N. Khoussainova, M. Balazinska, and D. Suciu. Probabilistic Event Extraction from RFID Data. In International Conference on Data Engineering, pp. 1480 - 1482, 2008

[8] W. Wang, J. Sung, and D. Kim. Complex event processing in epc sensor network middleware for both rfid and wsn. In Proc. of the 11th IEEE Symp. on Object Oriented Real Time Distributed Computing(ISORC), pp. 165 169, 2008.

[9] EPCglobal, Tag Data Standards Version 1.4 Ratified on June 11, 2008

[10] ISO/IEC, ISO/IEC 15961(Data Protocol: Application Interface)

[11] Alien Technology, Reader Interface Guide, 2008

[12] EPCglobal, The Application Level Events (ALE) Specification, Version 1.1.1

[13] EPCglobal, Low Level Reader Protocol (LLRP), Version 1.0.1

Thanks to...

대학교 입학한지 엊그제 같은데, 이제 대학원 석사로 졸업하려니 지나간 세월이 스쳐 지나갑니다. 대학원 생활이 힘들었지만, 한편으로는 가장 보람되었고, 앞으로 살아가는데 있어서 가장 좋은 추억이자 버팀목이 될 것입니다. 이 점을 가슴에 새기며 이제 다른 곳에 가서도 이 추억을 잊지 않을 것입니다.

먼저, 부족하였던 절 받아주시고 믿어 주신 송하주 교수님께 감사의 말씀을 드립니다. 그리고 논문을 내는데 여러 가지 결정적인 도움을 많이 주신 명환형님, 오탈자를 가장 많이 찾아 주었던 영준, 구현을 가장 많이 같이 한 성진, 잔소리를 많이 하셨지만 뼈가 되고 살이 된 태용 형, 여러 가지 많은 도움을 받았던 영민, 절 많이 믿고 따라 주었던 주형이와 재형이에게 감사의 말을 전합니다.

그리고 거명하진 않지만, 많은 교수님과 조교 선생님들에게도 감사의 말씀을 드리며, 같이 수업을 들으며 생활을 같이한 대학원 선배 및 후배들에게도 감사의 말을 전합니다.

이렇게 키워주시고 많은 뒷바라지 해주신 부모님께 감사의 말을 전하고 틈 있을 때마다 저에게 많은 조언을 해준 누나에게 감사의 말을 전합니다.

저는 이제 가지만, 영원히 IDBLAB일원이 되겠습니다.

2010년 7월

최승혁