RFID 응용프로그램의 효율적 개발을 지원하기 위한 Business Event Framework

이찬영° 김영봉 문미경 염근혁
부산대학교 컴퓨터공학과
cyoung°@hananet.net {saram, mkmoon, yeom}@pusan.ac.kr

Business Event Framework to Support Effective Development of RFID Application

Chanyoung Lee° Youngbong Kim Mikyeong Moon Keunhyuk Yeom
Computer Engineering Department, Pusan National University

요 약

RFID 응용프로그램은 필요한 기능을 수행하기 위하여 하위의 ALE Interface를 이용하거나 EPCIS 등의 외부 시스템과 통신을 해야 한다. 이러한 동작을 수행하기 위하여 해당 응용프로그램 개발자는 하위 ALE Interface를 사용하기위한 방법과 다른 추가 시스템의 사용법을 배워야 하고 정해진 통신 방법에 따라 프로그램을 작성해야 한다. 이러한 작업은 RFID에 대한 지식수준과 프로그래밍 실력에 따라 많은 시간과 노력을 필요로 하게 된다. 본 논문에서는 이러한 RFID 응용 프로그램 개발 시의 어려움을 최소화 하고자 Business Event Framework(BEF)을 제안한다. BEF는 비즈니스 로직과 직접적으로 관련이 없는 부분은 BEF가 직접수행하고 응용프로그램에서 꼭 필요한 이벤트만을 받을 수 있도록 지원하는 라이브러리이다.

1. 서 론

RFID(Radio Frequency IDentification) 시스템은 RFID를 이용하여 기존에 수동으로 처리하던 작업을 자동으로 처리할 수 있도록 자동화한 시스템이다. RFID 시스템은 RFID 정보를 이용할 수 있도록 지원하는 미들웨어와 이 미들웨어로부터 태그정보를 받아서 비즈니스 로직을 처리하는 응용프로그램, 그리고 외부 지원 시스템 등으로 구성된다. 응용프로그램 개발자는 원하는 태그정보를 얻기 위하여 ECSpec을 ALE Engine에 등록하고 ALE Engine은 그 결과로 ECReport를 넘겨주게 된다[1][2]. 응용프로그램에서는 이 ECReport를 분석하여 리더로부터 읽혀진 태그 정보를 얻게 되고 이 태그 정보를 이용하여 EPCIS에서 상세 정보를 질의하게 된다. 이 과정에서 특정 EPCIS의 주소를 알고자 ONS 질의 과정도 거치게 된다. 태그 정보에 대한 상세 정보를 얻게 된 뒤에는 추가적인 비즈니스 로직에 따른 처리를 하게 된다. 이처럼 응용프로그래머는 RFID 시스템을 위한 프로그램 작성 시 하위 미들웨어인 ALE Engine를 이용하고 추가적으로 EPCIS나 ONS등을 사용하게 되므로 RFID에 대한 기반 지식과 다양한 통신 방법에 대한 지식이 필요하게 된다. 그러나 이러한 지식은 해당 응용프로그램의 비즈니스 로직과는 큰 연관이 없는 기술적인 문제이므로 이러한 부분을 최소화함으로써 응용프로그램 개발의 비용과 시간을 줄이고자 한다.

본 논문에서는 위와 같은 문제점을 해결하기 위하여 ALE Engine과 응용프로그램 사이에 새로운 Business Event Framework(BEF)을 두고자 한다. 이러한 BEF는 비즈니스 로직과 직접적인 관련이 없는 여러 과정들을 캡슐화시켜 놓은 것으로 이것을 이용하는 프로그래머가 RFID 응용프로그램을 쉽고 빠르게 개발할 수 있도록 지원하게 된다.

2. 배 경

현재 RFID 미들웨어의 표준은 EPCGlobal에서 제시한 EPC Network Architecture에 기반하고 있다. 이 Architecture는 과거 Savant라고 불리는 미들웨어가 중심이 되다가 현재는 ALE Engine이 중심이 되는 새로운 구조로 바뀌었다. (그림 1)[1].

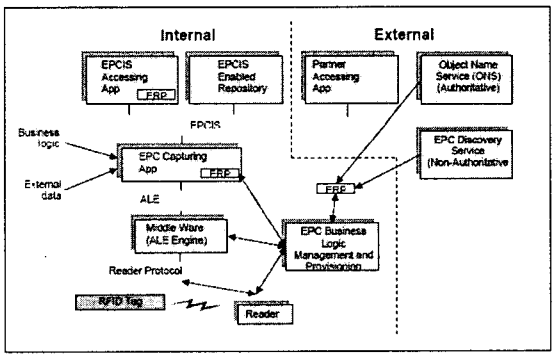

그림 1 EPC Network Architecture

그림 1과 같은 구조에서는 응용프로그램이 ALE Engine을 통하여 태그정보를 얻게 되는데 응용프로그램이 비즈니스 로직을 수행하기 위하여 필요한 정보는 태그 정보가 아니라 이 태그 정보를 이용하여 얻어진 실제

이 논문은 교육인적자원부 지방연구중심대학육성사업 "차세대물류IT기술연구 사업단"의 지원에 의하여 연구되었음

제품의 가격이나 명칭 또는 가격총합 등의 상세정보(Business Event)이다. 이처럼 응용프로그램이 필요로 하는 정보와 미들웨어인 ALE Engine이 넘겨주는 정보 사이에는 추상화 레벨이 다르기 때문에 응용 프로그래머는 추가적인 코드를 작성하여 하위 수준의 태그정보를 상위 수준의 정보로 가공하게 된다.

하지만 응용프로그램에 Business Event를 바로 전달해 줄 수 있게 되면 응용프로그래머가 해야 할 일이 많이 줄어들게 되어 쉽게 응용프로그램을 개발할 수 있다. 이러한 점에 착안하여 그림 2와 같은 구조를 제시하고자 한다.

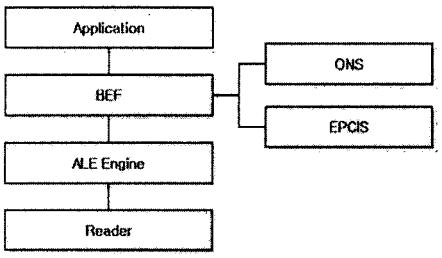

그림 2 BEF 기반 구성도

BEF 기반 구조에서는 태그 정보와 같은 하위 수준의 정보는 BEF가 처리를 담당하게 되고 응용프로그램이 원하는 수준의 정보로 가공이 완료되면 그 때 응용프로그램에 정보를 넘겨주게 된다. 이 때 BEF가 어떠한 형태로 정보를 가공할 것인지는 애플리케이션의 종류에 따라 다르기 때문에 이러한 처리과정을 모델링 할 수 있도록 비즈니스 이벤트 명세(Business Event Specification: BESpec)를 정의한다. BESpec은 Business Process를 정의하기 위해 개발된 Business Process Execution Language(BPEL)[3]의 아이디어를 바탕으로 RFID 시스템 고유의 특징을 반영하여 정의하였다. BEF에서는 응용프로그램으로부터 넘겨받은 BESpec을 분석하여 원하는 형태로 정보를 가공하여 응용프로그램으로 전달하게 된다.

3. 비즈니스 이벤트 명세 (BESpec)

BESpec은 응용프로그램 개발자가 원하는 최종 이벤트를 생성하기 위하여 기술하는 명세이다. 이 명세에는 여러 요소들이 미리 정의되어 있어서 응용프로그램 개발자가 이러한 요소들을 이용하여 원하는 결과를 얻을 수 있다. BESpec은 쉬운 작성과 장래의 확장성을 위하여 XML형태로 되어 있다. 각각의 요소들에 대한 문법은 XML 형식에 맞게 정의되어 있어서 사용자는 각 활동에 맞는 태그를 사용하여 쉽게 BESpec을 작성할 수 있다.

3.1 Variable

BESpec 태그정보를 읽었을 때 이에 따른 여러 처리과정을 거치게 된다. 각각의 처리과정은 독립적으로 수행될 수 있는데 이러한 각 처리과정 사이에 필요한 정보를 넘겨주기 위해 variable을 사용한다. 각 처리과정에서는 처리 결과를 variable에 저장한다던지 입력으로 variable을 사용하게 된다. 이러한 variable은 int, string 등의 일반적인 데이터뿐만 아니라 RFID 시스템에 특화된 데이터도 지원하여 BESpec의 사용 편이성을 높인다. 예를 들면 하위 ALE Engine에서 한 event cycle동안 넘겨주는 태그 목록들을 한 번에 저장할 수 있는 태그리스트 같은 데이터 타입을 제공하여 BESpec 작성 시 편의성과 가독성을 높일 수 있다.

3.2 처리 활동

응용프로그램에 따라 필요한 정보가 다르기 때문에 다양한 응용프로그램을 지원하기 위해서는 여러 가지 처리 활동을 지원해야 한다. 기본적으로 ALE Engine으로부터 태그정보를 얻는 활동, EPCIS 질의 활동 등이 있을 수 있고 추가적으로 산술 연산, 동기화 처리 등의 활동이 있을 수 있다.

3.2.1 ALE Activity

ALE Activity는 ALE Engine에서 어떠한 정보를 받을지 정의하게 된다. 리더로부터 정보를 읽는 시간, 필터링 정보, 결과에 포함할 내용, 그룹화 정보 등을 응용프로그램의 필요에 따라 정의하게 된다. 이러한 정의 내용을 바탕으로 실제 처리 시에는 ALE Engine에서 입력으로 받아들이는 ECSpec을 생성하여 ALE Engine에 넘겨주게 된다. 또한 결과로 EPReport가 수신하게 되면 이것을 분석하여 응용프로그래머가 원하는 데이터를 정의된 변수에 할당하게 된다.

3.2.2 EPCIS Activity

ALE Engine으로부터 받아들인 태그 정보를 이용하여 EPCIS에 저장된 상세 정보를 얻는 활동이다. 응용프로그래머는 태그에 따라 필요한 상세정보를 얻기 위하여 스키마와 속성이름만을 기술하여 원하는 정보를 얻게 된다. 그림 3은 EPCIS Activity를 기술하고 있는 BCSpeci에 대한 예이다. EPCListTemp에 저장된 태그 값들을 하나씩 EPCTemp에 대입하고 이 태그 값이 가지고 있는 "차량명"과 "배기량" 정보를 EPCIS에 접근하여 조회한 후, 값을 받아온다. 이 값들은 stringTemp와 stringTemp2 변수에 대입된다.

```
- <list source="EPCListTemp" assign="EPCTemp">
  - <EPCIS>
      <getAttribute epc="EPCTemp" schema="pnu/ai/static/class/자동차" xpath="차종명">stringTemp</getAttribute>
    </EPCIS>
  - <EPCIS>
      <getAttribute epc="EPCTemp" schema="pnu/ai/static/class/자동차" xpath="배기량">stringTemp2</getAttribute>
    </EPCIS>
```

그림 3 EPCIS Activity를 기술하고 있는 BESpec

3.2.3. Arithmetic Activity

처리과정 중에 필요한 기본적인 산술 연산을 수행한다. 응용프로그램에서 필요한 최종적인 정보가 수량의 합계나 가격의 총합 등일 때 사용될 수 있다.

3.2.4 Switch Activity

조건에 따라 다른 처리 과정이 필요할 때 사용하는 활

동이다.

3.2.5 Notification Activity

처리 과정에 완료된 최종 Business Event를 응용프로그램으로 전달하기 위한 활동이다. 전달 시 응용프로그램이 어떠한 정보인지 알 수 있도록 해더도 같이 전달하여 정보에 맞게 처리할 수 있도록 한다.

3.2.6 기타 Activity

위에 언급된 활동 이외에도 동기화처리, 리스트처리 등의 다양한 활동이 있다.

3.3 BESpec 예제

그림 4는 다음의 시나리오에 대하여 BESpec으로 기술한 예이다.

계산대 리더기를 지나는 상품들에 대하여, 애플리케이션에 상품명, 상품가격을 보낸다.

```
<?xml version="1.0" encoding="UTF-8"?>
<BESpec>
    <variables>
        <variable type="int" initValue="0">intTemp</variable>
        <variable type="string" initValue="tttggg">stringTemp</variable>
        <variable type="string">stringTemp2</variable>
        <variable type="EPCList">EPCListTemp</variable>
        <variable type="EPC">EPCTemp</variable>
    </variables>
    <ale:ECSpec>
        <logicalReaders>
            <logicalReader>total</logicalReader>
        </logicalReaders>
        <boundarySpec>
            <duration unit="MS">1000</duration>
        </boundarySpec>
        <reportSpecs>
            <reportSpec reportName="report1">
                <reportSet set="CURRENT"/>
                <output includeEPC="true" includeCount="true"/>
                <resultEPC>EPCListTemp</resultEPC>
                <resultCount>intTemp</resultCount>
            </reportSpec>
        </reportSpecs>
    </ale:ECSpec>
    <list source="EPCListTemp" assign="EPCTemp">
        <EPCIS>
            <getAttribute epc="EPCTemp" schema="pnu/ai/static/class/제품"
                xpath="제품명">stringTemp</getAttribute>
        </EPCIS>
        <EPCIS>
            <getAttribute epc="EPCTemp" schema="pnu/ai/static/class/제품"
                xpath="가격">stringTemp2</getAttribute>
        </EPCIS>
        <notify name="제품">
            <data name="제품명">stringTemp</data>
            <data name="가격">stringTemp2</data>
        </notify>
    </list>
```

그림 4 BESpec 예

4. Business Event Framework(BEF)

BEF는 BESpec에 따라 실제 동작을 수행하는 부분이다. 응용프로그램 개발자는 어떤 BESpec을 수행할 지 BEF에 알려주면 BEF는 해당 BESpec을 읽어서 분석을 하게 된다. 분석의 결과로 BESpec의 여러 활동들이 추출되면 순서에 따라 각 활동을 수행하게 된다. 각 활동을 수행하는 것은 내부적으로 독립적인 Agent가 수행을 하게 된다. 이를 통하여 향후 BESpec 확장 시 BEF 개발을 편리하도록 하였다.

수행 과정을 살펴보면 일반적으로 가장 먼저 ALE Activity부터 수행하게 되고 그 결과가 ALE Engine으로부터 오게 되면 각 결과에 따라 다른 쓰레드로 이후 처리과정을 수행하게 된다. 그림 3의 경우를 예로 들면 먼저 결과 태그 목록에 대하여 각 태그별로 제품명과 가격을 얻어오기 위하여 EPCIS와 통신을 한다. 이 때 응용프로그래머는 ALE나 EPCIS와의 통신 프로토콜이 웹 서비스인지 TCP/IP인지 또는 자바 RMI등을 이용하는지 전혀 알 필요가 없이 원하는 상세정보를 나타내는 스키마와 xpath만을 넘겨주게 되고 세부적인 것은 BEF가 다 알아서 처리하게 된다. 이 후에는 처리된 결과를 응용프로그램에 알려주게 되는데 현재에는 콜백 함수 형태로 전달을 하고 있고 향후에는 응용프로그램 작성자의 다양한 요구를 만족할 수 있도록 다양한 방법을 지원할 예정이다.

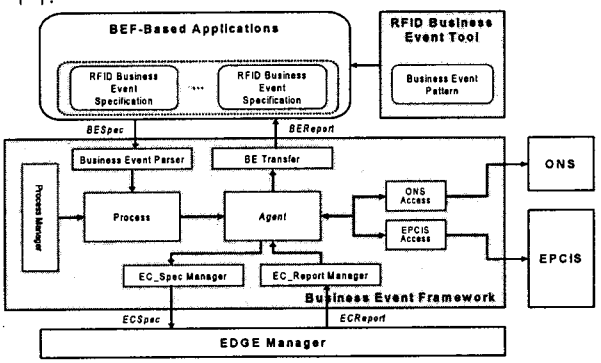

그림 5 비즈니스 이벤트 프레임워크 아키텍처

5. 결론

본 논문에서는 빠르고 쉽게 응용프로그램을 개발할 수 있도록 지원하는 BEF에 대하여 설명하였다. BEF를 이용하게 되면 응용프로그램 개발자는 RFID와 관련된 아주 상세한 내용과 외부 시스템과의 통신에 필요한 세부적인 지식이 필요 없이 쉽게 프로그램을 작성할 수 있다. BEF를 사용한 경우와 BEF를 사용하지 않은 경우의 응용프로그램의 코드 라인 수는 아주 큰 차이를 보인다. 물론 BESpec을 추가적으로 작성해야하지만 BESpec을 포함하더라고 BEF를 사용하지 않았을 때의 코드와는 많은 차이가 있다. 또한 응용프로그램 코드에서 비즈니스 로직을 처리하는 코드와 RFID 관련 처리를 하는 코드(BESpec)가 완전히 분리되어 있기 때문에 프로그램 디버깅이나 향후 프로그램 수정 또는 확장 시 가독성이 아주 높게 된다.

향후 연구과제로는 BESpec을 텍스트 형태로 작성하는 것이 아니라 GUI 형태의 BESpec 작성 툴을 개발하는 것이다. 이 툴을 개발하게 되면 BESpec 작성의 편의성을 향상시킬 수 있으며 오류의 수도 줄일 수 있게 될 것이다.

6. 참고문헌

[1] EPCGlobal, "EPCglobal Object Name Service (ONS) 1.0", April 2004.

[2] EPCGlobal,, "The Application Level Events (ALE) Specification, Version 1.0", February 2005

[3] Oracle , "Oracle BPEL Process Manager", June 2005.