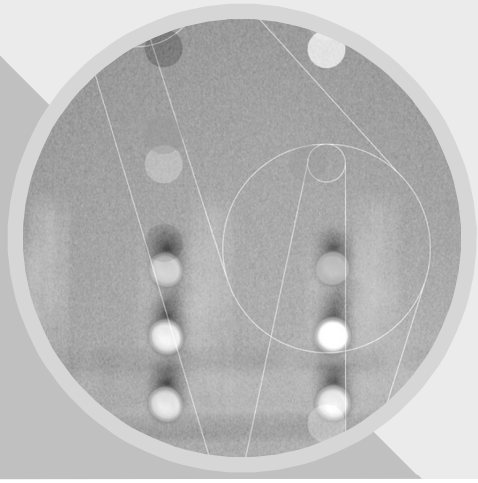


Chapter 07

RFID 미들웨어

▶▶▶ RFID MIddleware

2장에서도 얘기했듯이 적합한 태그와 리더를 선택하고 안테나를 설치할 위치를 결정하는 것은 RFID 시스템을 구축하는 데 있어서 첫 번째 단계에 지나지 않는다. 3장부터 6장까지는 RFID를 이루는 일부 물리적 구성 요소가 어떤 식으로 작동하는지 알아보았다. 이제 그런 물리적인 인프라를 통해 수집된 정보가 어떤 식으로 회사의 애플리케이션으로 넘어가서 소화되는지, 그리고 왜 RFID 미들웨어가 필요한지 알아보도록 하자.

동기

RFID 미들웨어를 사용하는 동기로는 크게 세 가지를 들 수 있다. 첫 번째는 애플리케이션을 장치 인터페이스로부터 캡슐화 하는 것이고, 두 번째는 리더와 센서에서 잡아낸 처리되지 않은 관측 결과를 처리해서 애플리케이션에서는 쓸모가 있는 고수준 이벤트만 볼 수 있도록 하고, 또한 애플리케이션에서 처리할 데이터 분량을 적정 수준으로 낮추는 것이다. 마지막으로 세 번째 동기는 리더를 관리하고 RFID 관측을 질의하기 위한 애플리케이션 수준의 인터페이스를 제공하는 것이다. 요즘 나오는 RFID 미들웨어들은 대부분 이런 기능을 제공한다. [그림 7-1]에 RFID 미들웨어의 핵심 구성 요소가 나와 있다.

[그림 7-1] RFID 미들웨어의 구성 요소

각 구성 요소가 필요한 이유에 대해서는 2장에서 살펴보았다. 여기에서는 RFID 미들웨어가 필요한 이유를 하나씩 자세히 짚어보자.

리더 인터페이스 제공

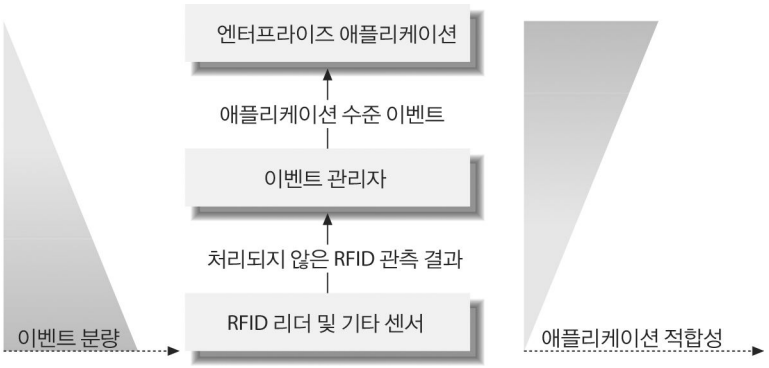
여러분이 가지고 있는 물리적인 인프라에서 애플리케이션이 리더 및 기타 센서와 어떤 식으로 인터페이스할지 생각해 보자. 애플리케이션을 만들 때마다 각 리더 유형마다 제공되는 API를 가지고 만들 수도 있을 텐데, 일반적인 엔터프라이즈 환경에서는 여러 회사에서 생산하는 다양한 리더들을 사용할 가능성이 매우 높기 때문에 정말 단순한 상황을 제외하면 별로 좋은 방법이 아니다. 웬만한 회사라면 소프트웨어 개발 전문 회사에서 리더 API를 바탕으로 제작한 별도의 장치 드라이버 또는 리더 인터페이스를 사용할 것이다. 리더 어댑터는 서로 다른 리더와 API 때문에 생길 수 있는 골치 아픈 문제를 해결할 수 있는 방법을 제공하며, 애플리케이션에서 하나의 추상적인 인터페이스만 사용하면 되도록 해 준다.

이벤트 필터링

RFID를 사용하는 도매 또는 소매 매장에는 리더가 수천 개까지는 아니더라도 적어도 수백 개 정도는 설치된다. 이런 리더들은 각각 RFID 태그를 읽기 위해 초당 몇 번씩 신호를 내보낸다. 2장에서도 논의했지만, 결과적으로 RFID 읽기 관측이 초당 수백만 건 정도 발생하게 된다. 리더와 센서에서 만들어진 처리되지 않은 관측 결과를 바로 엔터프라이즈 애플리케이션에 노출시키는 것은 소방전 호스에 물을 콸콸 틀어놓고 그 물을 마시려고 하는 것과 마찬가지다. 단순히 데이터 분량이 문제가 되는 것이 아니고, 엔터프라이즈 환경에서 쓸모가 있으려면 관측 결과를 어떤 식으로든 처리를 해야 한다. RF 통신의 물리적 특성 때문에 현재 상용 환경에서 읽기의 정확도는 80~99% 정도이다. 즉, 리더 근처에 태그가 100개 있었다면 매번 읽기 사이클마다 약 80~99개 정도가 읽힌다. 읽기 정확도가 100%가 안 되기 때문에 한 사이클에서 읽을 수 있었던 태그가 다음 사이클에서는 읽히지 않는 일도 종종 있을 수 있다. 재고 관리 시스템과 통합되어 있는 스마트 선반 애플리케이션이 있다고 해 보자. 그렇다면 스마트 선반에 시스템에서 오는 처리되지 않은 관측 결과를 재고 시스템에 바로 넘기는 것이 현명할까? 만약 그렇게 한다면 들어오는 데이터의 분량에 치이는 것은 물론이고, 스마트 선반에 있는 리더로부터 넘어오는 결과가 계속해서 바뀌기 때문에 관리하기도 쉽지 않다.

[그림 7-2]에 나와 있듯이 RFID 리더와 센서에서 나오는 처리되지 않은 관측 결과는 애플리케이션 수준에서는 별로 쓸모가 없다. 이런 관측 결과를 애플리케이션에서 사용할 수 있는 이벤트로 만들기 위해서는 또 다른 처리 절차가 필요하다. 예를 들어, 주문 관리용 애플리케이션에서는 매장 내 재고가 특정 수준 밑으로 내려갔다는 사실이 중요할 뿐 자질구레한

정보는 별로 필요가 없다. 주문 관리 시스템의 입장에서는 한 매장에 있는 리더의 개수나 그 구성 같은 것은 말할 것도 없고, 매장에 있는 아이템을 추적하는 데 RFID 리더가 쓰이는지의 여부도 별로 중요하지 않다. RFID 리더에서 매번 스캔을 할 때마다 그 데이터를 애플리케이션 수준의 필터링을 전혀 거치지 않고 바로 주문 관리 시스템에 넘기는 것은 불필요할 뿐만 아니라 생산성에도 도움이 되지 않는다. 결과적으로 리더와 센서에서 넘어오는 관측 결과를 정리하고 모으고 필터링하는 것은 물론, 애플리케이션 수준의 컨텍스트도 제공할 수 있는 미들웨어가 필요하다. 따라서 관측 결과를 애플리케이션에 보내기 전에 어떤 식으로든 처리를 해야만 한다. 리더와 센서에서 들어오는 처리를 거치지 않은 RFID 관측 결과를 평탄화하거나 엔터프라이즈 애플리케이션에서 더 쓸모가 있도록 만드는 것을 이벤트 필터링(event filtering)이라고 부른다. 그리고 이벤트 필터링 기능을 제공하는 구성 요소를 이벤트 관리자라고 부른다.



[그림 7-2] RFID 시스템의 계층별 이벤트 분량 및 애플리케이션 적합성

표준을 기반으로 하는 서비스 인터페이스 제공

RFID 미들웨어를 사용함으로써 얻을 수 있는 가장 큰 장점은 조그만 RFID 태그에 의해 만들어지는 정보의 홍수를 처리하는 표준화된 방법을 제공한다는 것이다. 따라서 필요한 것이 바로 RFID 데이터 모음에 애플리케이션 수준의 의미를 제공하는 서비스 지향 인터페이스 – 여기에서는 애플리케이션 수준 인터페이스라고 부르자 – 이다. 서비스 지향 아키텍처 원칙에 따라 이 인터페이스는 느슨하게 결합돼야 하고, 비동기식이어야 하며 현행 웹 서비스의 표준을 따라야 한다.

논리적 아키텍처

RFID 및 기타 원격 감지 기술은 사람이 직접 개입을 해야 하는 바코드와 같은 레이블링 기법으로는 불가능했던 수준의 자동화를 가능하게 해 준다. 하지만 이런 자동화 수준을 달성하려면 리더와 센서를 원격으로 감독하고 관리할 수 있어야 한다. 말단 장치를 감독하고 관리하는 데는 말단에서 작동하는 미들웨어 솔루션이 가장 적합하다. 따라서 RFID 미들웨어 솔루션은 앞에서 기술한 세 가지 기능 외에도 관리 및 감독용 인터페이스를 제공하거나 그런 인터페이스와 통할될 수 있어야 한다.

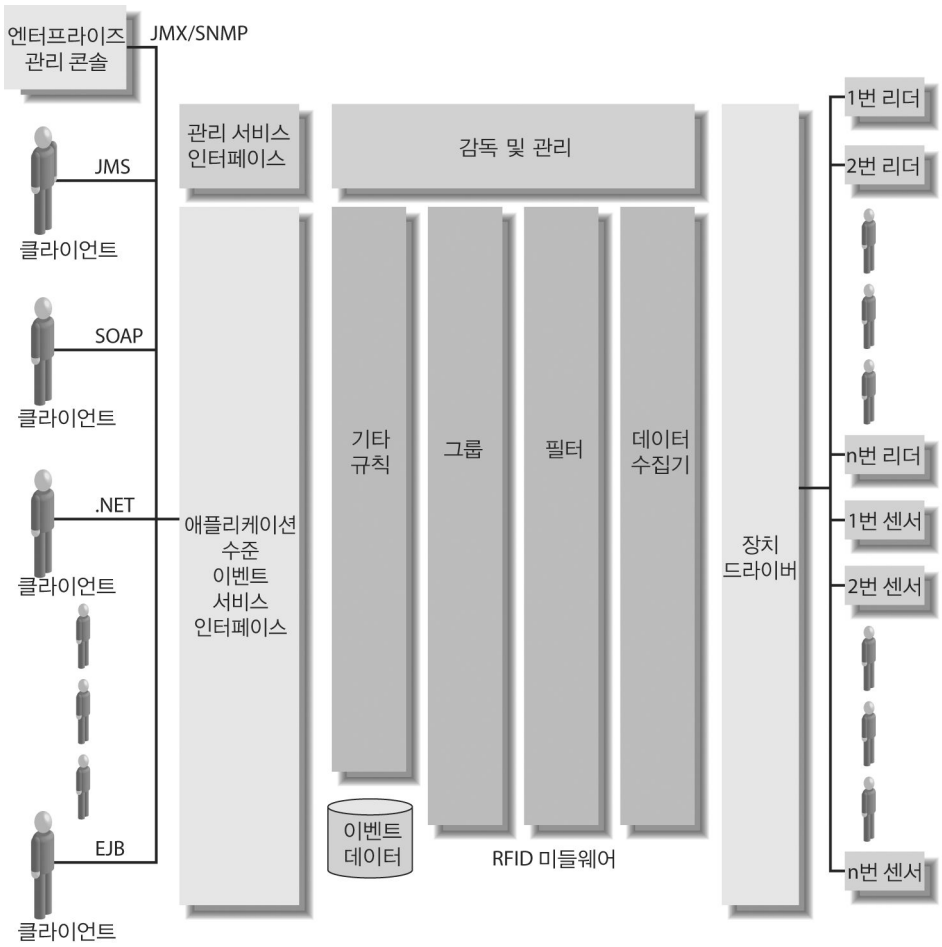
데이터가 많아지고 트랜잭션이 증가하면 네트워크, 서버 및 스토리지 인프라에 걸리는 부하도 늘어난다. 엔터프라이즈 애플리케이션은 보통 데이터 센터에 설치되기 때문에, 엔터프라이즈 애플리케이션에 RFID 리더의 관측 결과를 직접 노출시키면 애플리케이션만 힘들 뿐 아니라 WAN 인프라에도 크게 무리가 된다. 따라서 아주 간단한 용도로 활용하거나 시제품을 만드는 단계가 아니라면 애플리케이션과 말단 장치 사이에서 RFID 미들웨어를 사용해야 한다. 이 미들웨어는 적어도 리더의 유형별로 복잡한 특성을 애플리케이션으로부터 분리시키고, 리더로부터 들어오는 처리되지 않은 관측 결과 때문에 고생하지 않고도 유용한 애플리케이션 수준의 이벤트에만 초점을 맞출 수 있도록 도와줄 수 있어야 한다. 그리고 앞에서도 언급했듯이 미들웨어에서는 원격 감독 및 관리 기능도 있는 것이 좋다.

[그림 7-3]에 RFID 미들웨어의 개념적인 모형이 나와 있다. RFID 미들웨어에서는 하나 이상의 데이터 소스로부터 처리되지 않은 관측 결과를 받아온다. 데이터 소스는 RFID 리더나 온도 센서와 같이 물리적인 세계로부터 데이터를 수집하는 장치를 뜻한다. 리더로부터 관측 결과를 받으면 미들웨어의 이벤트 관리자 구성 요소에서 그 데이터를 결집시키고 변환하거나 필터링해서 애플리케이션에서 사용할 수 있도록 만들어준다. 이벤트 관리자는 RFID 관측 결과를 애플리케이션에 더 적합한 형태로 만드는 것 외에도 애플리케이션에서 처리해야 하는 데이터 분량을 줄이는 데도 도움을 준다.

[그림 7-3]에 나와 있듯이 RFID 미들웨어는 리더 발견, 준비, 감독, 관리를 지원하고 데이터 모음, 전환, 집적, 필터링 및 분류 메커니즘을 제공하고, 자바, J2EE, .NET 및 웹 서비스 같은 표준을 이용한 서비스 지향 인터페이스를 지원하고 원격 준비, 감독 및 관리 기능을 제공한다.

이 논리적인 아키텍처를 구현하는 방법은 다양하겠지만 가장 흔하게 볼 수 있는 EPCglobal의 애플리케이션 수준 이벤트(ALE; Application Level Event) 규격을 살펴보도록 하겠다. ALE 규격에서는 RFID 리더로부터 이벤트를 받아들이고, 그 이벤트들을 필터링하고 분류하기 위한 리더 중립적인 인터페이스를 정의한다. 지금부터는 EPCglobal에서 나온 ALE 1.0 규격에 대해 알아보도록 하겠다. 이 규격을 자세하게 살펴보면서 핵심 개념 및 API에 익숙해

지도록 하자. 시중에는 이미 ALE 규격을 구현한 제품들이 나와 있으며, 각각 서로 다른 확장 기능과 독특한 장점을 내세우고 있다. 이 장 후반부에서 몇 가지의 실제 구현을 다루도록 하겠다. ALE 규격의 전문(全文)은 *http://epcglobalinc.org*에서 찾아볼 수 있다.



[그림 7-3] RFID 미들웨어 제품의 개념적인 아키텍처

애플리케이션 수준 이벤트 규격

ALE 규격은 클라이언트가 다양한 소스로부터 필터링 및 정리가 완료된 EPC 관측을 가져올 수 있도록 하기 위해 EPCglobal에서 개발한 애플리케이션 수준 인터페이스 표준이다. ALE 인터페이스를 이용하면 클라이언트에서 이벤트 처리 방법을 설정하고, 보고서 형태로 필터링된 이벤트를 요청할 수 있다. 그 전신인 Savant(ALE와 Savant 참조)와 마찬가지로 ALE

규격에서도 데이터 소스에 더 가까운 곳으로 EPC 데이터 처리를 넘길 수 있는 방법을 제공한다. 그렇게 하기 위해 ALE 규격에서는 ALE 클라이언트와 서버 사이에서 서비스 인터페이스와 상호 작용 모형을 정의한다. 하지만 Savant 규격과 달리 ALE 규격에서는 서비스 인터페이스가 어떤 식으로 구현되어야 하는지, 또는 어디에 배치할 수 있는지를 지정하지 않는다. 예를 들어, ALE 서비스는 독립적으로 배치될 수도 있고, 리더나 애플리케이션 서버 클러스터에 배치될 수도 있다. 그리고 ALE 규격은 리더 제작업체로 하여금 구현 기법을 선택할 수 있는 기능을 제공한다. 어떤 서비스든지 ALE 인터페이스 규격의 요구 사항만 만족시킬 수 있다면 EPCglobal ALE 규격에 부합하는 것으로 간주된다(EPCglobal에서 ALE 호환 소프트웨어를 테스트하기 위한 방법을 제공할 계획을 가지고 있긴 하다).

ALE 규격의 핵심 장점을 짚어보면 다음과 같다.

■ 이벤트 관리 표준

ALE 규격에서는 RFID 리더로부터 오는 이벤트를 받아서 필터링하고 분류하기 위한 리더 중립적인 인터페이스를 제공한다. ALE 호환 미들웨어를 사용하는 애플리케이션에서는 각 리더의 장치 드라이버가 없어도 되고, 각 리더별 프로그래밍 인터페이스를 사용할 필요도 없다.

■ 확장성

ALE 규격은 매우 확장성이 좋다. 예를 들어, ALE 규격은 EPC 이벤트 소스를 대상으로 하고 있지만 별도의 익스텐션을 만들어서 비 EPC 태그에 연결하거나 RFID 리더가 아닌 장치와 연결하는 것도 가능하다.

■ 인터페이스와 구현의 분리

ALE 규격에서는 클라이언트와 RFID 미들웨어 사이의 인터페이스만 제공할 뿐이고, 자세한 구현 방법은 제작업체 쪽에서 알아서 해야 한다. 이런 접근법을 취하면 제작업체 쪽에서는 기술 플랫폼, 개발 옵션, 추가 기능 등에 있어서 선택의 폭이 넓어진다. 예를 들어, ALE 서비스를 제공하는 소프트웨어는 말단 쪽에 독립형 모듈 형태로 배치될 수도 있고, 애플리케이션 컨테이너의 내부에 배치될 수도 있으며, RFID 리더 안으로 들어갈 수도 있다. 각 배치 방식마다 장단점이 있기 때문에 실제 구현 방법은 선택하기 나름이다.

애플리케이션 수준 이벤트 규격에서는 애플리케이션 수준 이벤트 서비스에 접근하기 위한 WS-*-호환 웹 서비스 바인딩 인터페이스를 제공한다. 구현을 할 때는 ALE 인터페이스를 (SOAP/HTTP 같은) 유선 규약을 통해 공개하거나 언어 API를 통해 공개하는 등 적당히 융통성을 발휘하는 것이 가능하다.

COLUMN

ALE와 Savant

ALE가 나오기 전에는 자동 ID 센터에서 "Savant"라는 구성 요소를 제안했다. "Savant"라는 용어는 일반적으로 일련의 데이터 소스(리더)와 엔터프라이즈 애플리케이션 사이에서 데이터를 필터링하기 위한 용도로 사용되는 소프트웨어를 뜻했다. 원래는 Savant 규격이 RFID 이벤트 처리용 표준을 제공하기 위해 제안되었지만 어떤 서비스가 제공되는지 보다는 이벤트 관리자가 어떻게 구현되는지에 더 초점을 맞추게 되었다. Savant 규격은 EPCglobal의 RFID 이벤트 관리용 규격인 ALE 규격이 나오면서 더 이상 사용되지 않게 되었다.

ALE 서비스 인터페이스를 본격적으로 파헤쳐 보기 전에 우선 몇 가지 중요한 개념과 용어에 익숙해지도록 하자.

핵심 개념과 용어

ALE 규격이 어떤 식으로 작동하는지 이해하려면 몇 가지의 핵심 개념과 용어를 먼저 짚고 넘어가야 한다. 우선 이벤트 발신자부터 시작해서 읽기 사이클, 이벤트 사이클에 대해 알아보자.

일단 이벤트 발신자 및 이벤트 사이클 뒤에 숨어있는 개념들을 이해하고 나면 애플리케이션과 RFID 미들웨어 사이에서 지원되는 ALE 규격을 구현하는 핵심적인 상호 작용 모형을 살펴보도록 하겠다.

이벤트 발신자

이벤트 발신자는 RFID 태그의 존재를 파악하거나 물리적인 세계로부터 무엇이든 관측해 내는 장치를 뜻한다. 이벤트 발신자의 예로 RFID 리더와 센서를 들 수 있다. ALE 규격에서는 물리적인 장치를 리더하고 구분해서 생각한다. ALE 규격의 문맥 안에서는 물리적 장치가 하나 이상의 안테나가 달린 RFID 리더일 수도 있고, EPC 호환 바코드 스캐너일 수도 있고 또 다른 장치일 수도 있다. ALE 규격에서는 리더를 추상적인 개념으로 정의한다. 본질적으로 리더는 처리되지 않은 EPC 이벤트(또는 관측)를 제공하는 데이터 소스를 뜻한다. 리더는 다양한 방식으로 대응될 수 있다.

■ **단일 물리 장치에 한 리더가 대응되는 경우**

리더가 단일 물리 장치에 대응될 수도 있다. 안테나가 한 개인 RFID 리더, EPC 호환 바코드 스캐너, 모든 안테나에서 오는 관측이 전부 하나로 합쳐지는 방식으로 여러 안테나에 연결된 리더와 같은 것이 이런 예에 해당된다.

■ **같은 물리 장치에 여러 리더가 대응되는 경우**
안테나가 여러 개 연결되어 있고, 각 안테나를 서로 다른 소스로 간주하는 리더의 경우와 마찬가지로 한 리더가 여러 리더에 대응되는 경우도 있다.

■ **여러 물리 장치가 한 리더에 대응되는 경우**
여러 리더가 하나처럼 작동해서 종합적인 관측 결과를 도출해 내는 경우도 있다. 예를 들어 두 대 이상의 리더를 사용하여 삼각측량 방식으로 위치 정보를 알아낼 수도 있다.

ALE 규격에서는 하나 이상의 리더를 참조하기 위해 사용하는 레이블(또는 이름)인 논리적인 리더라는 개념을 지원하기도 한다. 사실 이런 기능은 특정 지역에서의 관측을 몇 개의 리더를 가지고 잡아내려고 하는 경우에 사용할 수 있도록 ALE 규격에서 제공하는 분류 메커니즘이다. 예를 들어, 선적장이 10개 있는 창고가 있는데 각각 리더가 두 대씩 설치되어 있다면 그 리더들을 몇 개의 논리적인 리더로 분류할 수 있다. 이런 식으로 하면 각 리더로부터 오는 이벤트를 종합할 필요 없이 단 몇 개의 논리적 리더에만 질의를 보내면 새로 들어오는 제품 정보를 파악하는 것이 가능해진다.

논리적인 리더의 개념을 활용하여 애플리케이션과 리더의 배치 설정을 분리시킬 수도 있다. 예를 들어, 선적장에 있는 여러 대의 물리적인 리더 및 센서를 "선적장 1번"이라는 하나의 논리적인 리더로 대응시킬 수 있다. 이렇게 해 두면 나중에 리더의 설정이 변경되더라도 애플리케이션을 고칠 필요 없이 물리적인 리더와 논리적인 리더 사이의 대응 관계만 변경하면 된다.

읽기 사이클

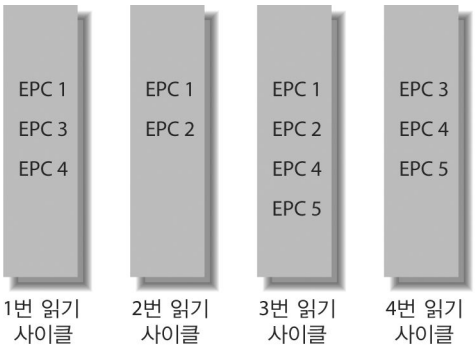
이벤트 관리를 이해하려면 이벤트가 리더에서 어떻게 만들어져서 ALE 서버로, 그리고 ALE 서버의 클라이언트로 전달되는지를 이해하는 것이 중요하다. 리더에서는 정해진 주기마다, 또는 요청에 따라 RFID 태그를 스캔하거나 다른 물리적인 관측을 할 수 있다. 정해진 주기마다 스캔을 한다면 각 스캔을 읽기 사이클(read cycle)이라고 부른다.

읽기 사이클은 리더와의 상호 작용에 있어서 기본 단위가 된다. 각 읽기 사이클마다 리더에서 일련의 RFID 관측 결과를 반환한다. ALE 규격의 경우에는 EPC(Electronic Product Code)가 관측 결과가 된다. ALE 규격에는 리더에서 지원하는 읽기 사이클 타이밍에 대한 제약조건은 포함되어 있지 않다. 읽기 사이클은 시간 간격, 데이터의 양, 기타 센서 입력 등을 바탕으로 정해질 수 있다. [그림 7-4]에 한 리더로부터 오는 읽기 사이클이 그림으로 나와 있다.

한 번의 읽기 사이클 동안 읽은 EPC의 집합을 S라고 표기하면 [그림 7-4]에 나와 있는 네 번의 읽기 사이클의 결과는 다음과 같이 쓸 수 있다.

```
S1 = {EPC1, EPC3, EPC4}
S2 = {EPC1, EPC2}
S3 = {EPC1, EPC2, EPC4, EPC5}
S4 = {EPC3, EPC4, EPC5}
```

즉, 1번 읽기 사이클(S1)에서는 EPC1, EPC3, EPC4가 반환된다.



[그림 7-4] 읽기 사이클의 예

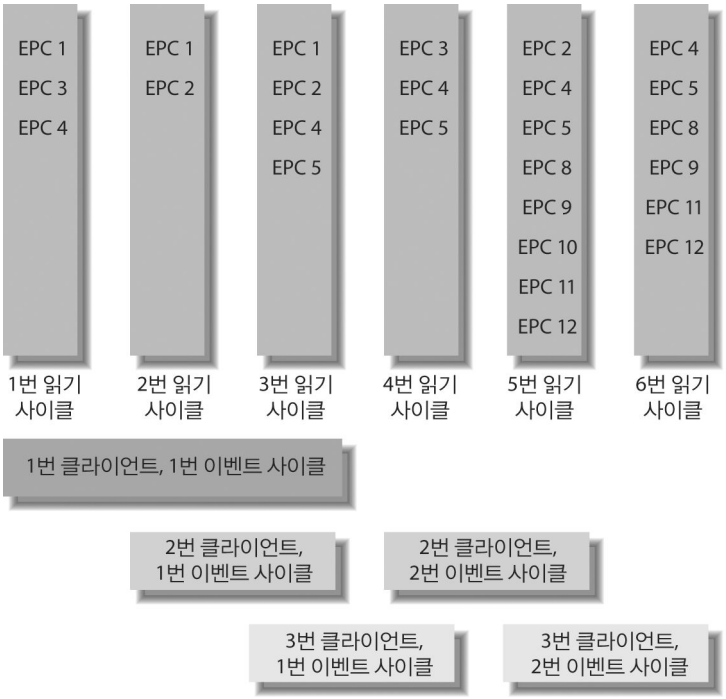
이벤트 사이클

애플리케이션에서 여러 읽기 사이클에 걸쳐서 수집된 정보를 요구할 수도 있다. 예를 들어 재고 관리 시스템에서는 어떤 아이템을 주문해야 할지에 대한 자료를 한 시간마다 갱신해야 할 수 있을 것이다. 결과적으로 일련의 읽기 사이클을 분류할 수 있는 더 높은 수준의 추상화 능력이 필요하다. 바로 이 부분에서 이벤트 사이클(event cycle)이라는 개념이 도입된다. 클라이언트에서 ALE 서비스와 상호 작용을 할 때의 기본 단위가 바로 이벤트 사이클이다.

이벤트 사이클과 읽기 사이클은 꽤 다양한 방식으로 대응될 수 있다. 예를 들어 한 이벤트 사이클이 여러 읽기 사이클에 걸쳐 있을 수도 있고, 일련의 읽기 사이클에 여러 이벤트 사이클이 겹쳐서 대응되는 것도 가능하다. 한 이벤트 사이클이 여러 리더로부터 들어오는 읽기 사이클로 구성될 수도 있다.

[그림 7-4]에 나와 있는 읽기 사이클의 예를 다시 한 번 사용해 보자. 1번 클라이언트의 1번 이벤트 사이클이 1번, 2번, 3번 읽기 사이클에 걸쳐서 적용된다고 해 보자.([그림 7-5] 참조) 클라이언트와 ALE 서비스 사이의 상호 작용에 있어서는 이벤트 사이클이 기본 단위가 되기 때문에 클라이언트에서는 읽기 사이클마다 수집된 EPC에 대한 정보는 알지 못한다. 클라이언트에서는 이벤트 사이클 동안 수집된 EPC 관측 결과의 합집합만을 원하는 상황이 바로 이런 경우에 해당된다. 1번 이벤트 사이클(E1)에서는 S1 U S2 U S3 즉, 다음과 같은 관측 결과를 반환하게 될 것이다.

```
E1 =  S1 U S2 U S3 = {EPC1, EPC2, EPC3, EPC4}
```



[그림 7-5] 이벤트 사이클과 읽기 사이클 사이의 관계

동시에 여러 이벤트 사이클이 활성화될 수도 있고, 서로 다른 이벤트 사이클이 서로 다른 읽기 사이클에서 시작하고 끝날 수 있다. 한 클라이언트에서 여러 요청을 동시에 할 수도 있기 때문에 한 클라이언트에서 여러 이벤트 사이클을 만들어내는 것도 가능하고, 여러 클라이언트에서 동시에 요청을 함에 따라 여러 이벤트 사이클이 만들어질 수도 있다.

이벤트 사이클과 읽기 사이클은 애플리케이션에서 이벤트를 잡아내는 데 있어서 시간 간격 또는 이벤트 윈도우를 지정할 수 있게 해 주는 중요한 개념이다.

이벤트 사이클은 여러 읽기 사이클에 걸쳐서 진행될 수 있기 때문에 애플리케이션 입장에서 더 유의미한 논리적 관측 윈도우를 설정하는 것이 가능해진다.

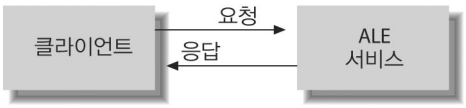
상호 작용 모형

ALE 규격에서 읽기 사이클과 이벤트 사이클을 구분함으로써 얻을 수 있는 유연성에 대한 이해를 바탕으로 클라이언트와 ALE 서비스 사이에서 사용할 수 있는 상호 작용 모형에 대해 알아보자. 애플리케이션 개발자라면 ALE 규격에서 지원하는 상호 작용 모형이 그리 낯설게 느껴지지 않을 것이다. 클라이언트에서는 실시간으로 (동기식) 서비스를 요청하는 방식과

특정 조건이 만족되면 정보를 보내도록 등록을 하는 방식(비동기식) 중에서 원하는 것을 선택할 수 있다.

동기 모드

핵심적인 상호 작용 모형 가운데 하나로 요청/응답 모형을 들 수 있는데, 이 모형에서는 ALE 서비스에 대한 모든 메소드 호출이 동기식으로 처리된다. [그림 7-6]에 동기식 상호 작용 모형이 나와 있다. ALE 규격에서는 동기식 모형으로 즉시 응답과 폴링, 이렇게 두 가지를 지원한다. 이 두 모드가 어떤 식으로 작동하는지는 잠시 후에 조금 더 자세히 알아보도록 하겠다.



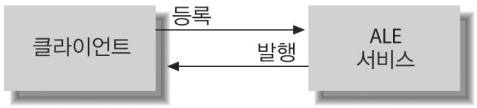
[그림 7-6] 동기식 상호 작용 모형

비동기 모드

ALE 인터페이스에서는 클라이언트가 이벤트에 등록을 하는 비동기식 모형도 지원한다. 이벤트가 발생하면 ALE 서비스에서 비동기식으로 데이터를 클라이언트에 전달한다. 이 모드를 구현할 때는 JMS, TIBCO, MQ-시리즈, 이메일, SOAP 같은 다양한 메시징 기법 가운데 적당한 것을 선택해서 사용하면 된다.

클라이언트에서는 통지 URI를 사용해서 이벤트에 등록한다. 통지 URI는 HTTP, TCP 또는 단순한 파일 형식 등을 바탕으로 구현된다. HTTP 기반 통지 URI에서는 HTTP 규약의 POST 동작을 통해 이벤트 사이클 보고서 전달을 설정한다. TCP 통지 URI를 사용하게 되면 그냥 TCP 연결을 통해 이벤트 사이클 보고서를 전달할 수 있다. 파일 통지 URI를 사용할 때에는 이벤트 사이클 보고서가 파일에 기록된다.

[그림 7-7]에 비동기식 상호 작용 모형이 나와 있다. 이 모형에 대한 내용은 잠시 뒤에 다시 다룬다.



[그림 7-7] 비동기식 상호 작용 모형

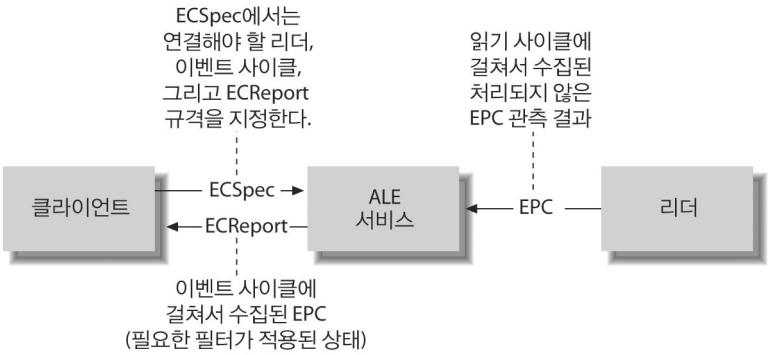
데이터 요소

지금까지 상호 작용 모형에 대해 주로 알아보았다. 이제 각 구성 요소 사이에서 교환되는 핵심 데이터 요소들에 대해 살펴보자.

클라이언트에서 가장 중요한 일 가운데 하나는 EPC 데이터를 요청하는 것이다. 그리고 ALE 서비스에 이벤트 사이클 규격(ECSpec; Event Cycle Specification)을 제공하여 EPC 데이터를 요청한다. ECSpec은 이벤트 사이클을 기술하며 생성되어야 하는 보고서에 대한 규격을 제공한다. ECSpec은 ALE API와 연관된 두 가지 핵심적인 데이터 유형 중의 하나이다(나머지 하나는 ECReport 즉, 이벤트 사이클 보고서(Event Cycle Report)이다). ECSpec은 이벤트 사이클의 시작과 끝을 결정짓는 규칙 및 이벤트 사이클 동안 생성되는 보고서에 대한 규칙을 정해주는 규격이다. 그리고 한 이벤트 사이클 동안 하나 또는 그 이상의 리더의 읽기 사이클에 걸쳐서 데이터를 뽑아오기 때문에 논리적인 리더의 목록도 거기에 포함된다. ECSpec의 예를 들어보면 다음과 같다.

```
ECSpec

readers : List
// 한 이벤트 사이클 동안 하나 또는 그 이상의 리더의 읽기 사이클에 걸쳐서
// 데이터를 뽑아오기 때문에 논리적인 리더의 목록도 여기에 포함된다.

Boundaries : ECBoundarySpec
// 이벤트 사이클의 경계(시작 및 끝)이 어떻게 결정되는지 지정해주는 부분

reportSpecs : List
// 이벤트 사이클이 실행된 후에 반환되어야 할 보고서의 목록을 지정하는 부분

includeSpecInReports : boolean
// true로 설정되어 있으며 ALE를 구현할 때, 보고서에 ECSpec 전체를
// 포함시키도록 한다.

<<확장 지점>>
```

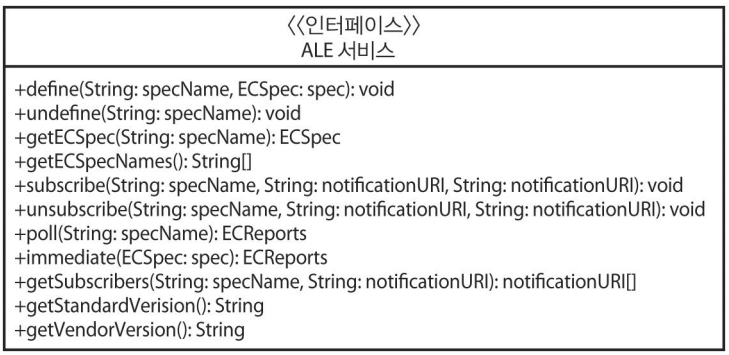
ALE에서 “보고서(report)”는 이벤트 사이클로부터 ECReport 인스턴스 형태로 반환되는 것을 뜻한다. 보고서 규격은 ECReportSpec으로 표현되며, 여기에서는 필터링, 분류 및 기타 데이터 처리 방법 등을 제공한다. [그림 7-8]에 핵심적인 데이터 요소들이 나와 있다. 자세한 내용은 나중에 데이터 모형 섹션에서 살펴보도록 하겠다.



[그림 7-8] 핵심 데이터 요소

ALE 서비스 인터페이스

지금까지는 ALE 규격을 이해하는 데 필요한 핵심 개념 및 용어에 대해 알아보았다. 이제 핵심 ALE 서비스 인터페이스를 살펴보자. EPCglobal ALE 규격에서는 핵심 ALE API를 추상적으로 정의한다. 그 규격에서는 ALE API를 위한 WS-I 호환 SOAP 바인딩도 제공한다. 인터페이스 구조 및 SOAP 바인딩이 WS-I 규격에 호환되도록 구현하기만 하면 ALE 규격에 맞출 수 있다. [그림 7-9]에 ALE 인터페이스가 나와 있다.



[그림 7-9] ALE 서비스 인터페이스

ALE 서비스를 위한 주 인터페이스를 자바 문법으로 표현해 보면 다음과 같다(생산업체마다 구현 방법이 조금씩 다를 수 있다).

```
public interface ALE {
    public void define(String specName, ECSpec spec) throws
DuplicateNameExpcetion,ECSpecValidationException,
SecurityException, ImplementationException;
    // ECSpec 정의
```

```
    // 다음과 같은 상황에서 ECSpecValidationException이 발생됨:
    // - readers 매개변수가 정의되지 않았거나 알 수 없는
    //   논리 리더 이름이 포함되어 있을 경우
    // - reportSpecs 매개변수가 정의되지 않았거나
    //   널이거나 중복된 ECReport 이름이 포함되어 있을 경우

    public void undefine(String specName) throws NoSuchNameException,
SecurityException, ImplementationException;
    // specName을 정의 해제

    public ECSpec getECSpec(String specName) throws
NoSuchNameException, SecurityException, ImplementationException;
    // 지정된 규격 이름에 대응되는 이벤트 사이클 규격을 반환

    public String[] getECSpecNames( ) throws SecurityException,
ImplementationException;
    // ALE 서비스에서 정의된 모든 이벤트 사이클 규격 이름을 반환

    public void subscribe(String specName, String notificationURI)
throws NoSuchNameException,invalidURIException,
DuplicateSubscriptionException, SecurityException,
ImplementationException;
    // 지정된 이벤트 사이클마다 반환되는 이벤트에 대한
    // 보고서를 받을 수 있도록 등록

    public void unsubscribe(String specName, String notificationURI)
throws NoSuchNameException,invalidURIException,
DuplicateSubscriptionException, SecurityException,
ImplementationException;
    // 지정된 이벤트 사이클 규격에 대한 등록 해지

    public ECReports poll(String specName) throws
NoSuchNameException,SecurityException, ImplementationException;

    public ECReports immediate(ECSpec spec) throws
ECSpecValidationException, SecurityException,
ImplementationException;
    // 다음과 같은 상황에서 ECSpecValidationException이 발생됨:
    // - readers 매개변수가 정의되지 않았거나 알 수 없는
    //   논리 리더 이름이 포함되어 있을 경우
    // - reportSpecs 매개변수가 정의되지 않았거나
    //   널이거나 중복된 ECReport 이름이 포함되어 있을 경우
```

```
    public notificationURI[]  getSubscribers(String specName) throws
NoSuchNameExpcetion, SecurityException, ImplementationException;
    // 등록자 URI의 목록 반환

    public String getStandardVersion(  ) throws SecurityException;
    // ALE 규격의 버전 번호 반환

    public String getVendorVersion(  ) throws SecurityException;
    // 생산업체 쪽의 버전 번호 반환

}
```

사용 시나리오

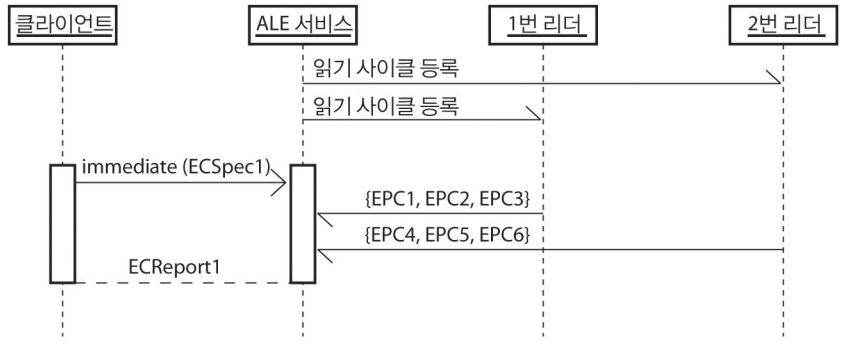
앞에서 논의했듯이 ALE 규격에서는 클라이언트와 서비스 사이에서 비동기식 및 동기식 상호 작용 모형을 모두 지원한다. 지금까지 ALE 인터페이스 API에 대해 배운 것을 바탕으로 그런 상호 작용 모형이 어떤 식으로 사용되는지 살펴보자. 그 과정에서 ALE 구조의 몇몇 구성 요소에 대해 더 깊이 이해할 수 있는 기회도 있을 것이다.

동기 모드

우선 클라이언트에서 일련의 리더로부터 들어오는 이벤트를 한 번 요청하는 방법, 그리고 그 과정에서 처리되지 않은 EPC 관측 결과를 필터링하고 분류하는 방식을 지정하는 방법에 대해 알아보도록 하자. ALE 규격에서는 이런 상호 작용 모드를 "즉시(immediate)" 모드라고 부른다.

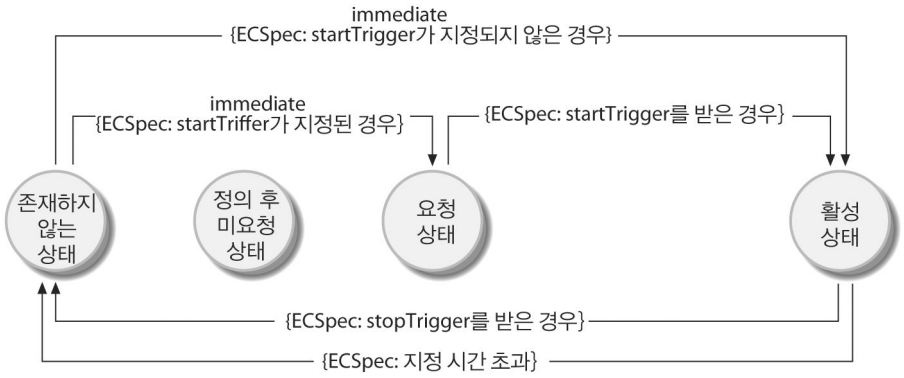
즉시 모드. 즉시 모드는 ALE 서비스에 접근하기 위해 사용할 수 있는 두 가지 동기식 방법 가운데 하나이다. 우선 클라이언트에서 ECSpec을 생성하고 설정하고 나서 ALE 서버의 "즉시" 서비스를 호출한다. ALESpec에서는 즉시 서비스의 매개 변수로 전달되는 ECSpec을 검사한다. 조금 전에 "데이터 요소" 절에서도 설명했듯이, ECSpec에는 클라이언트에서 이벤트를 받고자 하는 리더들의 목록이 지정되어 있다. 그리고 이벤트 사이클 경계(경계 객체로 지정)도 지정되어 있으며, ECReportSpec 즉, 보고서 규격에서 지정한 리더에서 수집된 처리되지 않은 EPC 관측 결과를 필터링하고 분류하는 메커니즘도 지정되어 있다. ECSpec에는 보고서 규격을 여러 가지 포함시킬 수 있기 때문에 클라이언트에서는 똑같은 관측 결과들을 여러 방식으로 필터링 및 분류하는 것이 가능하다. 예를 들어 면도기용 EPC와 면도크림용 EPC를 따로 분류하고 싶다면 ALE 서비스 측에 그런 기준을 요청할 수 있다. 필터링 및 분류 기능에 대해서는 나중에 더 자세히 알아보도록 하자.

[그림 7-10]에 클라이언트에서 즉시 서비스를 호출했을 때 일어나는 일이 그림으로 나와 있다. 이 요청은 ALE 서비스에 의해 지원되며, ALE 서비스에서는 필요한 리더에 적절한 읽기 사이클만큼 등록을 하고 리더로부터 이벤트 데이터를 받아온다. 그리고 그 데이터를 처리한 다음 그 결과를 반환한다.



[그림 7-10] 즉시 모드 절차도

지금까지 쭉 살펴봤듯이 ECSpec에서는 리더로부터 아무 처리도 거치지 않은 관측 결과를 가져오고, 필터 처리를 하고, 데이터 집합으로 분류하고, 마지막으로 보고서로 포장하기 위한 규칙을 제공한다. 이제 클라이언트에서 즉시 모드로 처리할 ECSpec을 제출했을 때 상태가 어떤 식으로 전환되는지 살펴보자. [그림 7-11]에 나와 있는 것처럼 ECSpec에는 세 가지 상태가 포함되어 있다.



[그림 7-11] ECSpec 상태 모형 – 즉시 모드

■ 정의 후 미요청(Defined but unrequested) 상태

ALE 서비스에서 정의된 메소드를 통해 생성되고 특정 ECSpecName(문자열)과 연계되었을 때 이 상태로 들어간다. 즉시 모드는 데이터를 한 번만 요청하기 위한 용도로 사용된다. 따라서 이 모드에서는 ECSpec을 나중에 다시 사용하기 위해 논리적인 이름(ECSpecName)을 할당할 필요가 없다. 폴링 및 비동기 모드에서 정의 후 미요청 상태를 사용한다.

■ **요청(Requested) 상태**

ECSpec이 ALE 서비스에서 처리되기를 기다리고 있는 상태를 요청 상태라고 부른다.

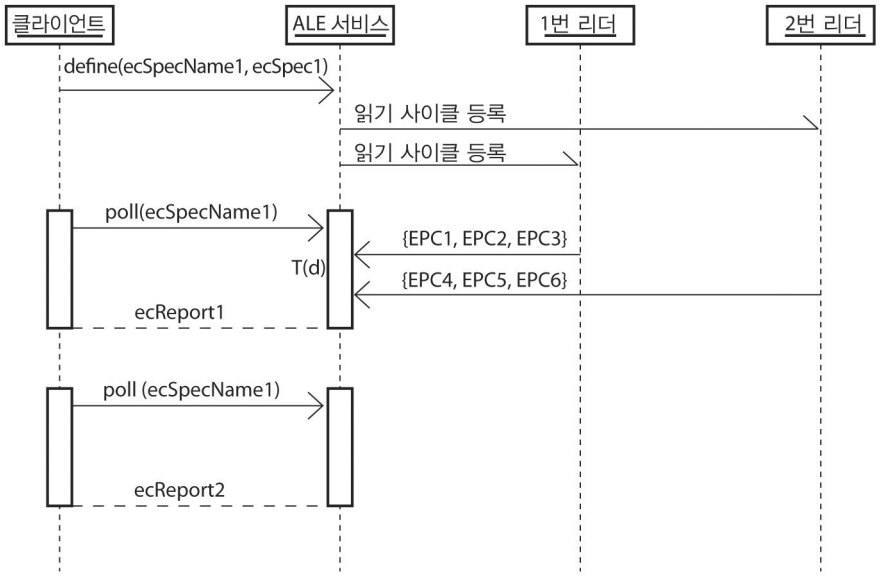
■ **활성(Active) 상태**

ALE 서버에서 주어진 이벤트 사이클에 맞춰서 리더로부터 처리되지 않은 EPC 관측 결과를 받아들이고 있으면 그 ECSpec은 활성 상태에 있다고 부른다.

클라이언트에서 즉시 모드를 요구하는 immediate 메소드를 호출할 때 매개 변수로 전달되는 ECSpec은 그 경계 규격(ECBoundarySpec)에 올바른 시작 트리거(ECSpec.ECBoundarySpec.startTrigger)가 포함되어 있기만 하면 요청 상태에 있는 것으로 볼 수 있다. 즉, ALE 서버에서 ECSpec을 바탕으로 데이터를 수집하기 시작하려면 시작 트리거가 발동되어야 한다. 이 트리거는 다른 센서에 의해서 만들어질 수도 있다(예를 들어 컨베이어 벨트에서 팔레트에 물건을 실을 때 사용하는 센서 등을 생각할 수 있다). 적절한 정지 트리거(ECSpec.ECBoundarySpec.stopTrigger 속성으로 지정됨)가 들어오면 ALE 서버는 ECSpec에서 지정한 리더로부터 데이터를 수집하는 일을 멈추고 필터링 및 분류가 끝난 데이터를 클라이언트로 보낸다.

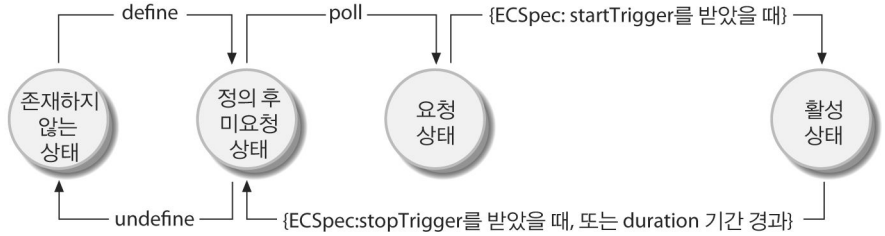
시작 트리거가 지정되어 있지 않으면 ECSpec은 활성 상태에 있는 것으로 간주된다. 이 경우에는 이벤트 수집을 위해 지정된 시간(ECSpec.ECBoundarySpec.duration)이 경과되면 ALE 서버에서 ECSpec에서 지정한 리더로부터 데이터를 수집하는 일을 멈추고 필터링 및 분류가 끝난 데이터를 클라이언트로 보낸다.

폴링 모드. EPC 데이터에 대해서 산발적으로 한 번씩만 보고서를 받는 것이 아니라 미리 정해진 조건에 따라 정기적으로 업데이트를 받고 싶다면 ALE 서버의 폴링(polling) 인터페이스를 사용해야 한다. 폴링은 동기식으로 작동된다. 어떤 면에서 보면 등록을 한 후에 이벤트 사이클이 생성되고 나면 등록을 해지하는 것과 비슷하다고 볼 수 있다. [그림 7-12]에 나와 있는 것처럼 클라이언트에서는 우선 ECSpec을 생성한 다음 ALE 서비스의 define 메소드를 호출해서 논리명을 할당한다. 일단 ECSpec이 정의되고 나면 클라이언트에서는 poll 메소드를 호출할 수 있다. ALE 서비스에서는 poll 메소드를 처리하기 위해 ECSpec을 점검하고 이벤트 경계, 논리 리더 등을 결정한다. (앞에서 설명한 즉시 모드의 경우하고 비슷한 과정을 거침) 이벤트 사이클이 끝나면 ALE 서비스에서는 요청받은 EPC 데이터가 들어있는 보고서를 반환한다.



[그림 7-12] 폴링 모드 절차도

클라이언트에서 폴링을 할 때 ECSpec의 상태가 어떤 식으로 전환되는지 살펴보자. 클라이언트에서 define 메소드를 호출하면 ECSpec은 정의 후 미요청 상태가 된다. ALE 서비스에 주어진 이벤트 사이클 규격에 따라 폴링을 해 달라는 요청이 들어가면 시작 트리거에 의해 리더 관측 결과를 기록하고, 클라이언트에서 요청한 필터링 및 분류 메커니즘에 따라 그 관측 결과를 처리하는 작업이 시작되기를 기다린다(물론 이 모든 절차가 이벤트 사이클 규격에 정의되어 있다). 이 기간 동안에는 ECSpec이 활성 상태가 된다. 폴링 시간이 지나고 나면 ECSpec은 다시 정의 후 미요청 상태로 돌아간다. 클라이언트에서 이 규격을 바탕으로 하는 데이터를 추가로 원한다면 [그림 7-13]에 있는 것처럼 폴(poll) 메소드를 다시 호출해야 한다.

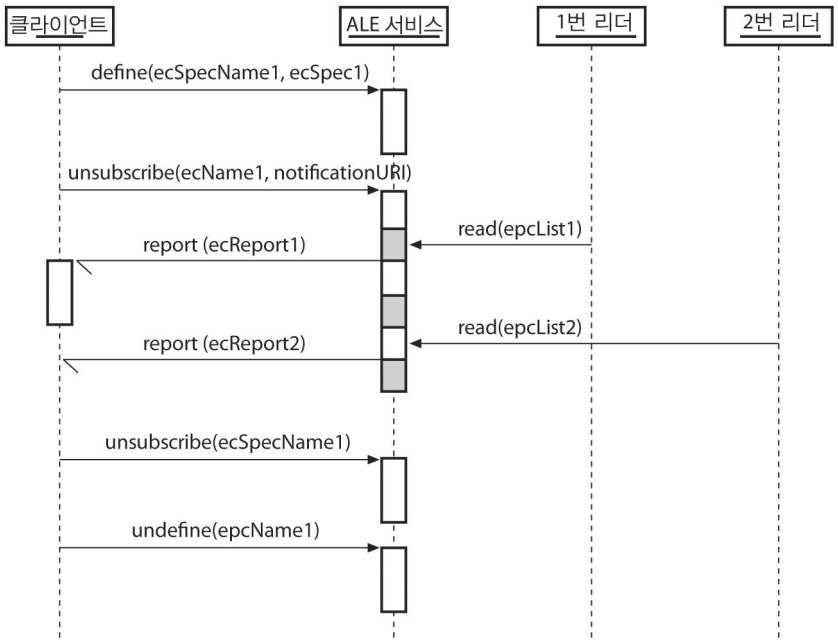


[그림 7-13] ECSpec 상태 모형 – 폴링 모드

비동기 모드

비동기 ALE 서비스 상호 작용 모드에서는 다른 비동기식 메시징 아키텍처에서와 마찬가지로 전통적인 발행/등록(publish/subscribe) 메커니즘을 사용한다. [그림 7-14]에 나와 있는

것처럼 클라이언트에서는 이벤트 규격을 정의한 후 등록을 하여 정기적으로 갱신되는 데이터를 받는다. 일단 등록을 하면 정기적으로 ALE 서비스로부터 필터링 및 분류가 끝난 EPC 데이터를 받을 수 있다. 더 이상 갱신된 내용을 받고 싶지 않으면 등록을 해지하면 된다.



[그림 7-14] 비동기 상호 작용 모형

ALE 규격에 따르면 비동기 상호 작용 모형에서 보고서를 HTTP, TCP, 그리고 파일 직접 기록, 이렇게 세 가지 방식으로 발행할 수 있다.

발행 방식은 클라이언트에서 접속할 때 등록하는 URI의 종류에 따라 결정된다. 각 통지(notification) URI에 대해 간단하게 설명해 보면 다음과 같다.

■ **HTTP 통지 URI**

HTTP 통지 URI를 이용하면 ECReport를 HTTP 규약을 통해 발행할 수 있다. 이 보고서는 HTTP POST 방식으로 XML 형식으로 전송된다. HTTP URI 주소는 다음과 같은 식으로 구성된다.

```
http://호스트명:포트번호/URL의 나머지 부분
http://호스트명/URL의 나머지 부분
```

"호스트명"은 등록한 쪽의 DNS 이름 또는 IP 주소이고 "포트번호"는 등록자가 리스닝하고 있는 TCP 포트 번호이다. 포트 번호를 지정하지 않으면 80번 포트가 기본으로 사용된다. "URL의 나머지 부분"은 그 서버에서 HTTP POST를 처리할 수 있는 자원에 연결할 수 있는 주소의 나머지 부분이다.

■ **TCP 통지 URI**

TCP 통지 URI를 이용하면 TCP 연결을 통해 ALE 서버에 등록할 수 있다. 이 경우에도 ECReport는 XML 형식으로 전달된다. TCP 통지 URI는 다음과 같은 식으로 구성된다.

```
tcp://호스트명:포트번호
```

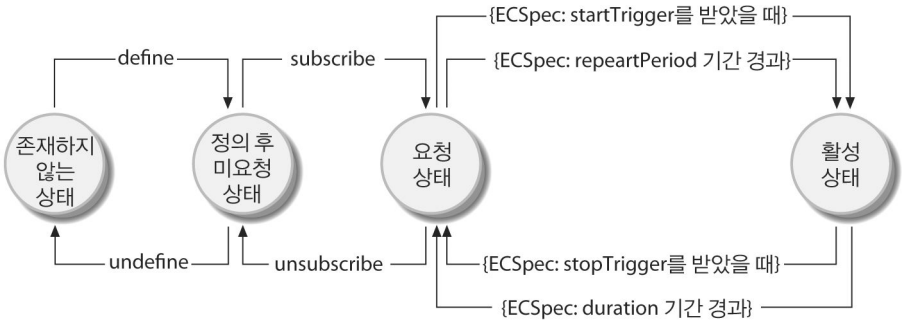
■ **파일 통지 URI**

클라이언트에서 파일 통지 URI를 사용하여 등록하면 ALE에서는 ECReport를 XML 형식의 파일로 발행한다. ALE 서비스에서는 이벤트 사이클이 넘어갈 때마다 그 결과를 지정된 파일에 덧붙인다. 파일 통지 URI는 다음과 같은 식으로 구성된다.

```
file://호스트명/경로
file://경로
```

전과 마찬가지로 "호스트명"은 파일 시스템에 보고서를 발행할 컴퓨터의 이름이다. 경로는 호스트의 파일 시스템에서 파일을 저장할 경로명을 뜻한다.

[그림 7-15]에 비동기 모드의 ECSpec의 상태 모형이 나와 있다. 클라이언트에서 유효한 ECSpec에 등록하면 이벤트 규격은 요청 상태로 들어간다. 일단 ECSpec이 요청 상태로 들어가면 ALE 서비스에서는 시작 트리거(ECSpec.ECBoundarySpec.startTrigger)를 받았을 때 또는 이전 이벤트 사이클의 후로 반복 기간이 경과되었을 때 EPC 데이터를 수집하고 처리하는 작업을 시작한다. 정지 트리거(ECSpec.ECBoundarySpec.stopTrigger)를 받거나 이벤트 사이클이 종료(ECSpec.ECBoundarySpec.duration)되면 ECSpec은 다시 요청 상태로 돌아간다.



[그림 7-15] ECSpec 상태 모형 – 비동기 모드

필터링 및 분류

클라이언트에서 처리되지 않은 EPC 관측 결과를 보고서에 수록하기 전에 어떤 식으로 처리해야 하는지 지정할 수 있다. ALE 규격에서는 필터링(filtering)과 분류(grouping), 이렇게

두 가지 이벤트 처리 메커니즘을 제공한다. 필터링은 이벤트 데이터의 특정 패턴을 기준으로 잡을 수 있는 기능을 제공하고, 분류는 서로 다른 리더로부터 여러 이벤트 사이클에 걸쳐서 수집한 데이터를 분류하는 기능을 제공한다. 필터링용 패턴과 분류용 패턴은 각각 ECFilterSpec과 ECGroupSpec을 통해 제공된다. ECSpec에는 여러 보고서 규격을 들어갈 수 있으며, 클라이언트에서 각 보고서 규격(ECReportSpec)마다 별도의 필터 규격(ECFilterSpec)과 분류 규격(ECGroupSpec)을 지정할 수 있다.

```
ECReportSpec

reportName : string
reportSet : ECReportSetSpec
filter : ECFilterSpec
group : ECGroupSpec
output : ECReportOutputSpec
reportIfEmpty : boolean
reportOnlyOnChange : boolean
<<확장 지점>>
```

ECFilterSpec으로 제공되는 필터는 리더로부터 받아온 처리되지 않은 EPC 관측 결과에 대해 적용되며, 분류 패턴이 적용되기 전에 적용된다.

```
ECFilterSpec

includePatterns : List
excludePatterns : List
<<확장 지점>>
```

ECGroupSpec은 필터링이 끝난 EPC를 보고서에 수록할 때 분류하는 방법을 지정해준다.

```
ECGroupSpec

patternList : List

<<확장 지점>>
```

필터링

클라이언트는 includePatterns와 excludePatterns, 이렇게 두 가지 패턴 목록을 가지고 필터링 조건을 지정할 수 있다. includePatterns 목록에 있는 패턴 중 적어도 하나에 매치되면서 excludePatterns 목록에 있는 패턴에는 하나도 매치되지 않는 EPC만 최종 보고서에 올라갈 수 있다. 형식적으로 정의를 해 보자면 다음과 같은 표현식을 쓸 수 있다.

```
F(R) = { epc | epc ? R &
(epc ? includePattern1 | epc ? includePattern2 |  ...  | epc ?
includePatternN) &
(epc ? excludePattern1 & epc ? excludePattern2 &  ...  & epc ?
excludePatternN)}
```

하나의 필터링 패턴으로 단일 EPC 또는 여러 EPC들을 골라낼 수 있으며, 필터링 패턴은 URI 문자열 형식으로 만들어진다. 필터링 패턴의 일반적인 형식은 다음과 같다.

```
urn:epc:pat:TagFormat:<데이터 필드>
```

TagFormat은 EPCglobal의 태그 데이터 규격(Tag Data Specification; 태그 형식에 대한 내용은 4장에 나와 있음)으로 정의된 태그 형식을 나타낸다. 데이터 필드의 일반적인 표현 방법은 아래에 나와 있지만 어떤 태그 형식을 쓰는지에 따라 이 필드가 달라질 수도 있다.

```
urn:epc:pat:TagFormat:FilterValue.CompanyPrefix.ItemReference.
SerialNumber
```

FilterValue.CompanyPrefix.ItemReference.SerialNumber 필드는 EPC의 데이터 필드에 대응된다. 어떤 TagFormat이 사용되는지에 따라 EPC를 지정하기 위해 사용하는 필드 수가 달라지며, 그 의미도 달라진다.

예를 들어, GID-96 형식에는 FilterValue 데이터 필드가 빠져 있어서 EPC가 다음과 같은 식으로 표현된다.

```
urn:epc:pat:GID-96:CompanyPrefix.ItemReference.SerialNumber
```

각 데이터 필드(Filtervalue.CompanyPrefix.ItemReference.SerialNumber)는 10진수, 별표/와일드카드, 또는 10진수 범위(low-high 형태)로 지정할 수 있다.

[표 7-1]에 몇 가지 필터의 예가 나와 있다. 대부분의 프로그래머들은 태그 형식이나 필터링 옵션에 대한 자세한 내용에 대해서는 신경 쓸 일이 없기 때문에 여기에서는 전반적인 개념에 대해서만 살펴보았다.

필터에 대한 자세한 내용은 4장에서 살펴보도록 하자. 다양한 태그 형식과 형식별로 사용할 수 있는 필터에 대해 자세하게 알고 싶다면 EPCglobal의 태그 데이터 규격에 대한 문서를 찾아서 읽어보면 도움이 될 것이다.

[표 7-1] GID-96 태그 형식에서 사용하는 필터의 예

필터	사용하는 유형	설명
urn:epc:pat:gid-96:18.200.3000	십진수	기업 접두어가 18, 아이템 레퍼런스가 200, 시리얼 번호가 3000인 EPC를 반환
urn:epc:pat:gid-96:18.200.*	와일드카드, 십진수	기업 접두어가 18, 아이템 레퍼런스가 200인 EPC를 시리얼 번호와 무관하게 모두 반환
urn:epc:pat:gid-96:18.[190-200].*	범위, 와일드카드	기업 접두어가 18이고 아이템 레퍼런스가 190에서 200 범위에 있는 EPC를 시리얼 번호와 무관하게모두 반환
urn:epc:pat:gid-96:*.*.*	와일드카드	GID-96 태그 형식을 가지는 모든 EPC를 반환

분류

분류 패턴에서도 앞에서 필터링에 대해 설명한 것과 비슷한 메커니즘을 사용하지만 십진수, 와일드카드(*), 범위 지정 외에 URI 필드에서 특별한 값으로 X를 사용할 수 있다. [표 7-2]에 분류 패턴을 지정할 때 사용할 수 있는 매개 변수들이 나와 있다.

[표 7-2] 분류 패턴에서 사용할 수 있는 URI 필드 값

패턴 URI 필드	설명
*	이 필드에 있는 모든 값을 한 그룹으로 분류
X	이 필드에 값에 따라 서로 다른 그룹으로 분류
숫자	이 필드에 지정된 숫자가 들어있는 EPC만 이 그룹으로 분류
범위[Low-high]	이 필드의 값이 주어진 범위에 속하는 EPC는 모두 이 그룹으로 분류

[표 7-3]에 분류 패턴의 예가 몇 가지 나와 있다.

[표 7-3] 분류 패턴의 예

패턴 URI	설명
urn:epc:pat:sgtin-64:*.*.*.*	모든 EPC를 한 그룹으로 분류
urn:epc:pat:sgtin-64:X.*.*.*	필터 값별로 분류
urn:epc:pat:sgtin-64:*.*.X.*	아이템 유형별로 분류
urn:epc:pat:sgtin-64:X.X.*.*	필터 값과 기업 접두어별로 분류
urn:epc:pat:sgtin-64:3.*.*.[0-100]	필터 값이 3이고 시리얼 번호가 0~100 범위에 있는 것을 모두 같은 그룹으로 분류
urn:epc:pat:sgtin-64:3.X.*.[0-100]	기업별로 분류하되 필터 값이 3이고 시리얼 번호가 0~100 범위에 있는 모든 EPC를 포함시킴

ALE 규격에 의하면 한 이벤트 사이클 동안 받아서 필터링을 마친 모든 EPC는 정확하게 한 그룹에 속해야 한다. 모든 EPC가 한 그룹씩에만 속하게 되도록 분류용 패턴의 결과가 서로 겹치지 않으면서 어떤 그룹에도 속하지 않는 EPC도 없어야 한다.

데이터 모형

이 절에서는 ALE 규격의 중요한 데이터 요소에 대해 살펴보자. ALE 호환 미들웨어를 사용하여 프로그래밍을 할 일이 없다면 이 절은 그냥 건너뛰어도 무방하다.

[그림 7-16]은 ALE 규격에 있는 중요한 데이터 유형을 정리해 놓은 다이어그램이다. 그림에 나와 있듯이, ECSpec에서는 클라이언트에서 받고 싶어 하는 데이터를 지정할 수 있는 방법을 제공한다. 클라이언트에서 데이터를 받고자 하는 리더 이름을 지정할 수 있고, 이벤트 사이클이 리더의 읽기 사이클에 어떤 식으로 대응되는지도 지정할 수 있다.

이벤트 경계는 ECBoundarySpec 데이터 유형을 사용하여 지정한다. 리더에서 오는 관측 결과를 필터링(ECFilterSpec)하고 분류(ECGroupSpec)하는 방법은 ECReportSpec으로 제공된다. 어떤 데이터를 어떻게 보고해야 할지 지정하는 일도 ECReportSpec을 통해서 한다.

ALE 서버에서 만들어내는 보고서는 ECReport에 의해 정의된다. ECReport는 하나의 이벤트 사이클에 의해 만들어지는 단일 보고서를 제공한다. ECReport에 들어가는 데이터는 ECReportGroup 인스턴스를 사용하여 분류한다.

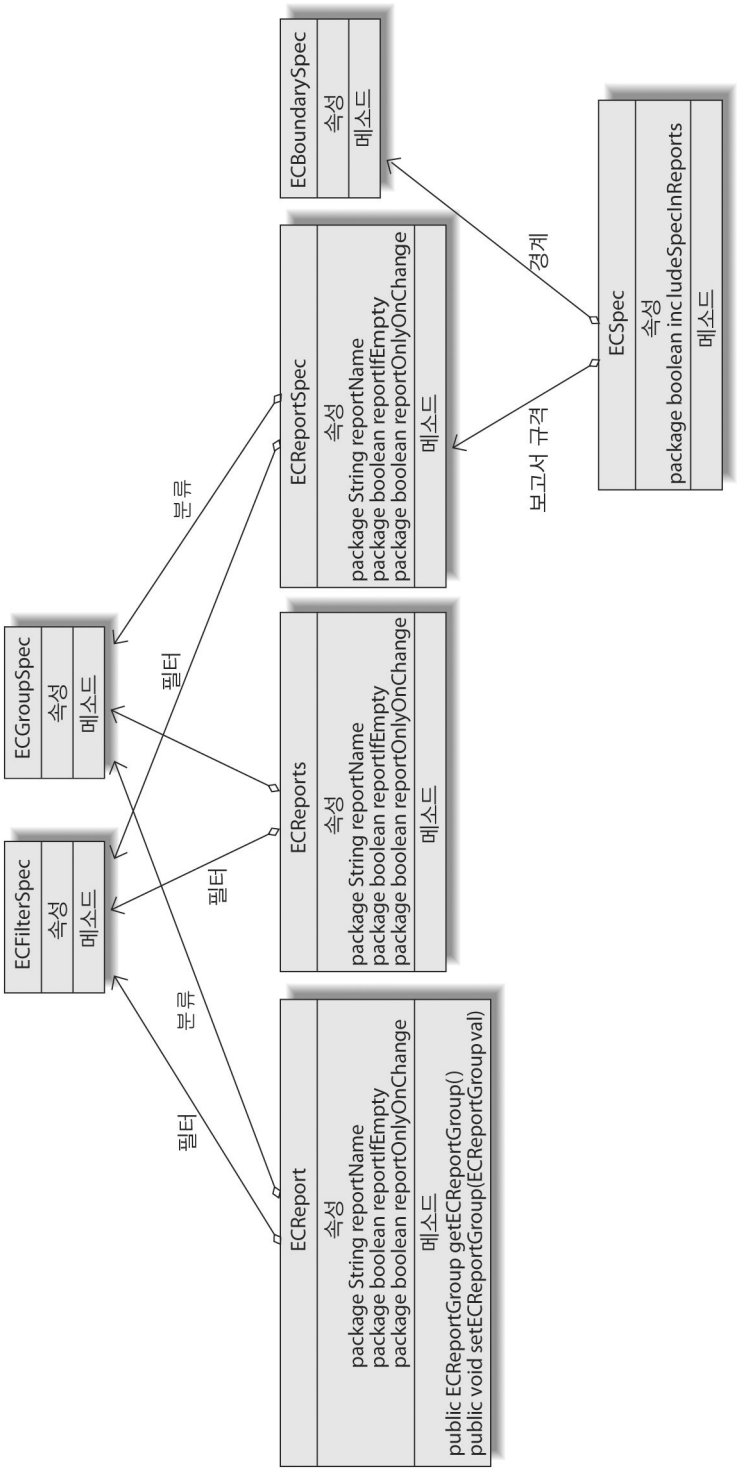
ECReportGroup의 각 인스턴스는 EC Report에 들어있는 EPC의 각 그룹을 나타낸다. EPC는 16진수 또는 10진수 형태로 보고된다.

ECRerportGroupMember 형식은 EPC가 어떤 식으로 보고되는지에 대한 템플릿을 제공한다.

ALE 규격은 각 생산업체별로 확장할 수도 있고, EPCglobal에서 차기 버전을 통해서 확장하는 것도 가능하다.

데이터 형식을 정의하는 부분에서 확장할 수 있는 부분은 <<확장 지점>>이라고 표시해 놓았다.

ALE에서 사용되는 각 요소에 대해 살펴보자.



[그림 7-16] ALE 관련 클래스 다이어그램

ECSpec

이벤트 사이클 동안 하나 이상의 보고서가 만들어진다. 이렇게 만들어지는 보고서의 규격은 ECSpec으로 기술된다.

```
readers : List
// 논리 리더의 목록. 이벤트 사이클 동안 하나 이상의 리더의 읽기 사이클
// 로부터 데이터를 가져와야 한다.

Boundaries : ECBoundarySpec
// 이벤트 사이클의 시작과 끝을 결정하는 방법을 기술하는 부분

reportSpecs : List
// 이벤트 사이클이 실행된 후에 반환되어야 하는 보고서의 목록

includeSpecInReports : boolean
// 참이면 생성되는 보고서에 전체 ECSpec이 포함된다.

<<확장 지점>>
```

ECSpec은 ALE API와 관련된 두 가지 핵심 데이터 유형 가운데 하나이다(나머지 하나는 ECReport임). 이벤트 사이클의 시작과 끝을 결정하기 위한 규칙과 그 이벤트 사이클로부터 만들어야 하는 보고서를 ECSpec을 통해서 지정한다. 또한 이벤트 사이클 동안 하나 이상의 리더의 읽기 사이클로부터 데이터를 가져와야 하므로 논리 리더명 목록도 여기에 들어있어야 한다.

ECBoundarySpec

ECBoundarySpec은 이벤트 사이클의 시작과 끝을 기술하기 위한 데이터이다.

```
startTrigger : ECTrigger
stopTrigger : ECTrigger
repeatPeriod : ECTime
duration : ECTime
stableSetInterval : ECTime
<<확장 지점>>
```

이벤트 사이클은 다음 중 한 가지 조건이 만족되면 시작된다.

- ECSpec이 요청 상태인 동안 지정된 시작 트리거가 들어왔을 때

- 지난 번 이벤트 사이클이 시작된 후로 반복 기간(repeatPeriod)이 경과되었고 ECSpec이 여전히 요청 상태에 있을 때

다음 중 한 가지 조건이 만족되면 이벤트 사이클이 종료된다.

- 경과 시간(duration) 필드에서 지정한 시간 간격만큼 시간이 경과되었을 때
- 정지 트리거가 들어왔을 때
- ECSpec이 정의 후 미요청 상태로 바뀔 때

ECTime

ECTime은 시간 간격을 정의하기 위한 용도로 사용된다.

```
duration : long
unit : ECTimeUnit
```

ECTimeUnit

ECTimeUnit은 ECBoundarySpec에서 사용되는 시간의 물리적인 시간 단위를 지정하기 위한 용도로 사용되며, 열거형이다.

```
<< 열거형>>

MS  // 밀리초
```

ECTrigger

ECTrigger라는 URI는 이벤트 사이클의 시작 또는 정지 트리거를 보여준다. 이 URI를 해석하는 방법은 ALE 구현 방식에 따라 결정된다.

```
triggerValue: URI
```

ECReportSpec

ECReportSpec은 한 이벤트 사이클 실행 결과로 반환되는 보고서를 기술하며, 보고해야 할 EPC에 대한 규칙을 제공한다. 현재 읽은 EPC를 모두 보고해야 할지, 또는 기존 이벤트 사이클에 추가된 내용, 또는 이전 사이클에서 빠진 내용만 보고할지 등을 그 규칙으로 지정한다.

```
reportName : string
// 생성할 ECReport의 이름. 이벤트 사이클이 끝날 때 만들어지는
// ECReport 인스턴스에 이 문자열이 그대로 복사된다.

reportSet : ECReportSetSpec
// 어떤 EPC 집합이 필터링용으로 입력되어야 하는지 지정하는 부분
// ECReportSpec은 열거형으로, 다음 중 한 가지 값을 가진다
// {CURRENT, ADDITIONS, DELETIONS}
// CURRENT: 현재 읽은 모든 EPC
// ADDITIONS: 이전에는 없었고 이번에 새로 등장한 EPC
// DELETIONS: 이전에는 있었지만 이번에는 없어진 EPC

filter : ECFilterSpec
// 리더 관측 결과를 필터링하는 방법을 지정하는 부분

group : ECGroupSpec
// 필터링된 EPC를 분류하는 방법을 지정하는 부분

output : ECReportOutputSpec
// 필터링 및 분류가 끝났을 때 EPC를 보고하는
// 방법을 지정하는 부분

reportIfEmpty : boolean
// 아무 값이 없어도 보고서를 만들어야 하는지를
// 지정하는 부분 (true/false)

reportOnlyOnChange : boolean
// 보고서에 들어있는 값들이 전과 다를 때만 보고서를
// 보낼지를 지정하는 부분 (true/false)

<<확장 지점>>
```

ECTerminationCondition

ECTerminationCondition이란 이벤트 사이클이 어떻게 끝나는지를 지정하기 위한 열거형을 말한다.

```
<<열거형>>

TRIGGER
// 정지 트리거가 들어왔을 때 이벤트 사이클이 종료됨
```

```
DURATION
// 지정된 시간이 경과되면 이벤트 사이클이 종료됨

STABLE_SET
// 지정된 시간동안 EPC들이 안정되게 관측되면
// 이벤트 사이클이 종료됨


UNREQUEST
// 요청/등록 클라이언트가 없으면
// 이벤트 사이클이 종료됨
```

ECReport

한 이벤트 사이클로 만들어지는 단일 보고서로서, ECReport에 들어가는 데이터는 ECReportGroup 인스턴스로 분류된다.

```
reportName : string
// ECReportSpec의 reportName 필드가 보고서 이름으로 쓰인다.

group : List
// ECReportGroup 인스턴스의 목록

<<확장 지점>>
```

ECReports

이벤트 로그로부터 출력되는 것은 ECReports로 기술된다.

```
specName : string
date : dateTime
ALEID : string
group : ECGroupSpec
totalMilliseconds : long
terminationCondition : ECTerminationCondition
spec : ECSpec
reports : List
<<확장 지점>>
```

ECReportSetSpec

필터링 및 출력용으로 쓰일 EPC 집합을 보여주는 열거형이다.

```
<<열거형>>
CURRENT
ADDITIONS
DELETIONS
```

ECReportOutputSpec

이벤트 사이클 보고서의 레이아웃을 제공한다.

```
includeEPC : boolean
includeTag : boolean
includeRawHex : boolean
includeRawDecimal : boolean
includeCount : boolean

<<확장 지점>>
```

ECReportGroup

ECReport에 들어가는 각 그룹을 나타낸다.

```
groupName : string
groupList : ECReportGroupList
groupCount : ECReportGroupCount

<<확장 지점>>
```

ECReportGroupList

ECReportOutputSpec의 includeEPC, includeTag, includeRawHex, includeRawDecimal 매개 변수 중 하나라도 참이면 ECReportGroup에 ECReportGroupList가 포함된다.

```
members : List  // ECReportGroupListMember 인스턴스의 목록

<<확장 지점>>
```

ECReportGroupListMember

ECReportGroupListMember를 이용하면 여러 EPC 형식이 보고서에 들어갈 수 있다. ECReportGroupListMember에 들어가는 URI는 ECReportOutputSpec의 부울 값에 대응된다. 예를 들어, ECReportOutputSpec의 includeEPC 속성이 참이면 epc의 URI 값은 널이 아니어야 한다.

```
epc : URI
tag : URI
rawHex : URI
rawDecimal : URI

<<확장 지점>>
```

ECReportGroupCount

ECReportGroupCount는 ECReportGroup에 들어간다. ECReportGroup을 가져오려면 그에 대응되는 ECReportOutputSpec의 includeCount가 참이어야 한다.

```
count : int

<<확장 지점>>
```

ECFilterSpec

EPC 패턴의 최종 목록에 어떤 EPC가 들어가야 할지 기술한다.

```
includePatterns : List
excludePatterns : List

<<확장 지점>>
```

ECGroupSpec

필터링된 EPC 및 보고서용으로 어떤 식으로 분류되는지를 정의한다.

```
patternList : List
```

상용 RFID 미들웨어

이미 다양한 이벤트 관리용 미들웨어가 시중에 출시되어 있다. 그 중에는 EPCglobal에서 제안한 ALE 규격을 기반으로 한 것도 있고, ALE가 나오기 전에 출시되긴 했지만 비슷한 이벤트 관리 기능을 제공하는 것도 있다. 이 절에서는 판매되고 있는 상용 미들웨어 중 몇 가지에 대해 훑어보도록 하겠다. 물론 여기에서 소개하는 제품들은 그냥 몇 가지 대표적인 상품들일 뿐, 모든 제품들을 여기에 수록하지는 못했다. 여기에서는 주로 정보가 얼마나

공개되어 있는지, 그리고 업체로부터 제품에 대한 설명을 책에 수록해도 되는지 물어본 결과를 바탕으로 소개할 제품을 골랐다. 포레스터를 비롯한 몇몇 애널리스트들이 정기적으로 RFID 미들웨어 평가 결과를 내놓기 때문에 실제 제품을 구입하고자 하는 독자들은 그런 정보들을 바탕으로 꼼꼼히 따져보는 것이 좋을 것이다. 여기에서는 상용으로 어떤 제품들이 나와 있는지 파악하고 어떤 기능들을 제공하는지 알아보는 수준으로 만족하도록 하자.

지금 소개할 네 가지 미들웨어 제품은 각각 리더와의 상호 작용을 외부에서는 신경쓰지 않아도 되도록 캡슐화하고, 이벤트를 관리하고, 애플리케이션을 위한 고수준 서비스 지향 인터페이스를 제공하는 핵심 기능을 제공한다. 이런 핵심 기능 외에도 다양한 관리 및 감시 기능, 서비스 지향 아키텍처 통합 기능, 다양한 ERP 패키지용 어댑터 제공 등의 추가 기능을 제공한다.

썬 마이크로시스템즈

썬 마이크로시스템즈는 RFID 시장에 초기에 뛰어든 업체 가운데 하나이다. 썬에서는 썬 자바 시스템 RFID 소프트웨어라는 자바 기반의 RFID 미들웨어 플랫폼을 제공한다. 썬의 RFID 소프트웨어는 EPC 네트워크를 위한 뛰어난 신뢰성과 확장성을 제공함은 물론, 다양한 기존의 백엔드 엔터프라이즈 시스템과의 통합 작업을 단순화시키는 데 초점을 맞추고 있다. 썬의 RFID 미들웨어는 자바 엔터프라이즈 시스템(JES)의 일부분으로, 썬 자바 엔터프라이즈 통합 서버(SeeBeyond의 후손이라고 볼 수 있음)를 비롯한 최신 엔터프라이즈 통합 서버군과 표준에 부합하여 통합할 수 있는 기능을 지원한다.

그 프로젝트는 RFID 이벤트 관리자, RFID 관리 콘솔, RFID 정보 서버, 그리고 어댑터 및 독립형 애플리케이션을 만들기 위한 소프트웨어 개발 키트(SDK), 이렇게 네 개의 구성 요소로 이루어진다.

■ RFID 이벤트 관리자

RFID 이벤트 관리자는 지니 기반의 이벤트 관리 시스템으로, 네트워크에 연결된 RFID 리더에서 생성되는 EPC 이벤트를 잡아내고 필터링하고 저장하는 기능들을 제공한다. RFID 리더와의 인터페이스, EPC 이벤트 수집, 중복된 정보 필터링, 그리고 다음 처리 단계를 수행하기 위해 RFID 정보 서버나 다른 ERP 소프트웨어로 이벤트를 넘겨주는 것이 이벤트 관리자의 주 임무이다. RFID 이벤트 관리자 지니 서비스는 지니 서비스 빈용 오픈 소스 컨테이너인 리오(Rio)를 통해 관리된다.

■ RFID 관리 콘솔

RFID 관리 콘솔(MC; Management Console)은 RFID 이벤트 관리자를 관리하고 감시하기 위한 브라우저 기반의 그래픽 인터페이스이다. RFID MC를 이용하면 RFID 리더 속성 및 필터나 커넥터와 같은 이벤트 관리자 구성 요소를 확인하고 수정할 수 있다. RFID MC는

썬 컨트롤 스테이션 2.2와 같은 룩앤필을 제공하는 스트럿츠 애플리케이션으로, RFID 리더 및 RFID 이벤트 관리자 구성 요소에 대해 테이블 기반의 디스플레이를 제공한다. 리더와 구성 요소의 속성을 수정해서 작동중인 시스템의 특성을 바로 변경하는 것도 가능하다. 그리고 사용자가 직접 리더를 분류할 수 있게 해 주고, 그룹 상태를 쉽게 파악할 수 있도록 보여주기도 한다. 리더의 알람 표 및 속성 관련 기능도 구현하고 있다. 각 리더는 그룹별로 표시되며, 그룹별 선택도 가능하다. 리더를 선택하면 리더 속성이 표시된다. 관리 도구에서는 리더 및 RFID 이벤트 관리자 시스템의 상태를 표시하기 위한 알람 디스플레이 프레임워크를 구현한다. 사용자는 알람 조건이 만족되면 사용자에게 이메일이 가도록 할 수도 있다. RFID MC에서는 JDBC 호환 관계형 데이터베이스를 사용하여 리더 분류 정보나 알람, 그리고 접근 권한이나 이메일 설정과 같은 시스템 설정을 지속적으로 유지시키는 기능도 제공한다. 오라클 8i, 오라클 9i, 오라클 10g, PostgreSQL 8.0.3과 함께 사용할 수 있다.

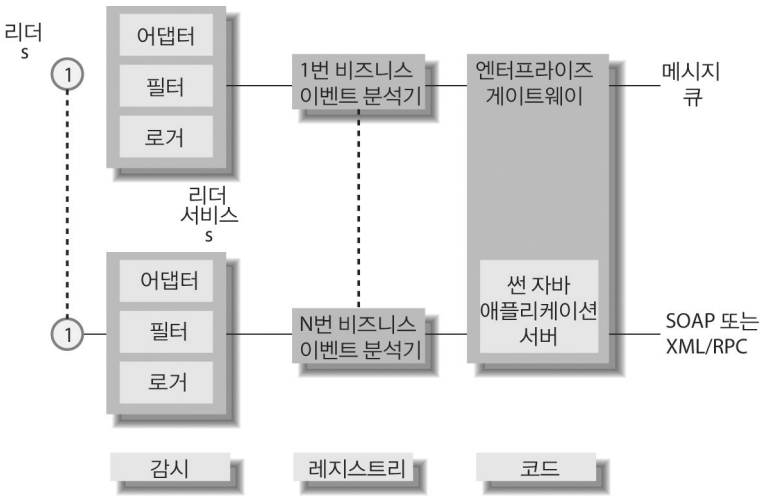
■ **RFID 정보 서버**

RFID 정보 서버(IS; Information Server)는 EPC 관련 데이터 획득 및 질의를 위한 인터페이스 역할을 하는 J2EE 애플리케이션이다. EPC 관련 데이터에는 이벤트 관리자로부터 받은 태그 관측 결과 데이터는 물론이고 EPC를 고수준 비즈니스 데이터에 대응시킨 데이터도 포함될 수 있다. RFID IS는 보통 일련의 저수준 관측 결과를 고수준 비즈니스 기능으로 바꾸기 위한 용도로 주로 쓰인다. 썬 자바 시스템 애플리케이션 서버 버전 8.1과 BEA 웹로직 9.0 인증을 받았다. 다른 애플리케이션에서도 XML 메시지 교환을 통해 IS와 연결할 수 있다. IS에서는 HTTP 및 JMS 메시지 전송을 통한 XML 메시지 인터페이스를 제공하며 JDBC 호환 관계형 데이터페이스를 가지고 모든 데이터를 지속적으로 보관할 수 있다. IS도 오라클 8i, 오라클 9i, 오라클 10g, PostgreSQL 8.0.3과 함께 사용할 수 있다.

■ **SDK**

구성 요소를 있는 그대로 사용하지 않고 직접 애플리케이션을 만들고 싶다면 SDK를 사용하여 직접 제품을 확장하는 것도 가능하다. 문서화도 잘 되어 있으며, 버전 3.0에서는 최신 리더 및 프린터 지원 기능도 추가되었다. 지니 2.0.1, 리오 3.1, 자바 웹 서비스 개발자 팩 1.5, 썬 자바 시스템 애플리케이션 서버 8.1을 바탕으로 만들어졌지만 이식성을 최대한 높일 수 있도록 설계되어 있어서 솔라리스, 리눅스, 윈도우즈 XP, J2ME CDC(임베디드 서비스)용 ALE 구현과 같은 다양한 플랫폼에서 사용할 수 있다.

[그림 7-17]에 썬의 자바 시스템 RFID 소프트웨어에서 핵심적인 논리 요소들이 나와 있다.



[그림 7-17] 썬 자바 시스템 RFID 미들웨어

ConnecTerra/BEA

ConnecTerra의 주력 제품인 RFTagAware는 장치 애플리케이션과 RFID 솔루션 개발용 소프트웨어 인프라 플랫폼이다. ConnecTerra는 애플리케이션 수준의 이벤트(ALE) 표준을 바탕으로 하는 미들웨어 솔루션을 가장 먼저 구현한 회사 중 하나이다. ConnecTerra는 EPCglobal 표준 그룹에도 매우 활동적으로 참여하고 있으며 설립자이자 CTO인 켄 트라웁 박사가 애플리케이션 수준 이벤트 규격 작성에 있어서 주도적인 역할을 하기도 했다.

RFTagAware를 이용하면 데이터베이스를 SQL로 추상화하는 것과 비슷한 방식으로 RFID 리더와 같은 장치를 추상화할 수 있다. 사용자는 관심 있는 이벤트를 데이터베이스 질의와 비슷한 식으로 기술하고 RFID 활동을 기반으로 만들어지는 결과에 등록하기만 하면 된다. 장치 내부에, 또는 장치 근방에 배치할 수 있는 소프트웨어인 RFTagAware 말단 서버(Edge Server)는 처리되지 않은 태그 정보를 가공한 다음 임의의 개수의 질의(이벤트 사이클 규격(Event Cycle Specification)이라고 부름)를 바탕으로 등록한 애플리케이션에 결과를 전송한다. 이 때 질의 및 등록 애플리케이션의 개수에는 제한이 없다. 다른 애플리케이션에서 사용 중인 질의에 영향을 미치지 않고도 실시간으로, 그리고 독립적으로 질의를 추가하거나 변경하거나 제거하는 것이 가능하다. 말단 서버에서는 장치 사용을 질의 내용을 수집하는 데 최적화시킴으로써 하드웨어 사용을 최적화시키는 일까지 맡아서 해 준다.

RFTagAware에서는 다음과 같은 기능을 제공한다.

■ 데이터 필터링 및 집약

RFID 리더에서는 짧은 시간 간격으로 처리되지 않은 저수준 데이터를 계속해서 뿜어낸다.

RFTagAware를 사용하여 원하는 정보를 정의하고, 들어오는 RFID 데이터를 제어하고 필터링하고 모을 수 있다.

■ **RFID 인프라 감시 및 관리**

RFTagAware는 리더를 중앙에서 감시하고 관리할 수 있도록 해 주는 관리 콘솔(Administration Console)을 제공한다. RFID 장치는 일반적인 IT 제품과는 그 관리법이 크게 다르다. 리더와 PC는 아키텍처 면에서는 유사성이 어느 정도 있지만 설정 방법, 작동 상태를 확인하는 방법, 고장 원인 등은 현저하게 다르다. ConnecTerra의 RFTagAware에서는 그런 장치의 속성들을 감시하고, 업무에 방해가 되기 전에 문제점을 찾아내거나 문제 발생 가능성을 인지할 수 있는 도구를 제공한다.

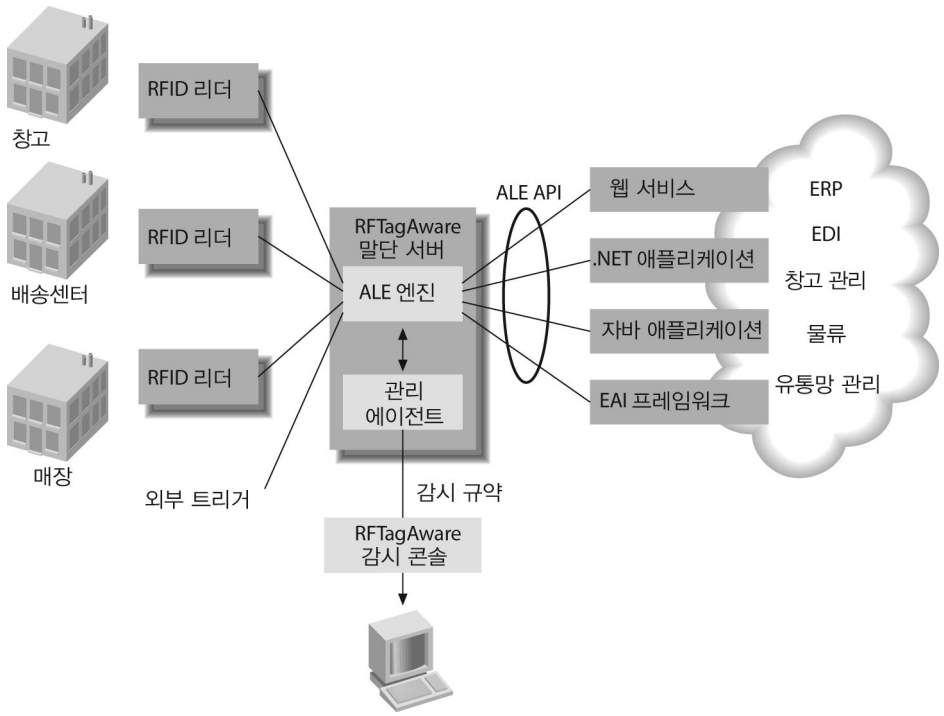
■ **엔터프라이즈 애플리케이션과의 데이터 통합**

RFTagAware에서는 ALE 호환 애플리케이션 프로그래밍 인터페이스를 제공하여 창고 관리 시스템이나 유통망 관리 시스템, ERP 애플리케이션 같은 기존 엔터프라이즈 도구 및 애플리케이션과 통합할 수 있게 해 준다. RFTagAware에서는 개발자들이 .NET이나 자바 도구를 사용하여 지역 작업 흐름을 생성하고 배치할 수 있도록 ALE 인터페이스에 대한 어댑터도 제공한다. 말단 서버에서는 계속해서 성장하고 있는 통지 드라이버 슈트(notification driver suite)를 활용하여 표준 및 준표준에서 특정 시스템 전용의 메시징 메커니즘에 이르기까지 다양한 방식에 대한 애플리케이션도 뒷받침한다. HTTP, SOAP/XML, JMS와 같은 표준 메커니즘은 물론이고 TIBCO, SeeBeyond, SAP 같은 업체별 엔터프라이즈 애플리케이션 통합 플랫폼도 이미 지원되고 있다.

■ **RAD**

앞에 나온 애플리케이션 통합 기능 외에도 ConnecTerra에서는 팔레트에 EPC 태그를 부착, 복합 포털, 다양한 모바일 리더 시나리오 등 다양한 분야를 커버할 수 있는 작업 흐름을 미리 구현해 놓은 것들을 제공한다. SAP AII, 엔터프라이즈 메시지 큐 제품 및 유명 엔터프라이즈 애플리케이션용으로도 제공하고 있으며, 기존 시스템에 RFID 데이터를 통합하는 과정을 훨씬 간단하게 처리할 수 있다. [그림 7-18]에 RFTagAware 미들웨어 플랫폼을 간략하게 그림으로 표현해 보았다.

말단 서버의 주 구성 요소로는 필터링 및 수집 엔진(ALE 엔진이라고도 부름)과 장치 관리 에이전트를 들 수 있다. RFTagAware 말단 서버에서는 다양한 리더 및 프린터는 물론, 리더 제어용 트리거로 쓰이는 다양한 센서 입력 인터페이스도 제공한다. 말단 서버에서는 EPCglobal ALE API를 구현하며, 아직 표준에는 포함되지 않은 태그 기록 및 기타 기능도 포함되어 있다. 그리고 말단 서버 작동을 원격으로 직접 확인할 수 있는 관리 콘솔은 물론이고, 말단 서버 및 장치를 관리하고 감시하기 위한 API도 제공한다.



[그림 7-18] ConnecTerra의 RFTagAware 미들웨어 플랫폼

ConnecTerra의 기술 아키텍처는 EPCglobal에서 표준을 만들기 위해 사용하는 아키텍처 프레임워크를 그대로 반영하고 있다. API가 정의된 다양한 계층을 사용하는데, 각 API는 구현에 따라 서로 다른 여러 규약으로 다시 대응된다. ConnecTerra에서는 표준 외에도 실시간 리더 감시, 태그 기록, 태그 감독은 물론 지역 작업 흐름 구성 요소 같은 영역에 대한 아키텍처 인터페이스도 추가했다.

GlobeRanger

GlobeRanger는 RFID, 센서 및 다른 말단 장치용 에지웨어(edgeware) 플랫폼을 제공하는 데 초점을 맞추고 있는 RFID 미들웨어 업체이며, 꽤 이른 시기부터 이 분야에 뛰어들었다. GlobeRanger에서는 파트너 OEM 및 VAR 등을 통해 iMotion 소프트웨어를 제공하며, 다양한 분야의 고객사에 대한 솔루션 구축은 파트너 OEM 및 VAR 업체들이 맡는다.

iMotion 소프트웨어 플랫폼에는 솔루션 개발, 배치 및 관리를 용이하게 해 주는 비주얼 툴들이 포함되어 있다. iMotion 플랫폼은 마이크로소프트의 .NET 프레임워크로 만들어졌으며, ALE나 EPCIS 규격과 같은 새로 만들어진 표준을 잘 활용하고 있다. iMotion 플랫폼은 네 가지 핵심 구성 요소로 이루어진다.

■ **말단 장치 관리**

말단 장치 관리(Edge Device Management) 구성 요소는 RFID, 모바일 및 센서 장치를 관리하기 위한 기능들을 쉽고 빠르게 사용할 수 있는 형태로 제공한다. ALE를 통해 제공되는 데이터 관리는 ALE 호환 애플리케이션이라면 어떤 애플리케이션에서든 손쉽게 사용할 수 있다.

■ **말단 절차 관리**

말단 절차 관리(Edge Process Management) 구성 요소는 시각적인 이벤트 작업 흐름 기능을 제공하여 절차를 설계할 때 RFID 데이터 및 업무 절차 흐름을 비주얼하게 가공할 수 있다.

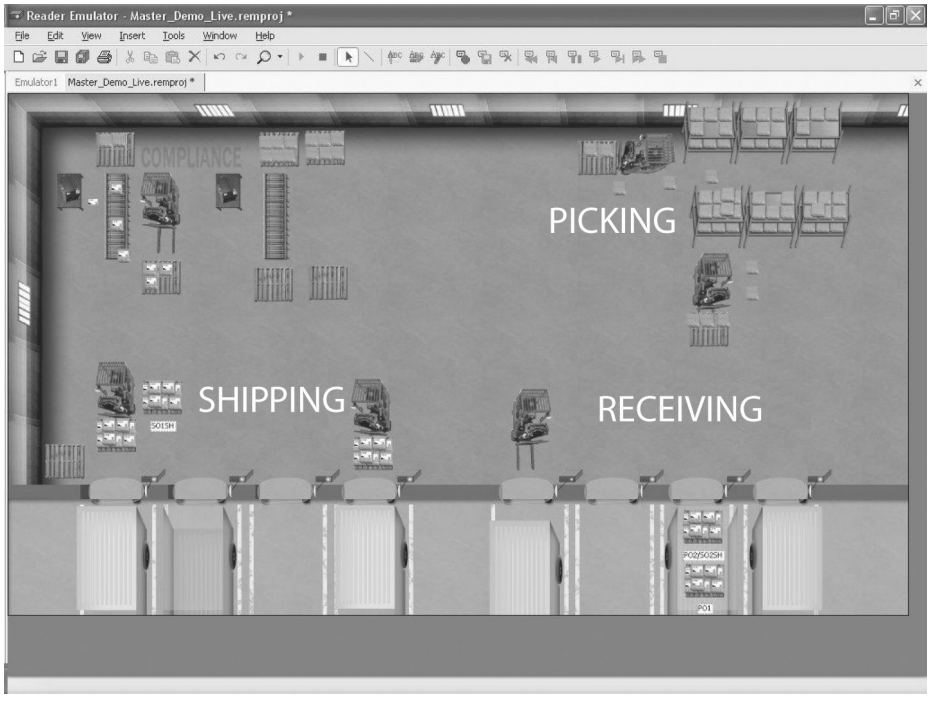
■ **엔터프라이즈 관리 콘솔**

엔터프라이즈 관리 콘솔(Enterprise Management Console)은 말단 장치와 네트워크의 작동 상태와 성능을 감시할 수 있는 중앙 집중적인 메커니즘을 제공한다.

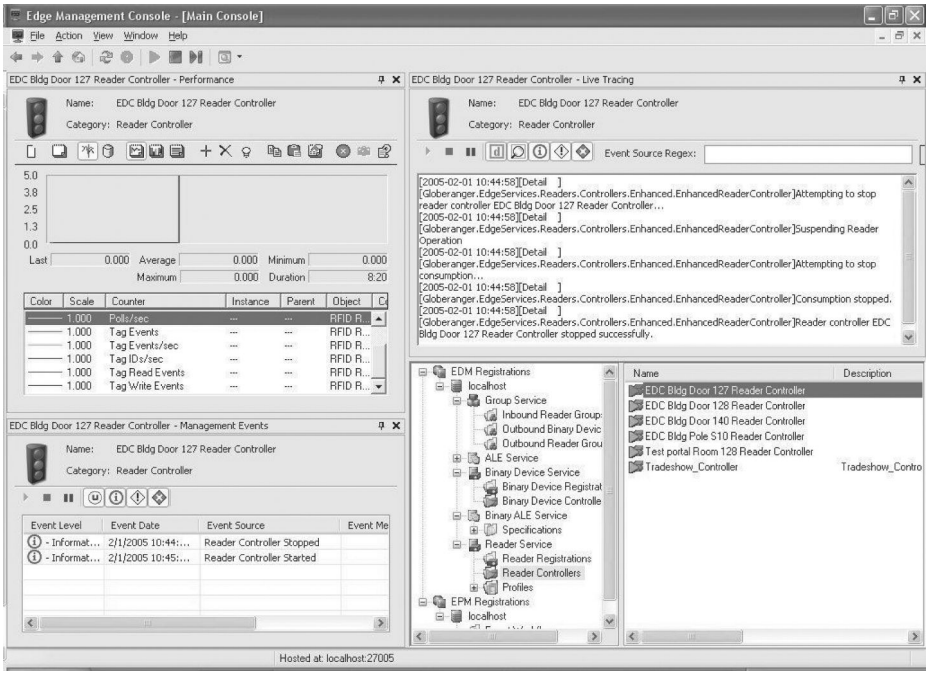
■ **비주얼 장치 에뮬레이터**

비주얼 장치 에뮬레이터(Visual Device Emulator)는 테스트 및 통합을 위한 배치 에뮬레이션 환경을제공해 주는 것으로, 이 업계에서는 최초로 만들어졌다.

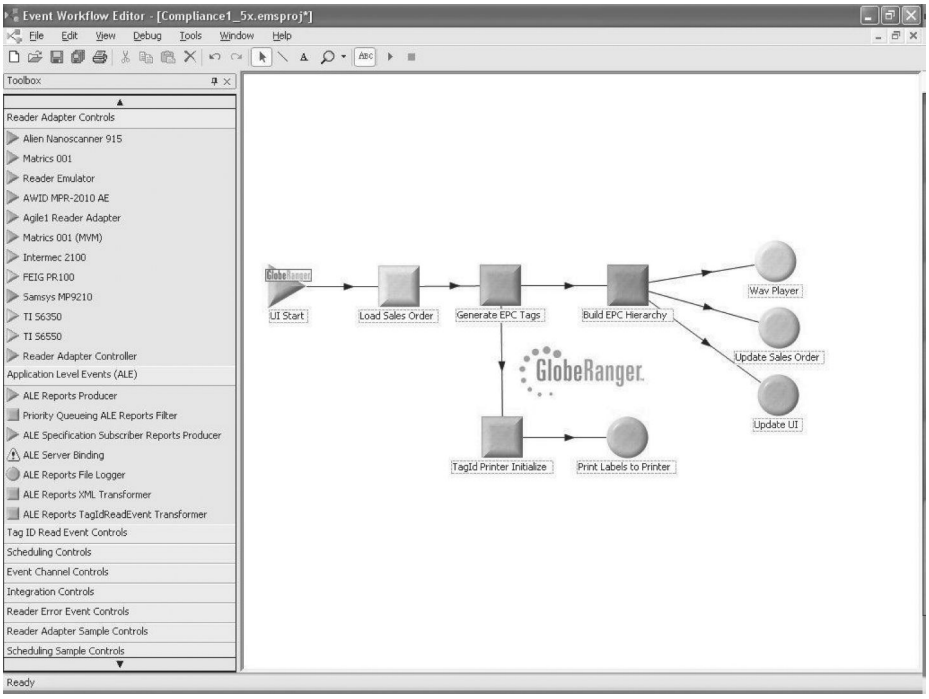
[그림 7-19]에서 [그림 7-21]에 걸쳐 GlobeRanger에서 나오는 비주얼 장치 에뮬레이터, 말단 관리 콘솔, 이벤트 작업 흐름 편집기의 스냅샷을 수록해 놓았다.



[그림 7-19] GlobeRanger의 비주얼 장치 에뮬레이터



[그림 7-20] GlobeRanger의 말단 관리 콘솔



[그림 7-21] GlobeRanger의 이벤트 작업 흐름 편집기

요약

이 장에서 배운 내용을 정리해 보면 다음과 같다.

- RFID 미들웨어는 RFID 솔루션에서 매우 중요한 요소 중의 하나이다.
- RFID 미들웨어를 사용하는 이유는 크게 세 가지로 나눌 수 있다. 첫 번째는 애플리케이션의 입장에서 장치 인터페이스 및 연결에 대한 정보는 전혀 모르면서도 리더에 연결할 수 있도록 하기 위함이고, 두 번째는 리더로부터 들어오는 처리되지 않은 RFID 관측 결과를 필터링하고 분류함으로써 애플리케이션에서 처리해야할 정보량을 줄이기 위함이다. 또한 세 번째는 리더를 관리하고 RFID 관측 결과를 질의하기 위한 애플리케이션 수준의 인터페이스를 제공하기 위함이다.
- RFID 미들웨어용으로는 여러 가지가 구현되어 있다. EPCglobal의 애플리케이션 수준 이벤트(ALE) 규격에서는 RFID 리더 및 기타 센서와 같은 다양한 소스로부터 수집되고 필터링 및 정리가 완료된 EPC 데이터를 얻을 수 있는 표준화된 인터페이스를 클라이언트에게 제공한다.
- Savant 규격은 더 이상 EPCglobal 아키텍처에 들어가지 않는다. Savant의 모든 기능은 ALE 표준으로 합병되었다.
- 기본적으로 ALE 규격을 이용하면 애플리케이션에서는 물리적인 RFID 인프라에는 신경 쓸 필요 없이 어떤 정보를 원하는지, 어떤 식으로 받고 싶은지만 기술하면 된다.
- ALE가 나오기 전에는 애플리케이션 개발자가 장치에 연결하고 데이터를 필터링하고 그 데이터를 사용할 애플리케이션에 데이터를 전달하는 기능을 직접 구현해야 했다. 하드웨어, 애플리케이션, 필터 규격 같은 것이 바뀌면 코드를 고치고 디버깅을 하고 테스트까지 한 다음 소프트웨어를 새로 배치해야 했다. 소규모 배치 환경이라고 하더라도 RFID 시장처럼 빠르게 변화하는 환경에서는 이런 작업 방식을 유지하려면 상당한 위험과 비용을 감당해야만 한다.
- 애플리케이션에서 처리해야 할 데이터 분량을 줄이고 데이터의 적합성을 높이려면 애플리케이션으로 보내기 전에 RFID 관측 결과를 미리 처리해야 한다. ALE 규격에서는 RFID 관측 결과를 필터링하고 분류할 수 있는 간단하면서도 유연한 메커니즘을 제공한다. 이런 필터링 및 분류 기능 때문에 애플리케이션에서 관심을 가지고 있는 이벤트만 뽑아내고 그런 이벤트에만 관심을 기울이는 것이 가능하다.
- ALE 규격 덕분에 애플리케이션을 인프라의 물리적인 계층하고 분리시킬 수 있기 때문에, 개발자 입장에서는 애플리케이션 수준의 작업에만 집중하면 된다.

- ALE 규격에서 지원하는 주된 상호 작용 모형은 ALE 서비스에 대한 모든 메소드 호출이 동기식으로 작동하는 요청/응답 모형이다. 동기식 모형에는 즉시 모드 및 폴링 모드, 이렇게 두 가지 상호 작용 모드가 있다. ALE 인터페이스에서는 클라이언트에서 이벤트에 등록하는 비동기 모형도 지원한다. 이벤트가 발생하면 ALE 서비스에서는 비동기식으로 클라이언트에 데이터를 전달한다.
- ALE 규격에서는 필터링(filtering)과 분류(grouping), 이렇게 두 가지 이벤트 처리 메커니즘을 제공한다. 필터링은 이벤트 데이터의 특정 패턴을 골라내는 기능을 제공한다. 분류는 서로 다른 리더로부터 여러 이벤트 사이클에 걸쳐서 얻은 데이터를 분류하는 기능을 제공한다.
- ALE 호환 미들웨어 제품이 이미 시장에 나와 있다. 기본 이벤트 관리 기능을 제공하고 ALE 규격을 지원하는 것 외에도 리더 및 센서 감시 및 관리, 그리고 작업 흐름 기능과 같은 분야에서 다양하면서도 심도 있는 추가 기능들을 제공한다.