RFID/USN 환경에서의 이기종 서비스 시스템 간 협업지원을 위한 비즈니스 로직 프레임워크 구현

Implementation of Business Logic Framework for Collaboration of Heterogeneous Service Systems in RFID/USN Environment

장준호(Juno Chang)*

초 록

유비쿼터스 컴퓨팅 환경을 지원하기 위한 다양한 미들웨어 및 비즈니스 로직 프레임워크에 대한 연구 및 개발이 이루어져왔으며 이를 통해 수많은 응용 서비스들이 제공되고 있다. 그러나, 사용자 측면에서의 진정한 유비쿼터스 환경은 개별 시스템들이 제공하는 서비스들이 통합적으로 지원되는 환경이다. 따라서, 다양한 이기종 서비스 시스템들이 적절한 수준의 협업을 통해 사용자의 요구사항을 만족시키는 미들웨어 플랫폼 및 비즈니스 로직 프레임워크의 개발이 요구된다. 또한, 사용자들이 다양한 이벤트에 따라 작동하는 서비스 시나리오를 원활히 처리할 수 있는 기능을 도입함으로써 협업 효과를 극대화할 필요가 있다. 이에 본 논문에서는 사용자가 원하는 시나리오에 기반하여 이기종 서비스 시스템 간의 협업을 지원하는 비즈니스 로직 프레임워크를 구현하고 이에 대한 성능분석 결과에 대해 설명한다.

ABSTRACT

Nowadays the research and development of various middleware for the ubiquitous computing environment has been demanded and many application services have been provided. The practical ubiquitous computing environment is the world where a variety of services based on RFID/USN devices are integrated with each other and make synergy effects all together. In the end user perspectives, there should be a middleware platform and business logic framework collaborating various services all together to satisfy the user requirements. In addition, the platform should provide the user to describe a scenario based on various event to maximize the collaboration. In this paper, we have proposed business logic framework which is supporting collaboration between heterogeneous service systems based on the user's scenario. We have also tried to show the efficiency and scalability of the proposed framework by providing the result of a couple of tests.

키워드 : 유비쿼터스 컴퓨팅 환경, 미들웨어 플랫폼, 비즈니스 로직 프레임워크, 시나리오 기반
Ubiquitous Computing Environment, Middleware Platform, Business Logic Framework, Scenario-based

본 논문은 상명대학교 2008년도 교내연구비 지원에 의해 수행되었슴.

* 상명대학교 디지털미디어학부

2009년 05월 24일 접수, 2009년 07월 20일 심사완료 후 2009년 07월 24일 게재확정.

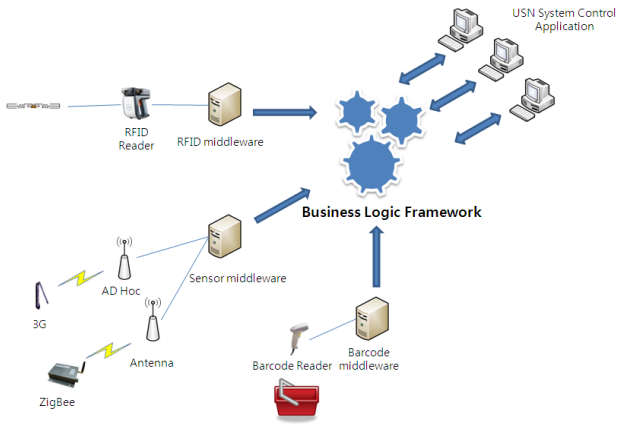
1. 서 론

컴퓨터 및 네트워크 기술의 발전이 유비쿼터스 컴퓨팅 환경 구축을 가속화함에 따라 이를 활용하는 다양한 응용 서비스가 도입되었으며, 나아가 독립적으로 존재하는 응용 서비스들의 유기적 연결이나 통합에 대한 관심이 증폭되고 있다. 특히, 편리하고 쾌적한 생활 및 산업 환경을 실현할 방안으로 다양한 유비쿼터스 센서 네트워크(Ubiquitous Sensor Network, 이하 USN) 기반 응용 서비스가 제안되었으며, 이들 서비스들을 연계함으로써 개별 서비스 만으로 불가능했던 복합적인 통합 연계 서비스의 제공이 가능해졌다.

RFID/USN 환경에서 응용 서비스 및 이들 간 유기적 연계를 통한 복합 서비스를 구현하기 위해서는, 방대한 양의 센서 데이터를 안정적으로 수집하여 응용 서비스 별로 기 정의된 시나리오에 맞게 실시간으로 전달할 수 있는 통합된 USN 응용 서비스 인프라 환경이 필요하다. 따라서 본 연구에서는, RFID/USN 환경에서의 이기종 서비스 시스템 간 시나리오 기반의 협업이 가능하도록 하는 비즈니스 로직 프레임워크를 제안한다.

본 프레임워크는 <그림 1>에서와 같이 개별 RFID 또는 센서 미들웨어들을 통칭하는 USN 미들웨어와 응용 서비스 사이에 존재하여 USN 미들웨어가 전송하는 리포트를 수집하고, 이에 상응하는 사용자 정의 비즈니스 프로세스와 응용 서비스들이 협업하여 손쉽게 통합 서비스를 구축하는 인프라 환경을 제공한다. RFID/USN 기반 응용 서비스들을 연계 통합하는 방법으로 개별 서비스의 독립성과 응용 서비스 연계시의 유연성을 보장할 수 있는 구조인 SOA(Service Oriented Architecture)를 채택하였다. SOA는 서비스 지향 아키텍처로 기존의 애플리케이션들의 기능들을 비즈니스적인 의미를 가지는 기능 단위로 묶어 표준화된 호출 인터페이스를 통해서 서비스라는 소프트웨어 컴포넌트 단위로 재조합한 후, 이 서비스들을 서로 조합하여 업무 기능을 구현한 애플리케이션을 만들어 내는 구조이다.

SOA는 크게 3가지 장점을 가진 구조이다. 첫 번째는 재활용의 증대이다. SOA는 Write



〈그림 1〉 비즈니스 로직 프레임워크 개관

Once Read Many라는 재활용의 속성을 이용한다. 이는 CBD(Component Based Development)와 동일한 개념을 갖는 것뿐만 아니라 SOA의 통합 기능을 이용함으로써 서비스 간의 연결을 합리적인 방법으로 해결하여 재활용 빈도를 증가시키고 안정적인 서비스 제공의 기반을 마련할 수 있다. 두 번째는 헐거운 결합이다. 결합력을 감소시키는 것은 재활용 증대와 함께 S/W 개발방법에서 지향하는 가장 중요한 목표중의 하나이다. 마지막으로 기술중립성이다. 기존의 컴포넌트들이 어떠한 플랫폼 기반에서도 SOA 기반환경에서 연계하여 함께 사용할 수 있다. 날로 고도화되는 유비쿼터스 컴퓨팅 기술과 외부의 디바이스들을 기반으로 하는 다양한 서비스 요구 증가는 SOA와 같은 유연한 연계 구조를 필요로 하며 앞으로 SOA의 활용 영역은 계속해서 넓혀질 것이라고 예상된다[1].

본 비즈니스 로직 프레임워크는 특정 USN 미들웨어와의 종속성을 피하고 외부에서 발생하는 이벤트와 일치하는 시나리오들을 실행시키기 위해 USN 미들웨어와 결합도가 없는(decoupled) EDA(Event Driven Architecture)를 가진다. SOA 구조를 위해 이기종 시스템 간의 연계 방식을 웹 서비스로 정의하였다. 미들웨어 플랫폼의 이식성을 고려하여 Java 기반으로 제작하였고 이기종 시스템 간의 쉬운 연계를 위해 스크립트 언어를 개발하여 시나리오 구현을 편리하게 해주었다. 이를 본 논문에서는 'XLogic(eXtensible Logic)'이라 명명하였다. 실시간 처리를 지원하기 위해 한번 실행된 XLogic 스크립트를 사용률에 따라 메모리상에 상주시키는 캐싱 방식을 사용하여 XLogic 스크립트의 파싱 시간을 최소화하였다.

본 논문의 구성은 다음과 같다. 제 2장에서 관련연구에 대해 기술하였으며, 제 3장에서 본 연구결과물인 비즈니스 로직 프레임워크의 구성 및 기능을 설명하였다. 제 4장에서는 기능을 시험할 테스트 시나리오의 내용에 대해 설명하고 있으며, 제 5장에서 성능 테스트 결과를 제시하였고, 마지막으로 제 6장에서 결론 및 향후 연구방향 등을 기술하였다.

2. 관련 연구

2.1 USN 미들웨어

초기 USN 미들웨어는 그 기능이 각 응용분야의 서비스에만 적용되는 수준으로 미들웨어에 대한 요구 사항이 크지 않았다. 그러나 u-Healthcare, u-Silvercare, u-Transportation, u-Hospital, 재난·재해 방지 및 범죄 예방, 환경 감시 시스템 등[2]과 같은 여러 종류의 유비쿼터스 환경을 바탕으로 한 응용 서비스가 발달함에 따라, 단순히 정보를 전달하는 범위를 벗어나 다양한 기능이 제공되는 USN 미들웨어가 소개 되었다. MiLAN 미들웨어[3]는 USN 응용 서비스의 QoS(Quality of Service) 요구조건에 대한 보장을 최우선 목표로 하여 개발되었다. DSWare 미들웨어[4]는 RFID 미들웨어와 유사하게 끊임없이 획득되는 센싱 정보에 대하여 이벤트들을 설정함으로써 USN 응용 시스템들에게 원하는 정보를 전송하는 이벤트 기반의 미들웨어이다. 또한, USN 응용 서비스 변화 및 센서 네트워크 주변 환경 변화에 따라 센서노드 미들웨어의 기능을 무선

통신을 통하여 동적으로 변화시킬 수 있는 Impala 미들웨어[5], 센서 네트워크에 존재하는 센싱 정보들을 분산 데이터베이스의 분산 데이터로 간주하여 USN 응용 시스템의 요구사항을 분산 질의 처리 과정으로 수행할 수 있는 TinyDB 미들웨어[6] 및 cougar 미들웨어[7] 등이 그 예이다.

2.2 COSMOS

ETRI에서 개발한 COSMOS(COmmon System for Middleware Of Sensor network)는 다양한 유형의 USN 응용 서비스에 공통적으로 필요한 미들웨어의 핵심기능을 추출하고, 이들을 표준화된 방식으로 제공하기 위한 기술 개발 및 표준화를 진행하였다. COSMOS의 주요 기능으로는 다양한 유형의 질의 지원(일시성, 연속성, 이벤트), 대용량 센서 네트워크 환경에 대하여 대량의 동시질의 처리 지원, 이종의 센서 네트워크에 대한 추상화 기능 지원이 있다[8, 9].

COSMOS는 크게 세 부분의 계층으로 나뉘어서 위의 기능을 수행한다. 첫 번째는 다수의 이기종 센서 네트워크들을 용이하게 통합할 수 있게 하고 센서 네트워크 인프라와 직접 연계 되는 부분으로 센서 네트워크의 상태를 실시간 모니터링 할 수 있도록 한 센서 네트워크 추상화 계층이 있다. 두 번째는 연속적으로 입력되는 센싱 정보에 대하여 의미를 부여하는 단계로서 센싱 정보를 효율적으로 관리하고 이들 센싱 정보와 기 구축된 비즈니스 정보를 분석하여 새로운 상황정보를 생성하는 역할을 수행하는 센서 네트워크 지능화 계층이다. 마지막으로는 USN 응용 서비스 시스템의 개발을 효율적으로 지원하기 위한 단계로서, USN 미들웨어를 용이하게 사용하기 위한 개방형 API를 제공하고, 다중의 서비스 사용자들에 대한 관리를 수행하고, 외부 서비스 시스템과의 연계를 지원하고, USN 응용 서비스 시스템이 요구하는 센서 노드 및 센서 네트워크 관련의 정적/동적 메타정보를 실시간으로 제공하는 USN 서비스 플랫폼 계층이다[8, 9].

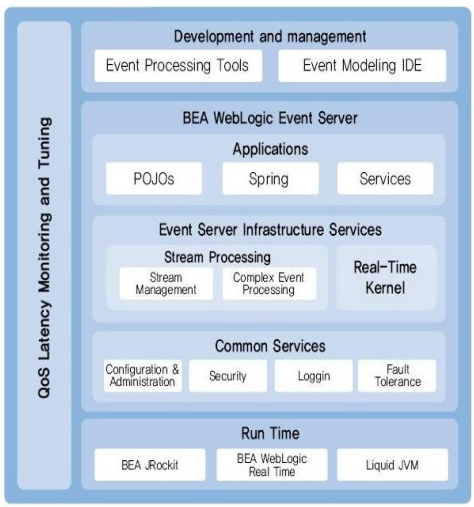
〈표 1〉 SOA와 EDA 특성

	EDA	SOA
트리거	이벤트 기반 트리거	서비스 소비자가 트리거
상호 규약 정보	이벤트 규격 정보	서비스 인터페이스 정보
흐름 제어 주체	이벤트 수신자	클라이언트
커뮤니케이션 Interaction 연결방식	동적, 병렬, 비동기 방식 단방향 Decoupled 1 : 1, 1 : N, N : N	순차 경로, 동기식 양방향(Request/Response) Loosely-Coupled 1 : 1
지향	실시간 서비스 제공	서비스 재사용과 공유

2.3 Event Driven SOA

2006년 6월 가트너 그룹에서는 EDA(Event Driven Architecture)와 SOA를 접목시킨 SOA 2.0의 개념을 새롭게 제시함으로써 향후 SOA의 발전 방향을 제시하였다[10]. SOA 2.0이란 SOA와 EDA가 결합된 형태의 개념으로서 Event Driven SOA라 불리기도 한다. SOA가 정의된 서비스 인터페이스를 이용하여 요청 및 응답을 통해 서비스를 연동하는 모델이라면 EDA는 이벤트에 대해 실시간으로 감지하고 대응하는 모델이다<표 1>.

SOA 2.0은 두 구조의 장점을 접목시켜 주변 환경의 변화를 감지하고 이에 능동적으로 개별 시스템의 연계 관계를 변화시킬 수 있게 되어, 사용자 맞춤형 서비스의 제공, 효율적인 서비스 요청 처리 및 시스템 연계 등의 작업이 가능하게 한다. SOA 2.0 개념에 맞는 개발을 위해 ORACLE[11], BEA[12], SAP[13], TIBCO[14] 등이 연구 중에 있다.

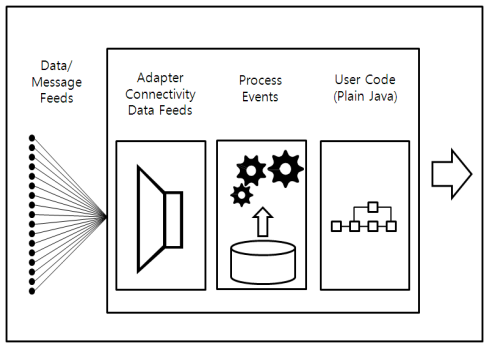


〈그림 2〉 BEA Event Driven SOA 솔루션

2.4 BEA Event Driven SOA 솔루션

BEA에서 개발한 BEA Event Driven SOA 솔루션은 대규모로 유입되는 다양한 이벤트를 처리할 수 있는 새로운 소프트웨어 패러다임에 기초하고 있으며 이를 통해 기업은 미래 상황을 효과적으로 예측하여 기업 투자수익(ROI)을 극대화할 수 있는 솔루션이다[12]. BEA Event Driven SOA 솔루션의 구성도는 <그림 2>와 같다.

BEA Event Driven SOA 솔루션은 크게 세 가지의 제품으로 구성되어 있다. 첫번째로 WebLogic Event Server는 대용량의 실시간 Event-driven 애플리케이션을 위한 Java 컨테이너이다. 두 번째는 WebLogic RealTime 이다. 이것은 표준 Java 기반 인프라에서 빠르고 예측 가능한 응답 시간과 짧은 대기 시간을 요구하는 애플리케이션을 지원하는 애플리케이션 서버이다. 세 번째로 WebLogic RFID Edge Server는 RFID리더 및 기타 장치를 제어하여 Event 데이터 안정화 및 필터링을 지원하며 WebLogic Event Server로의 데이터 전달을 담당한다. 이벤트 전달 과정부터 처리 순서는 <그림 3>과 같다.



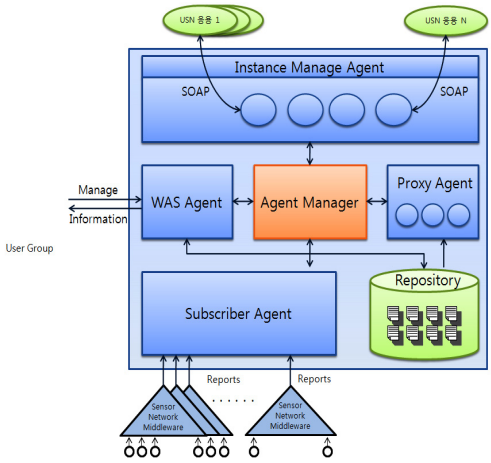
〈그림 3〉 BEA Event Driven SOA

3. 비즈니스 로직 프레임워크

본 논문에서 제시하는 비즈니스 로직 프레임워크는 RFID/USN 환경에서 시스템 통합을 위한 시나리오 기반 협업 지원 기능을 제공한다. 비즈니스 로직 프레임워크는 위 시스템 간 협업을 해결하기 위하여 네 종류의 Agent와 이를 관리하는 Agent Manager로 구성되며 사용자가 정의하는 시나리오를 지원할 수 있는 XML 형의 XLogic이라는 인터프리터 언어를 제공한다. 각 Agent는 Agent Manager를 중심으로 이벤트 정보를 교환하며 이들의 연관관계는 <그림 4>와 같다.

비즈니스 로직 프레임워크는 이식성을 고려하여 Java 기반으로 제작되었으며 이질적인 응용 서비스들과의 인터페이스를 SOAP로 정의 하였다. 내부에서 처리되는 모든 실행 과정들은 이벤트 단위로 수행되며, 주요 내부 이벤트는 <표 2>와 같다.

그리고 Agent들의 공통적인 기능들은 <표 3>과 같다.



〈그림 4〉 비즈니스 로직 프레임워크의 구조

〈표 2〉 주요 내부 이벤트

종 류	설 명
Reports Arrived Event	본 미들웨어 플랫폼에 XML 형태로 전송되는 리포트를 저장하고 있는 Java 객체
XLogic Script Execute Request Event	XLogic의 통계정보와 실행 계획을 가지고 있는 Java 객체

3.1 Subscriber Agent

RFID/USN 장비로부터 발생하는 이벤트는 USN 미들웨어가 센싱된 정보를 실시간으로 요청하는 snapshot 질의, 센싱 정보를 일정주기로 연속적으로 요청하기 위한 continuous 질의 등의 방법으로 수집된다.

Subscriber Agent는 USN 미들웨어와 미리 설정된 포트로 소켓통신을 하여 Multi Threading 방식으로 센싱된 정보를 전달 받은 후 내부에서 'Reports Arrived Event'로 변환하여 Agent Manager에 전달하는 역할을 한다.

〈표 3〉 Agent들의 공통 기능

메소드	기 능
getAgent Name( )	Agent의 이름을 반환함.
Start( )	Agent를 시작함.
Stop( )	Agent를 중지함.
getState( )	STARTED, STARTING, STOPPED, STOPPING의 상태로 현재 Agent의 상태를 반환함.
process Agent Event (Agent Event event)	자신에게 맞는 event를 처리함.

3.2 Agent Manager

Agent Manager는 Agent들을 관리하고 각 Agent에게서 전달 받은 모든 이벤트들을 적재적소에 전달하는 역할을 한다. 내부에서 발생하는 모든 이벤트는 항상 Agent Manager를 거쳐 전달되는데 Agent Manager는 자신에게 등록된 Agent들에게 이벤트를 Broadcasting 방식으로 전파하게 된다. 이때 등록된 Agent들은 자신이 처리해야 할 이벤트만을 받아들이며 그 외의 이벤트들은 무시한다.

3.3 Proxy Agent

Proxy Agent는 이벤트의 처리 속도 향상을 위한 역할을 한다. 한번 이상 실행된 XLogic 스크립트는 Repository에 저장된 XML 형태가 아닌 실행 가능한 Java 객체 형태로 메모리에 상주시킴으로써, 추후 동일한 XLogic 실행 시 파싱 작업을 줄여 효율적으로 자원을 관리한다.

Proxy Agent는 Agent Manager로부터 'Reports Arrived Event'를 받아 이벤트가 가진 고유한 값을 추출한다. 이 값이 메모리에 상주하고 있는 XLogic이 가진 값과 같다면 해당 XLogic을'XLogic Script Execute Request Event'로 변환시켜 Agent Manager로 전달한다. 만약 메모리에서 발견하지 못할 경우 Repository에 이벤트의 고유 값에 적합한 XML 형태의 XLogic을 질의하여 이를 파싱 작업을 통해 실행 가능한 Java 객체로 메모리에 상주 시킨 후, 위와 동일한 방법으로 이벤트를 Agent Manager에게 전달한다.

3.4 Instance Manage Agent

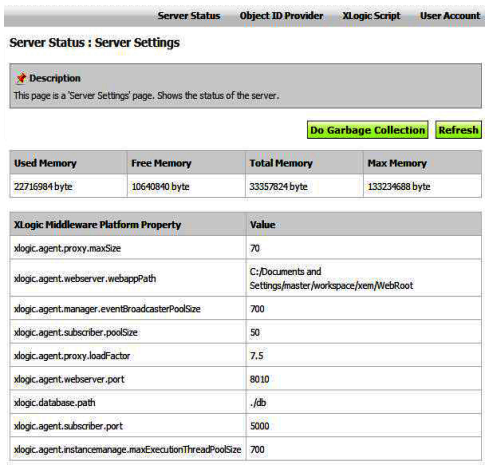
Instance Manage Agent는 XLogic을 실행시키고 실행되고 있는 XLogic들에 대한 성공 횟수, 실패횟수, 마지막으로 실행된 시간, 현재 상태를 관리해준다. 위의 항목들은 Enterprise Manager(이하 EM)란 웹 인터페이스를 통해 실시간으로 제공되며 사용자가 통계 정보를 보고 XLogic에 대한 실행 전략을 세울 수 있게 도와준다.

3.5 WAS Agent

사용자에게 직접적으로 제공할 기능을 웹 서비스로 외부에 노출시키는 역할과 사용자의 편리한 관리를 위한 EM(<그림 5>)의 웹 서버 역할을 한다.

3.6 Trigger Mode

외부에서 발생하는 이벤트가 본 미들웨어


Server Status | Object ID Provider | XLogic Script | User Account

Server Status : Server Settings

Description
This page is a 'Server Settings' page. Shows the status of the server.

Do Garbage Collection | Refresh

Used Memory	Free Memory	Total Memory	Max Memory
22716984 byte	10640840 byte	33357824 byte	133234688 byte

XLogic Middleware Platform Property	Value
xlogic.agent.proxy.maxSize	70
xlogic.agent.webserver.webappPath	C:/Documents and Settings/master/workspace/xem/WebRoot
xlogic.agent.manager.eventBroadcasterPoolSize	700
xlogic.agent.subscriber.poolSize	50
xlogic.agent.proxy.loadFactor	7.5
xlogic.agent.webserver.port	8010
xlogic.database.path	./db
xlogic.agent.subscriber.port	5000
xlogic.agent.instancemanage.maxExecutionThreadPoolSize	700

〈그림 5〉 Enterprise Manager

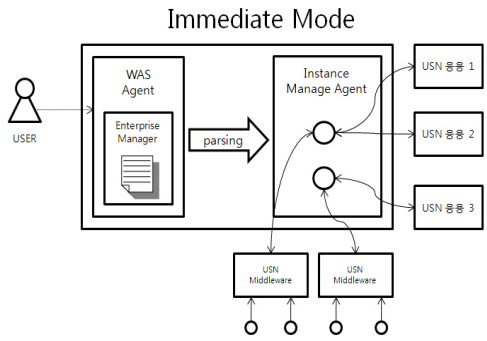
플랫폼에서 수집되었을 때에 그에 상응하는 시나리오가 실행되는 방식을 말한다<그림 6>.

최초 Subscriber Agent에서 이벤트를 수집하면 Agent Manager에게 'Reports Arrived Event'를 전송하게 된다. Agent Manager는 Subscriber Agent로부터 전달 받은 이벤트를 자신에게 등록된 Agent들에게 모두 전달한다. 그러면 'Reports Arrived Event'를 처리할 수 있는 Proxy Agent는 이벤트에서 추출한 고유의 값에 상응하는 XLogic이 메모리에 상주하면 다시 통계 정보를 추가해서 Agent Manager에게 'XLogic Script Execute Request Event'로 전달한다.

Agent Manager는 다시 등록된 Agent들에게 모두 전달하고 'XLogic Script Execute Request Event'를 처리할 수 있는 Instance Manage Agent가 XLogic을 실행시킨다.

3.7 Immediate Mode

사용자가 작성한 스크립트를 즉시 실행 시키는 방식을 말한다. 사용자가 EM에서 XLogic을 작성하여 실행시키면 Instance Manage Agent에서 XML 형태로 작성된 XLogic을 Java 객체로 변환하여 실행한다<그림 7>.

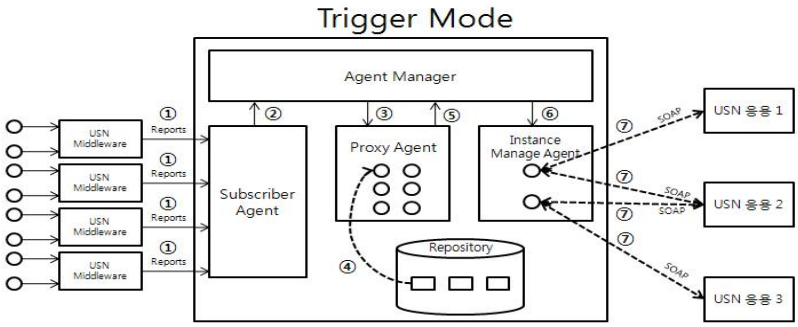


〈그림 7〉 Immediate Mode 이벤트 흐름도

4. 테스트 시나리오

본 비즈니스 로직 프레임워크가 처리해야 할 비즈니스 이벤트를 이질적인 서비스들과 연계하여 처리하는 과정을 검증하기 위해 다음과 같은 시나리오를 통해 테스트 하였다.

1. 00기업 소속 야구단 선수 A씨는 지금 선수촌에 입주하여 한 달간 집중 훈련을 받고 있다.
2. A씨는 아침식사를 하러 선수촌에 있는 식당에 들어선다.



〈그림 6〉 Trigger Mode의 이벤트 흐름

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xlogic : XLogicScript xmlns : xlogic = "urn :spoon : xlogic : script : xsd : 1"
 xmlns : xsi = "http://www.w3.org/2001/XMLSchema-instance"
 xsi : schemaLocation = "urn : spoon : xlogic : script : xsd : 1 http://spoon.smu.ac.kr/xlogic.xsd">
  <!-- 위치 1-->
  <xlogic : set name = "tags" select = "//tag" > ${_REPORTS} < /xlogic : set>
  <!-- 위치 2-->
   <xlogic : invokeWebService service = "PlayerInfoService"
  url = "http://team.oocorp.com/pm/baseball" name = "find_player">
  <rawtags xmlns = "oocorp">
   <xlogic : iterator name = "tag" source = "tags">
    <tag>${tag}</tag>
   </xlogic : iterator>
  </rawtags>
 </xlogic : invokeWebService>

 <xlogic : set name = "player" select = "//id" > ${find_player} < /xlogic : set>
  <!-- 위치 3-->
   <xlogic : invokeWebService service = "RestaurantPayService"
  url = "http://restaurant.xxxcorp.com/PayService"
  name = "restaurant_pay_result">
  <foods xmlns = "xxxcorp">
   <xlogic : iterator name = "tag" source = "tags">
    <food>${tag}</food>
   </xlogic : iterator>
  </foods>
 </xlogic : invokeWebService>
  <!-- 위치 4-->
   <xlogic : invokeWebService service = "CalorieService"
  url = "http://restaurant.xxxcorp.com/CalorieService"
  name = "calorie_result">
  <foods xmlns = "xxxcorp">
   <xlogic : iterator name = "tag" source = "tags">
    <food>${tag}</food>
   </xlogic : iterator>
  </foods>
 </xlogic : invokeWebService>
   <xlogic : set name = "calorie" select = "//calorie">
  ${calorie_result}
 </xlogic : set>
  <!-- 위치 5-->
   <xlogic : invokeWebService service = "PlayerManagementService"
  url = "http://team.oocorp.com/pm/baseball" name = "manage_result">
  <updateFoodLog xmlns = "oocorp">
   <player>${player}</player>
   <calorie>${calorie}</calorie>
  </updateFoodLog>
 </xlogic : invokeWebService>
</xlogic : XLogicScript>
```

3. 착용하고 있는 ID카드에 내장된 RFID로 소속 선수임을 확인 받고 식당에 입장한다.
4. A씨는 평소 즐겨먹는 음식을 위주로 쟁반에 음식들을 담는다.
5. 쟁반을 들고 RFID 안테나가 설치된 곳을 지나면,
   a. 식대가 계산되어 식당 측의 “종합 결제 시스템”에 전달되고, 전달된 식대는 월 말에 00기업으로 일괄 청구된다.
   b. 선수별 신상관리를 위해 음식의 종합 칼로리가 계산되어 00기업의 “선수 관리 전사 시스템”으로 전송된다.

다음은 이러한 시나리오를 XLogic으로 변환한 결과이다. XLogic을 위한 시나리오의 작성은 XML 프로그래밍을 의미한다.

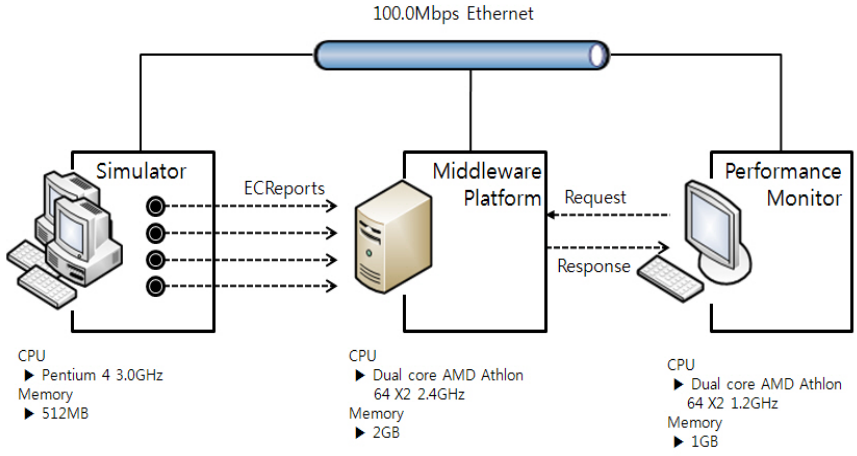
XLogic의 수행 과정은 다음과 같다.

1. 태그 정보가 본 미들웨어 플랫폼의 Subscriber Agent에 도착한다.
2. Trigger 모드의 작동 방법에 따라 위의 XLogic이 실행된다.
3. 태그 정보들을 변수로 받는다(위치 1).
4. 유입된 태그 ID에서 선수 태그 ID를 변수로 받아온다(위치 2).
5. 음식의 태그 ID를 식당 측 “Restaurant Pay Service”로 전송하여 계산을 한다(위치 3).
6. 음식의 태그 ID를 식당 측 “Calorie Service”로 전송하여 칼로리를 구한다(위치 4).
7. 구해진 칼로리를 해당 선수의 일지에 기록하기 위해 ○○기업 측 “Player Management Service”로 전송한다(위치 5).

실행 결과 이벤트에 대해 작성한 시나리오가 이질적인 서비스들과 상호 작용하여 실행 순서에 맞게 수행됨을 확인할 수 있었다.

5. 성능 테스트 결과

본 비즈니스 로직 프레임워크의 성능을 측정하기 위해 <그림 8>와 같은 테스트 환경



〈그림 8〉 성능 테스트 환경

을 구축하였다. 본 성능 테스트 환경은 BEA Weblogic RFID Edge Server Ver. 2.2의 리더 시뮬레이터 및 본 미들웨어 플랫폼, 성능 모니터로 구성하였다. 성능 테스트의 목적은 본 논문에서 제시하는 비즈니스 프레임워크의 처리 효율성(efficiency)및 확장성(scalability)를 검증하기 위한 것이다.

첫 번째 테스트로 Repository에 저장된 XML 형태의 XLogic을 Java 객체로 파싱하는 속도를 측정하였다. 테스트를 위해 다양한 길이의 XLogic을 작성하고 java.lang 패키지의 System 클래스를 사용하여 파싱을 시작하기 전과 후의 시간 차이를 계산하였다. 파싱 시간의 평균값을 구하기 위해 각 XLogic들을 20번씩 파싱하여 그 때마다 걸린 시간의 평균치를 계산하여 <표 4>와 같은 결과를 얻었다.

이 테스트를 통해 XLogic을 실행하는데 소요되는 시간에 대한 가이드 라인을 제시할 수 있다.

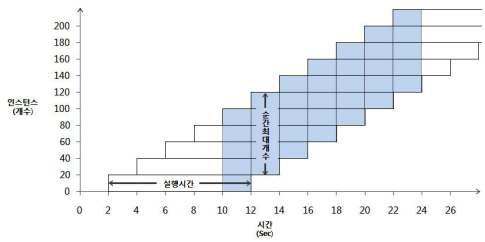
두 번째 테스트는 이벤트 처리시 JVM에서 차지하는 메모리 비율을 측정하였다. 테스트를 위해 리더 시뮬레이터가 RFID 리더 장비를 에뮬레이션하여 GID-64 형식의 EPC 데이터를 가진 XML 형태의 ECReports를 본 미들웨어 플랫폼에 전송한다. 본 미들웨어 플랫폼은 ECReports에 맞는 XLogic을 실행 시키는 동안 기본 힙 영역 32MB, 최대 힙 영역 156MB로 설정된 JVM을 성능 통계 감시 도구(jvmstat)를 사용하여 메모리 영역을 측정한다. 성능 모니터는 EM을 통해 XLogic의 실행 횟수를 화면에 출력한다. 성능 측정을 위해 40대의 시뮬레이터에서 송신하는 각각의 ECReports에 부합하는 XLogic을 약 10초 간 실행할 수 있도록 다음과 같이 작성하였다.

실험 환경에서 전송 주기에 따른 순간적인 최대 XLogic의 실행 개수를 계산하여(<그림 9>) Instance Manage Agent의 최대 실행 가능한 pool 크기를 설정하였다.

〈표 4〉 XLogic 라인수에 따른 파싱시간

XLogic(Line)	시간(ms)
10	0.36
100	1.34
1000	3.92
1500	6.17
2000	8.83

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xlogic : XLogicScript xmlns : xlogic = "urn : spoon : xlogic : script : xsd : 1"
xmlns : xsi = "http://www.w3.org/2001/XMLSchema instance"
xsi : schemaLocation = "urn : spoon : xlogic : script : xsd : 1
http://spoon.smu.ac.kr/xlogic.xsd ">
    <xlogic : set name = "tags" select = "//tag" > ${_REPORTS} </xlogic : set>
    <xlogic : print > ${"Test Script Start"} < /xlogic : print>
    <xlogic : wait>${10000}</xlogic : wait>
    <xlogic : print>${"Test Script End"}</xlogic : print>
</xlogic : XLogicScript>
```



〈그림 9〉 전송 주기 2초일 때 XLogic의 순간최대 실행 개수(예)

가로의 막대 길이는 XLogic의 실행 시간을 나타내며 세로의 색칠된 높이는 XLogic의 순간적인 최대 실행 개수를 나타낸다. 리포트의 전송 주기 별로 나타내면 <표 5>와 같다.

측정 시간은 각 주기마다 24시간으로 지정하였고 리포트 전송 주기 별 Heap 메모리 영역 차지 비율을 <표 6>과 같이 얻었다.

JVM의 메모리영역은 초기 Heap 메모리 영역과 최대 Heap 메모리 영역을 설정할 수 있는데 초기 메모리 영역은 필요에 따라 메모리 영역이 증가하게 되고 그 영역이 최대 메모리 영역의 크기를 넘어설 경우 메모리 부족 현상이 나타난다. 테스트 결과 Heap 메모리의 시간 전송 주기별 증가는 <그림 10>과 같다. 테스트 결과 리포트의 전송주기가 감소하여 리포트의 양이 급격히 증가하여도 일정한 메모리 영역으로 수렴하고 있는 것을 알 수 있다.

〈표 5〉 전송 주기 별 Pool 크기

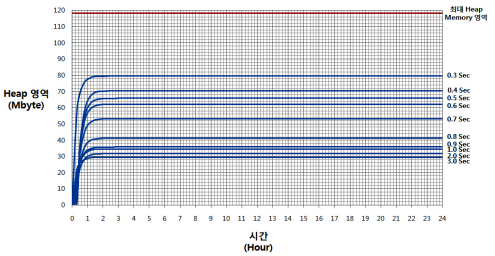
리포트 전송 주기(초)	순간 최대 실행 개수(회)	설정한 Pool 크기(개)
3.0	160	200
2.0	200	250
1.0	400	450
0.9	440	500
0.8	500	550
0.7	560	600
0.6	640	700
0.5	800	900
0.4	1000	1050
0.3	1320	1350

〈표 6〉 리포트 전송 주기 별 Heap 메모리 영역

리포트 전송 주기(초)	Heap 메모리 영역	총 실행 횟수(회)
3.0	29.562	1151695
2.0	31.787	1727893
1.0	34.579	3455931
0.9	35.824	3839817
0.8	41.341	4319952
0.7	53.239	4937131
0.6	62.199	5759932
0.5	65.840	6911935
0.4	70.543	8639538
0.3	79.578	11518936

6. 결론 및 향후 연구

새롭게 펼쳐지는 유비쿼터스 기반의 미래



〈그림 10〉 시간대 별 Heap 메모리 영역의 변화 곡선

사회를 구현하기 위해서 핵심 인프라 환경이라고 할 수 있는 RFID/USN 기술에 대한 많은 연구가 이루어져 왔다. 특히 소프트웨어 분야로서 미들웨어 플랫폼에 대한 연구 및 개발이 활발하게 진행되어 왔으며, 이미 수많은 응용 프로그램들이 다양한 미들웨어를 기반으로 서비스를 제공해오고 있다. 따라서, 새로운 시스템을 위한 미들웨어를 개발하는 것만큼이나 기존 미들웨어들이 새로운 서비스를 위해 적절하게 협업할 수 있는 플랫폼에 대한 연구가 중요하다.

본 논문에서는 RFID/USN 환경에서 기존의 이기종 시스템들이 사용자가 원하는 시나리오에 따라 적절하게 협업할 수 있는 비즈니스 로직 프레임워크를 개발하고 이에 대한 다양한 성능분석을 통해 현실적으로 사용 가능한 수준의 프레임워크임을 검증하였다.

다만, 사용자가 시나리오를 작성하기 위해서 XML을 활용해야 하는 점은, 향후 일반 사용자들의 접근성 및 사용자 편의성을 향상시키기 위한 시각화된 시나리오 작성 도구의 개발을 통해 개선해야 할 것이다. 또한, 이기종 서비스 시스템간의 협업 효율성 및 성능 제고를 위한 방안에 대해 지속적인 연구를 수행할 계획이다.

참 고 문 헌

[1] 백종현, 김형석, 김영호, 한상인, “SOA 플랫폼 분석과 시장전망”, 정보과학회지, 제25권, 제21호, 2007.

[2] 김관중, 김선지, 김내수, 표철식, “USN 서비스 및 시장 동향”, 정보과학회지, 제25권, 제12호, 2007.

[3] W. Heinzelman, A. Murphy, H. Carvalho, and M. Perillo, “Middleware to Support Sensor Network Applications,” IEEE Network Magazine Special Issue Jan. 2004.

[4] Shuoqi Li, Sang H. Son, and John A. Stankovic, “Event Detection Services Using Data Service Middleware in Distributed Sensor Networks,” Information Proc. In Sensor Networks, Apr. 2003, LNCS 2634, pp. 502-517.

[5] T. Liu and M. Martonosi, “Impala : A Middleware System for Managing Autonomic,” Parallel Sensor Systems, Proc. ACM SIGPLAN Symp. Principles and Practice of Parallel Programming, 2003, pp. 107-118.

[6] S. R. Madden, M. J. Franklin, and J. M. Hellerstein, “TinyDB : An Acquisitional Query Proccessing System for Sensor Networks.” ACM TODS, Vol. 30, No. 1, 2005, pp. 122-173

[7] Yong Yao and J. E. Gehrke, “The Cougar Approach to In-Network Query Processing in Sensor Networks,” SIGMD RECORD. Vol. 31, No. 3, Sep. 2002.

[8] 김민수, 이용준, 박종현, “USN 미들웨어 기술개발 동향”, 전자통신동향분석, 제22권, 제3호, 2007.

[9] Yung Bok Kim, Chul-Su Kim, Jun

Wook Lee, “A Middleware Platform Based on Multi-Agents for u-Healthcare Services with Sensor Networks,” Symposium on Applied Computing Proceedings of the 2008 ACM symposium on Applied computing.

[10] Oracle, Gartner, “Java One Conference in San Francisco,” 2006.

[11] An Oracle White Paper “The Instantly Responsive Enterprise : Integrating Business Process Management and Complex Event Processing,” http://www.oracle.com/technologies/bpm/docs/instantly-responsive-enterprise-white-paper.pdf, 2008.

[12] BEA, “BEA Event Driven SOA 솔루션,” http://kr.bea.com/BEA_SolutionGuide_2.pdf.

[13] Matthias Heiler Solution Engineer SAP NetWeaver SAP Deutschland AG and Co. KG, “Enterprise SOA Event-Driven Architecture in the Context of Enterprise SOA.”

[14] Global Headquarters 3303 Hillview Avenue Palo Alto, CA 94304 “Event-Driven SOA : A Better Way to SOA,” http://www.tibco.com/resources/solutions/soa/event-driven_soa_wp.pdf.

저 자 소 개

장준호	(E-mail : jchang@smu.ac.kr)
1990년	서울대학교 계산통계학과 (학사)
1992년	서울대학교 전산학 (석사)
1998년	서울대학교 전산학 (박사)
1998년~2003년	아이투 테크놀로지스(i2 Technologies) 컨설팅 이사
2003년~현재	상명대학교 디지털미디어학부 교수
2004년~2006년	정보통신연구진흥원 인력양성사업단장