유비쿼터스 컴퓨팅 환경을 위한 에이전트 시스템과 인포메이션 버스 어댑터간 상호 운용성을 위한 기법

박 상 용[†] · 한 승 욱[††] · 윤 희 용[†††]

요 약

유비쿼터스 컴퓨팅 환경의 관심과 요구가 고조됨에 따라 다양한 상황에 능동적이고 지능적인 에이전트의 역할이 점차 중요시 되고 있다. 이러한 에이전트는 ACL(Agent Communication Language)을 통신 언어로 사용하여 에이전트간 자율적인 상호 운용성 문제를 해결하기 위해 정보를 교환 한다. 본 논문에서는 소프트웨어 에이전트의 국제표준화 단체인 FIPA(Foundation for Intelligent Physical Agents)의 에이전트 프레임워크와 CORBA 이벤트 서비스 기반으로 자체 개발한 인포메이션 버스 어댑터간의 효율적인 통신을 보장하는 상호 운용성 기법을 제안한다. 비에이전트 플랫폼인 인포메이션 버스 어댑터와 JADE 플랫폼간 통신(메시지 교환)을 보장하기 위한 인터페이스인 EMTI(Efficient Message Transport Interface) 설계 및 구현 방법에 대해 살펴보고, 이기종 플랫폼간 다량의 메시지를 송수신하여 그 안정성과 성능에 대해 평가한다.

키워드 : 유비쿼터스 컴퓨팅, 에이전트, FIPA, EMTI, 인포메이션 버스 어댑터, CORBA, 이벤트 서비스

The Methodology for Interoperability between Agent Framework and Information Bus Adapter for Ubiquitous Computing Environments

Sang Yong Park[†] · SeungWok Han[††] · Hee Yong Youn[†††]

ABSTRACT

The role of autonomic and intelligent agents in various environments is getting more important as demand on ubiquitous computing grows. The agents exchange information using the ACL (Agent Communication Language) to autonomously solve the problems. In this paper we propose a way of efficient interoperability technique between the agent framework built based on the international standard FIPA(Foundation for Intelligent Physical Agents) and the CORBA event service-based information bus adapter developed by us. The design and implementation of EMTI (Efficient Message Transport Interface) allowing communication between the information bus adapter which is non-agent platform and JADE platform are presented, and its performance is evaluated by letting them exchange a large amount of messages.

Key Words : Ubiquitous Computing, Agent, FIPA, EMTI, Information Bus Adapter, CORBA, Event Service

1. 서 론

최근 유비쿼터스 컴퓨팅 환경에 대한 관심과 요구사항이 증대됨에 따라 다양한 상황에 능동적이고 자가 성장이 가능한 에이전트 플랫폼의 연구가 활발히 진행되고 있다. 에이전트는 동적으로 개방된 환경에서 효율적으로 작업을 수행하고 문제를 스스로 해결하는 능력을 가진다[2, 6, 7]. 소프트웨어 에이전트 기술은 초기에는 인공지능 연구자들에 의해 추론 및 판단 기능을 수행할 수 있는 독립적인 작업 형태의기술로 시작하였으나, 그 이후 분산 환경에서 멀티 에이전트[8-15] 간의 협동을 통해 인간의 특정한 임무를 대신하여 수행함으로써 임무를 효과적으로 해결하는 중요한 수단으로 발전하고 있다.

기본적으로 소프트웨어 에이전트 국제 표준화 단체인 FIPA[3] 에이전트 프레임워크는 에이전트 통신 언어인 ACL (Agent Communication Language)을 사용하여 멀티 에이전트간에 메시지를 교환한다. 이탈리아 통신 업체의 TI LAB에서 개발된 JADE 에이전트 플랫폼[2]은 FIPA 스펙을 준수하며 개발자와 사용자를 위한 GUI 환경을 제공함으로써 에이전트 관리의 효율성을 높이는 대표적인 에이전트 프레임워크이다. 또한, 스니퍼와 인트로스펙터와 같은 툴을 제공함으로써 에이전트간 통신 메시지를 추적할 수 있다.

에이전트 프레임워크에서 사용되는 통신 언어인 ACL은 에이전트들 사이에서 자율적인 문제 해결을 위해 사용되며

※ 이 논문은 21세기 프론티어 연구개발 사업의 일환으로 추진되고 있는 정보통신부의 유비쿼터스컴퓨팅및네트워크원천기반기술개발사업과 2006년도 두뇌한국21사업에 의하여 지원되었음.
† 준 회 원 : LG전자
†† 정 회 원 : 성균관대학교 정보통신공학부 박사과정
††† 종신회원 : 성균관대학교 정보통신공학부 교수(교신저자)
논문접수 : 2005년 10월 18일, 심사완료 : 2006년 6월 1일

기존의 통신 프로토콜과는 상당한 차이가 있다. 또한, ACL은 에이전트간 정보교환에 필요한 충분한 표현력과 적절한 의미구조를 갖고 있으며, 무엇보다 에이전트의 가장 큰 특성 중의 하나인 자율성(autonomy)을 보장 한다[6, 7]. 하지만, 이러한 장점에도 불구하고 기존의 에이전트 플랫폼과 비에이전트 플랫폼간의 원활한 통신을 지원하기 위해서는 플랫폼 내부의 변경이나 새로운 통신 방법이 제시되어야 한다.

유비쿼터스 환경에서는 기기들이 다양한 방식으로 통신하며 의미와 정보를 전달하기 때문에 각 컴퓨팅환경에 적합한 프레임워크 방식으로 개발되어야 한다. 따라서, 기존의 CORBA기반으로 작성된 기업 솔루션, 전자상거래와 같은 기존 시스템을 변경하지 않고 에이전트의 지능적인 특성을 활용하여 서비스를 제공하기 위해 ACL을 사용하는 에이전트와 비에이전트 플랫폼간의 통신을 위한 매개체가 필요하다. 이러한 문제점을 해결하기 위해 본 논문에서는 이기종 플랫폼간 상호 통신을 보장하는 방법으로 EMTI (Efficient Message Transport Interface)를 정의하고 CORBA 이벤트 서비스[5] 기반으로 자체 개발된 인포메이션 버스 어댑터[1]와 JADE 플랫폼을 이용하여 해결 방법을 제안한다.

본 논문의 2장에서는 관련 연구로서 멀티 에이전트 개발 및 관리를 용이하게 하는 JADE 플랫폼의 특징 및 구성 요소와 CORBA 이벤트 서비스에 대해 살펴 본다. 3장에서는 제안하는 시스템 구현 방법에 대해 구체적으로 언급하고, 4장에서는 위에서 언급한 인포메이션 버스 어댑터와 에이전트 플랫폼간 성능을 평가 분석한다. 마지막으로 향후 연구 방향에 대해 언급하고, 본 논문의 결론을 맺는다.

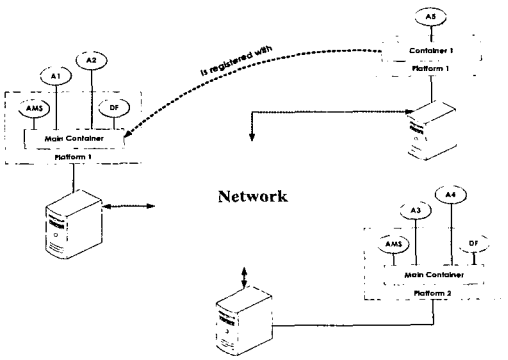
2. 관련 연구

본 장에서는 멀티에이전트의 정의 및 JADE 와 CORBA 이벤트 서비스에 대하여 살펴본다.

2.1 JADE(Java Agent Development Framework)

멀티 에이전트 시스템 환경의 에이전트란 분산환경에서 상호 협력을 통해 작업을 수행하는 컴퓨터 프로그램을 말한다[1]. 독립적인 응용 프로그램으로는 해결할 수 없는 보다 복잡한 서비스를 여러 에이전트들의 상호 협력을 통해 제공할 수 있다. 또한, 사용자 컴퓨터에는 많은 에이전트가 존재하기 때문에 에이전트의 사용법을 모르더라도 자주 이용하는 에이전트를 통해 자신도 모르게 다른 에이전트를 사용하게 된다. 멀티 에이전트 시스템은 서로 협력하는 에이전트로 구성되어 있다. 그리고 에이전트의 기능에 대한 요구사항이 증대되면 새로운 기능을 가진 에이전트를 시스템에 추가함으로써 시스템을 쉽게 확장시킬 수 있다. JADE는 이러한 멀티 에이전트의 개발을 쉽고 빠르게 해주는 하나의 미들웨어 프레임워크로서 FIPA 스펙을 준수한다. 다음은 JADE의 구성 요소를 나타낸다.

- AMS(Agent Management System) : 에이전트 생성 및 제거를 담당하고 네이밍 서비스를 지원한다.



(그림 1) JADE 플랫폼

- DF(Directory Facilitator) : 옐로우 서비스를 지원함으로써 에이전트는 네트워크 상에 존재하는 다른 에이전트를 찾는 방법을 제공한다.

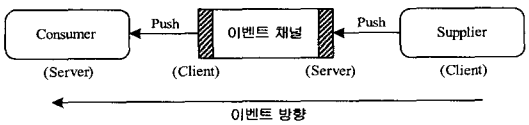
(그림 1)과 같이 컨테이너는 하나 이상의 에이전트를 실행하기 위한 환경을 제공해 준다. 원격에서 생성된 에이전트를 메인 컨테이너에 등록 한 후 네트워크를 통해 원격 에이전트와 통신을 수행한다. 하나의 독립적인 JADE 플랫폼에서는 반드시 하나의 메인 컨테이너만을 가질 수 있다.

2.2 CORBA이벤트 서비스

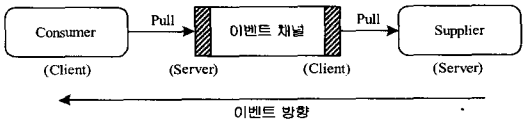
이벤트 서비스[5]는 CORBA 호출 모델의 제한적인 특성을 개선하기 위해 설계된 비동기적 메시지 전송을 지원하며, 하나 이상의 공급자가 하나 이상의 소비자에게 메시지를 전송하는 것을 가능케 한다. CORBA 이벤트 서비스[5]는 (그림 2)와 같이 supplier, consumer, 그리고 이벤트 채널로 구성되어 있는데, supplier와 consumer는 이벤트 채널에 연결되며 각각 이벤트를 생성하고 이벤트를 사용한다. 이벤트 채널은 consumer측에서 supplier에 대한 어떠한 정보도 요구하지 않으면서 이벤트를 전송할 수 있도록 하고 이벤트 서비스의 중개자로써 supplier와 consumer의 등록, 삭제, 데이터 전송의 신뢰성 및 에러 컨트롤과 관련된 일을 수행한다.

이벤트 서비스는 크게 push모델과 pull모델을 제공한다. Push 모델에서는 (그림 2)와 같이 supplier는 이벤트 채널에 이벤트를 push하고, 이벤트 채널은 받은 이벤트를 consumer에게 넘겨준다.

반면 pull 모델에서는 (그림 3)과 같이 push 모델과는 반대로 이벤트 채널 중개자에게 이벤트를 요구하고 이벤트 채널은 supplier에게 특정의 이벤트를 달라고 요청한다.



(그림 2) Push 모델



(그림 3) Pull 모델

CORBA 이벤트 서비스를 이용하여 애플리케이션을 개발할 때push 모델과 pull 모델의 조합을 통해 4가지 모델 - push/push, push/pull, pull/pull, pull/push - 을 사용할 수 있는데 개발하고자 하는 애플리케이션의 특성과 디자인을 고려하여 선택해야 한다.

3. 제안하는 시스템의 설계와 구현

본 장에서는 CORBA이벤트 서비스가 가지고 있는 문제점을 개선하고 에이전트 플랫폼과 효율적인 메시지 통신을 위한 시스템 설계 및 구현방법에 대해 소개한다.

3.1 EMTI(Efficient Message Transport Interface)

3.1.1 전체 구조

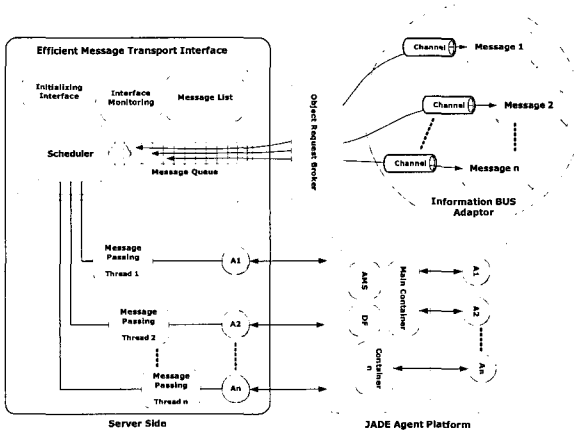
(그림 4)는 비에이전트 플랫폼인 인포메이션 버스 어댑터와 JADE 플랫폼간 통신을 위한 전체 구조를 도식한 것이다. 전체 구조도는 크게 클라이언트 사이드의 인포메이션 버스 어댑터, JADE 플랫폼, 그리고 이 두 플랫폼간의 브리지 역할을 하는 서버 측의 EMTI인터페이스로 나뉘어진다.

• 클라이언트 측

특정 이벤트가 발생하면 인포메이션 버스 어댑터에 이벤트를 전송할 전용 채널이 생성된다. 채널은 이벤트의 종류와 내용에 따라 자동적으로 생성된다. 또한, 채널은 이벤트의 수에 따라 동적으로 증가되거나 감소되어 최적의 숫자를 유지함으로써 채널 관리에 따른 오버헤드를 줄인다. 발생한 이벤트는 네이밍 서비스인 omniNames를 이용하여 어느 객체에게 메시지를 전달할 것인가를 찾고, 그 후 ORB(Object Request Broker)를 통해 서버 구현 객체에게 전달된다. 어댑터는 기본적으로 push 및 pull 모델을 지원 한다.

• 서버 측

서버 측에서는 EMTIServant 클래스의 인스턴스를 생성하여 ORB및 EMTI 인터페이스를 초기화하고 네이밍 서비스에 네이밍 컨텍스트를 등록한다. 또한, 에이전트 플랫폼에서 이벤트에 대한 결과를 저장할 MessageList를 생성한다. 서버가 초기화 된 후, EMTI인터페이스에서는 JADE 플랫폼의 리모트 에이전트와 통신하기 위한 로컬 에이전트를 createNewAgent() 메소드를 이용하여 생성한다. 리모트 에이전트 플랫폼의 AMS(Agent Management System)에이전트에게 질의를 보내 현재 등록되어 있는 에이전트의 리스트를 받아온 후 활성화된 해당 에이전트와 통신하게 된다. 에이전트는 기본적으로 activated와 suspend의 상태를 가진다.



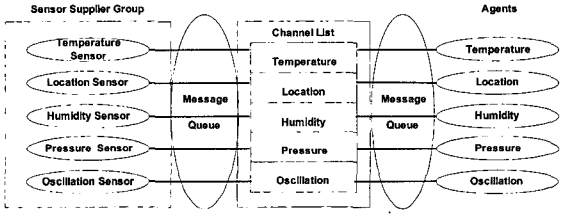
(그림 4) 전체 구조

• 에이전트 플랫폼

멀티 에이전트 시스템에서는 에이전트 사이의 정보교환이 매우 빈번하게 일어나며, 이 과정은 AMS에 의하여 제어된다. 서버에서 수신된 이벤트 데이터는 컨텍스트의 종류에 따라 각기 다른 성격의 에이전트가 수행된다. 에이전트 플랫폼에서는 해당 동작을 수행한 후, 결과 값을 요청한 곳으로 보내기 위해 EMTI인터페이스내의 Message List에 값을 저장한다.

(그림 5)는 메시지 큐에 전송되는 컨텍스트의 종류에 따라 에이전트에 메시지를 전달하는 구조를 나타낸 것이다.

멀티 에이전트는 특정 컨텍스트 공급자와 소비자간의 이벤트를 전달하도록 구성되어 있다. EMTI에 적용된 컨텍스트 기반 통신은 상황정보 공급자와 소비자 사이에 상황에 대한 이벤트를 통하여 통신 함으로써 공급자와 소비자간의 통신을 분리하였다.



(그림 5) 컨텍스트 종류에 따른 에이전트

3.1.2 IDL 명세

(그림 6)는 IDL을 이용하여 클라이언트에서 서버로 접근하기 위한 인터페이스를 보여준다. IDL내에는 Events 인터페이스가 정의되어 있으며 이를 통해 외부에서의 접근을 가능케 한다. passing_message(in EventData data) 메소드는 어댑터에서 발생한 이벤트 데이터를 수신하여 에이전트 플랫폼으로 메시지를 전달하는 역할을 한다. 그 외 에이전트를 찾거나 에이전트 리스트를 파악하는 등의 보조적 역할을 하는 메소드가 있다. IDL은 프리컴파일러에 의해서 컴파일되어 서버 클래스의 구현을 위한 skeleton 및 stub코드를 생성하며, 생성된 코드를 기반으로 서버의 구현객체를 정의한

```
module emti
{
        interface  Events
        {
            string passing _message(in EventData data);
            string search_specified_agent(in string agent_name);
            string get_agent_list(in short list_index);
            short get_agent_count();
            .....
            .....
            oneway void shutdown ();
        };
};
```

(그림 6) IDL 명세

다. IDL로 인터페이스를 통일하는 기법으로, Java, C++, python 등의 언어에 상관없이 클라이언트가 구현 확장도어 ORB를 통해 서버와 연결할 수 있는 장점을 가진다.

3.1.3 각 모듈의 기능

• IS(Initializing System)

IS는 이기종 플랫폼간에 통신을 하기 위한 인터페이스를 초기화 한다. 메시지를 임시 저장하기 위한 arrayList 및 메시지 큐를 생성하고, 스케줄러를 활성화 시킨다.

• IFM(Interface Monitoring)

EMTI 인터페이스의 모니터링을 담당한다. 메시지 전달 스레드의 오버헤드를 체크하고, 작업 수행을 모니터링 한다.

• ML(Message List)

에이전트 플랫폼에서 에이전트간 협업에 의해 처리된 결과 데이터는 EMTI 인터페이스를 통해 다시 서버 측으로 전달된다. ML에서는 리턴된 데이터를 arrayList에 의해 생성된 버퍼에 저장하여 데이터 손실에 따른 문제점을 해결하고 안정성을 향상시킨다.

• Message Scheduler

스케줄러는 다중 스레드를 이용하여 작업을 스케줄링 한다. 스케줄러가 시작될 때 getEventMessage()가 실행되고, getEventMessage()는 메시지 큐에서부터 작업의 내용을 가져오는 디스패처(Dispatcher)의 역할을 수행한다. 또한, 이벤트 메시지의 컨텍스트 타입 및 내용을 검사한 후 메시지 큐에 쌓인 메시지를 무작위로 추출하여 처리한다.

• MQ(Message Queue)

EMTI내에 메시지를 임시 저장하는 가변적인 큐를 두고 큐에 담긴 메시지의 종류와 속성을 파악하여 해당 메시지를 추출하여 통신한다.

3.2 인포메이션 버스 어댑터

인포메이션 버스 어댑터는 omniORB[4]에서 이벤트 서비스를 담당하는 omniEvents를 기반으로 개발 되었다. 기존 omniEvents는 이벤트 데이터 처리량과 상관없이 정적으로 채널을 생성하고 관리한다. 이러한 방식은 고정적으로 컴퓨팅 리소스를 점유하며, 유비쿼터스 컴퓨팅 환경과 같이 동적으로 변화하고 센서 및 RFID등에서 수시로 발생한 이벤트 메시지를 처리하기에는 적절하지 못하다. 인포메이션 버스 어댑터는 omniEvents서비스 사용시 복잡하게 사용되는 코드를 캡슐화하는 방법으로 (그림 7)과 같이 다양한 클래스 인터페이스를 지원한다. 이는 내부구조를 정확히 파악하지 못한 상태에서도 구현이 용이하다는 장점이 있다. 또한, 기존 이벤트 서비스에서는 제공되지 않는 신뢰성 보장을 위해 부가적인 로직을 이벤트 채널 서비스에 추가하였다.

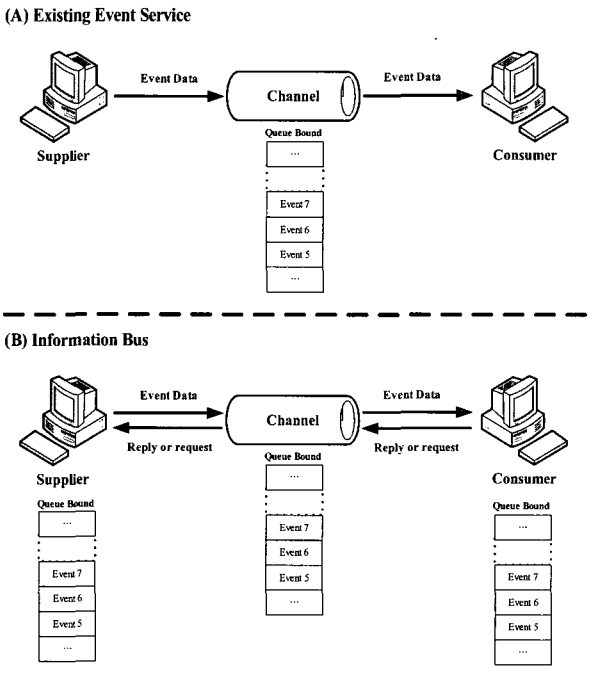
```
#define CH_DEF      0
#define CH_SUP      1
#define CH_CON      2
typedef void (CALLBACK * MSGRECV)(void *);

extern "C" __declspec(dllimport) unsigned int uTAdapterCreate();
extern "C" __declspec(dllimport) void uTAdapterDelete(unsigned int nInstance);
extern "C" __declspec(dllimport) int uTAdapterInitialize(unsigned int nInstance, const char* pszClassOfChannel, const
                                 char* pszDetailOfChannel, int channel_op, bool bNameservice = true);
extern "C" __declspec(dllimport) int uTAdapterUninitialize(unsigned int nInstance);
extern "C" __declspec(dllimport) int uTPushProxyConnect(unsigned int nInstance, bool bSupplier = true) ;
extern "C" __declspec(dllimport) int uTPullProxyConnect(unsigned int nInstance, bool bSupplier = true) ;
extern "C" __declspec(dllimport) int uTProxyDisconnect(unsigned int nInstance);
extern "C" __declspec(dllimport) int uTSleep(unsigned int nInstance, int nSleepInterval);
extern "C" __declspec(dllimport) int uTSemaphore_Post(unsigned int nInstance);
extern "C" __declspec(dllimport) int uTSemaphore_Wait(unsigned int nInstance);
extern "C" __declspec(dllimport) int uTMutex_Wait(unsigned int nInstance);
extern "C" __declspec(dllimport) int uTMutex_Signal(unsigned int nInstance);
extern "C" __declspec(dllimport) int uTMessage_Send(unsigned int nInstance, void * szMsg, size_t size, bool back = true);
extern "C" __declspec(dllimport) int uTMsgMethodRegister(unsigned int nInstance, MSGRECV pMsgCallBack);
extern "C" __declspec(dllimport) char* uTGetMessage(unsigned int nInstance);
extern "C" __declspec(dllimport) int uTRegistry_NameService(unsigned int nInstance, const char* pszNameServiceUrl);
```

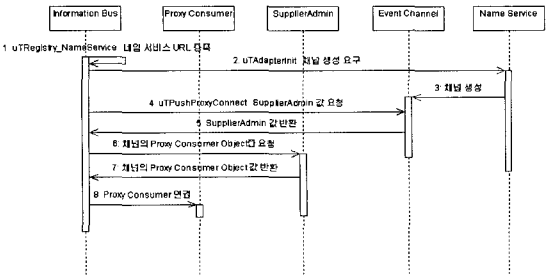
(그림 7) 인포메이션 버스 어댑터 인터페이스

어댑터에는 ORB 네이밍 서비스에 채널 이름을 관리할 수 있는 로직을 추가함으로써 인포메이션 버스 내에서 채널 관리 자동화, 채널 자동 생성/삭제, 그리고 채널 네임 룩업 서비스 기능을 제공한다. 이와 같은 기능은 이벤트 컨텍스트의 종류와 내용에 따라 자동으로 채널을 분리해 생성시키며, consumer에서 받고자 하는 이벤트 데이터 만을 추출하여 서비스를 가능하게 한다. 또한, consumer와 supplier 로직에 필터링 기능을 추가하는 것보다 더 나은 성능이 보장된다는 것을 다음 장의 평가를 통해 보여준다. (그림 8)은 기존 CORBA 이벤트 서비스가 가지고 있는 문제점을 해결하고 성능과 신뢰성을 높이는 방식을 보여준다. 이벤트 채널, consumer 및 supplier마다 각각의 메시지를 저장하는 큐를 두는데, 작동 방식을 살펴보면 다음과 같다. 먼저 송신 측에서는 데이터를 보내기 전에 임시저장소에 메시지를 보관한다. 만약 데이터가 손실되면 저장소의 해당 데이터를 재전송한다. 수신 측에서는 메시지를 받게 되면 ACK신호를 전송하게 되고 송신 측에 저장되어 있던 내용을 삭제한다. 이러한 메커니즘으로 전체 시스템의 신뢰성을 향상시킬 수 있지만, 성능이 다소 떨어지는 문제가 있다.

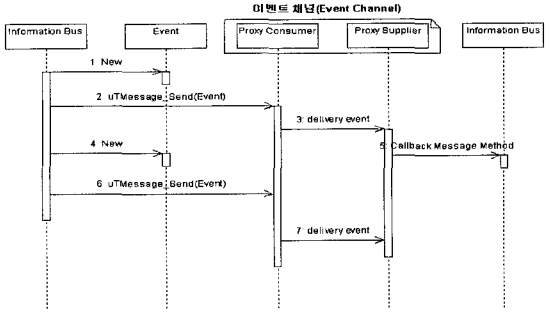
(그림 9)는 어댑터의 채널 접속 과정을 보여준다. 어댑터는 가장 먼저 uTRegistry_NameService() 인터페이스를 이용하여 네이밍 서비스의 URL을 등록한다. 그 후 uTAdapterInit() 인터페이스를 통해 초기화 및 채널을 생성하고 네이밍 서비스에 채널 이름을 설정한다. 채널이 생성된 후 Supplier-Admin값을 이용하여 ProxyConsumer오브젝트를 얻는다. 결과적으로, 반환된 오브젝트를 이용하여 통신을 하게 된다.



(그림 8) CORBA 이벤트 서비스와 어댑터



(그림 9) 채널 접속 시퀀스 다이어그램



(그림 10) 이벤트 전달 시퀀스 다이어그램

(그림 10)은 채널이 생성된 후 로컬에서 원격으로 이벤트가 전달되는 과정을 보여준다. 이벤트 발생시 uTMessage_Send(Event) 인터페이스를 이용하여 어댑터에 전달하며 전달된 이벤트는 (그림 9)와 같은 과정을 통해 얻어진 ProxyConsumer 오브젝트를 가지고 이벤트 채널에 push 또는 pull하게 된다. 이벤트 채널 내에 어떤 이벤트가 도착하면 ProxyConsumer는 ProxySupplier에게 전달되며, 이벤트를 받고자 하는 consumer는 인포메이션 버스 어댑터의 CALLBACK 인터페이스를 이용하여 이벤트를 전달한다.

현재 인포메이션 버스 어댑터는 C++과 Java 언어를 기본으로 지원한다. 또한, 모바일 환경을 위한 Pocket PC용 어댑터, 리눅스 환경을 위한 Linux용 어댑터, 그리고 윈도우용 어댑터를 제공한다. 이와 같이 다양한 플랫폼과 언어를 지원함으로써 동적인 속성이 존재하는 유비쿼터스 컴퓨팅 환경의 요구사항을 충족시킬 수 있다.

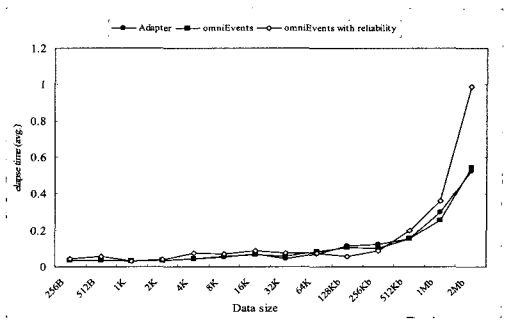
4. 성능 평가

본 장에서는 앞에서 구현한 인포메이션 버스 어댑터 및 EMTI인터페이스를 통해 다량의 메시지를 송수신 함으로써 인터페이스의 효율성에 대해 평가한다.

4.1 인포메이션 버스 어댑터의 성능 평가

(그림 11)은 CORBA 이벤트 서비스 기반으로 개발된 인포메이션 버스 어댑터, 기존의 omniEvents 와 신뢰성을 지원하는 omniEvents 각각의 3가지 케이스에 대한 성능 평가를 보여준다.

성능 평가는 5대의 PC를 가지고 dummy데이터를 256 bytes부터 2M bytes까지 크기를 점차 증가시키면서 전송속도를 측정하였다. 결과 값에 대한 신뢰성을 높이기 위해 각각의 dummy 데이터 전송을 1000회 수행하였다. 신뢰성을



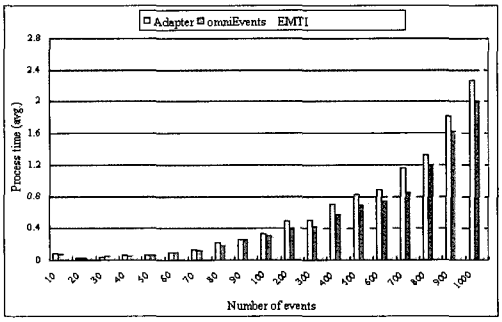
(그림 11) 어댑터 성능

고려하지 않은 omniEvents와 제안된 어댑터의 성능은 근소한 차이를 보임을 알 수 있다. 이는 신뢰성을 지원하면서도 어댑터의 성능은 크게 저하되지 않는 결과를 보여준다.

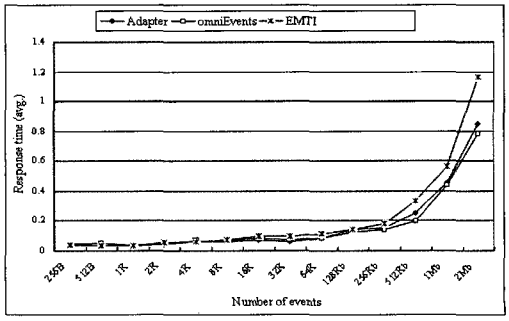
4.2 에이전트 플랫폼과 어댑터간의 통신 성능

에이전트 플랫폼과 어댑터간의 통신 성능 평가는 각각5대의 PC에 어댑터와 EMTI를 적용한 에이전트 플랫폼을 설치한 후 데이터 크기와 메시지 처리속도를 측정함으로써 수행하였다. 어댑터에서 생성한 dummy이벤트 메시지를 에이전트 플랫폼에서 수신한 후 바로 피드백 해 주는 방식으로 수행하였고, 신뢰성 있는 측정을 위해 1000회 반복하였다.

(그림 12)는 이벤트 메시지의 수를 10개에서 1000개까지 증가시키면서 측정한 평균 처리시간을 나타낸다. 신뢰성을 높이기 위해 일정량의 이벤트 메시지를 수신 한 후 각각의 메시지마다 ACK신호를 전송하는 신뢰성 로직을 추가한 제안된 어댑터의 성능이 기존의 신뢰성을 지원한 omniEvents의 성능보다 나은 결과를 보여준다. 또한, 초기 10개에서의 평균 처리 시간은 20개나 30개의 이벤트 메시지 처리 시간보다 높은



(그림 12) 메시지 개수에 따른 처리 속도



(그림 13) 데이터 크기 별 응답 시간

것을 볼 수 있다. 이는 초기 메시지를 전송하는 과정에서 에이전트 플랫폼의 AMS에이전트에게 질의를 보내 해당 에이전트를 찾고 연결하는 과정에서 약간의 지연이 발생하기 때문이다.

(그림 13)은 앞의 (그림 11)에서의 어댑터 자체 성능측정과 동일한 방법으로 수행한 결과를 나타낸다. 데이터 크기를 증가시키면서 어댑터, omniEvents 및 신뢰성을 지원하는 omniEvents의 성능을 측정하였고, 앞에서와 비슷한 결과를 보여준다.

5. 결론 및 향후 연구

본 논문에서는 에이전트 플랫폼과 CORBA 이벤트 서비스 기반의 인포메이션 버스 어댑터간의 통신을 보장하기 위한 인터페이스 설계 및 방법에 대해 알아 보았다.

인포메이션 버스 어댑터는 omniORB의 이벤트 서비스를 담당하는 omniEvents를 기반으로 개발 되었으며, 실시간으로 변화하는 유비쿼터스 컴퓨팅 환경에 적합한 동적 채널 생성, 신뢰성 지원 로직, 채널 관리 자동화 및 채널 네임 룩업 서비스 등을 지원한다. EMTI인터페이스의 궁극적인 목적은 어댑터에서 발생한 이벤트 메시지를 에이전트 플랫폼으로 전달하여 처리하기 위함이다. 또한, ACL 메시지 타입을 지원하지 않는 비에이전트 플랫폼에 적용하여 플랫폼의 효율적인 동작을 보장하고 인터페이스 모니터링 및 에이전트 수행 후의 결과 값을 리턴하기 위한 메시지 리스트를 제공함으로써 안정성을 높이는 것이다. 두 플랫폼간의 성능 평가를 통해서, EMTI는 두 플랫폼간의 신뢰성을 지원하는 동시에 기존 omniEvents보다 메시지 처리 속도 면에서 향상된 것을 볼 수 있었다. 이벤트 메시지에 대한 신뢰성을 지원하면 그렇지 않은 경우 보다 성능이 약간 감소된다. 이러한 특성을 고려하여 적용하고자 하는 시스템이 요구하는 신뢰도에 따라 선택적으로 사용하는 것이 중요하다 하겠다.

향후에는 본 연구를 토대로 제안된 인터페이스내의 에이전트 보안과 신뢰성을 지원하는 동시에 성능을 향상시킬 수 있는 방법에 대한 연구가 진행되어야 하겠다.

참고문헌

[1] Seungwok Han, Hee Yong Youn, "A Middleware Architecture for Community Computing with Intelligent Agents".

[2] JADE, Java Agent Development framework http://jade.cselt.it

[3] FIPA-Foundation for Intelligent Physical Agents. http://www.fipa.org.

[4] Sai-Lai Lo and David Riddoch, "The omniORB version 4.0 User's Guide," AT&T Laboratories Cambridge, October, 2004.

[5] Object Management Group, "Event Service Specification," October, 2004.

[6] 송종철, 정현수, 홍기채, "멀티에이전트 시스템의 연구 동향", December, 2003

[7] 최영미, 윤소정, "멀티 에이전트 시스템에 관한 고찰", September, 1998.

[8] Michael Luck, Peter McBurney, Chris Preist "Agent Technology : Enabling Next Generation Computing," AgentLink community. (2003)

[9] F. Bellifemine, A. Poggi, G. Rimassa. "Developing Multi-Agent Systems with a FIPA-compliant Agent Framework," Software: Practice & Experience, 31:103-128, 2001.

[10] Fabio Kon, Roy Campbell, M. Dennis Mickunas, Klara Nahrstedt, and Francisco J. Ballesteros. "2K: A Distributed Operating System for Dynamic Heterogeneous Environments," in 9th IEEE International Symposium on High Performance Distributed Computing. Pittsburgh. August, 1-4, 2000.

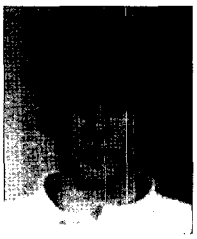
[11] R.Deters, "Scalability & Multi-Agent Systems", 2nd International Workshop Infrastructure for Agents, MAS and Scalable MAS. 5th int.conference on Autonomous Agents, May-June, 2001.

[12] O.F.Rana, K.Stout, "What is Scalability in Multi-Agent Systems", Autonomous Agents 2000, June'00, ACM Press.

[13] Niek Wijngaards, Maarten van Steen, Frances Brazier, "On MAS Scalability", Proc.2nd Int'l Workshop on Infrastructure for Agents, MAS and Scalable MAS. May, 2001.

[14] M. Henning and S. vinosky, "Advanced CORBA Programming with C++" addition-wesley, Boston, 1999.

[15] L.C.Lee,H.S.Nwana,D.T.Ndumu, P De Wilde , "The stability, scalability and performance of multi-agent Systems", BT Technol J Vol.16, No.3, July, 1998. 94.



박 상 용

e-mail : utri@ece.skku.ac.kr

2003년 한국산업기술대학교 컴퓨터공학과(학사)

2005년 성균관대학교 컴퓨터공학과(석사)

2006년~현재 LG전자

관심분야 : 유비쿼터스 컴퓨팅, 미들웨어, 에이전트 시스템

한 승 욱

e-mail : lovedonny@skku.edu

2003년 강남대학교 전자계산학과(학사)

2005년 성균관대학교 정보통신공학부(석사)

2005년~현재 성균관대학교 정보통신공학부 박사과정

1996년~2003년 삼성소프트웨어 멤버십 연구원

관심분야 : 시스템 소프트웨어, 분산처리, 미들웨어

윤 희 용

e-mail : youn@ece.skku.ac.kr

1977년 서울대학교 전기공학과(학사)

1979년 서울대학교 전기공학과(석사)

1988년 Univ. of Massachusetts at Amherst 컴퓨터공학과(박사)

1988년~1991년 Univ. of North Texas. 조교수

1991년~1999년 Univ. of Texas at Arlington, 부교수

1999년~2000년 한국정보통신대학교 교수

2000년~현재 성균관대학교 정보통신공학부 교수

유비쿼터스컴퓨팅기술연구소 소장

관심분야 : 모바일 컴퓨팅, 분산처리, 시스템 소프트웨어