로보틱스 미들웨어 플랫폼의 통신 모델과 컴포넌트 상호 운영성 분석

구용기*, 권순범*, 신동렬*
*성균관대학교 전자·전기 컴퓨터 공학
e-mail : cutedual@skku.edu

Analysis of robotics platform as communication model and component interoperability

Yong-Ki Ku*, Soon-Bum Kwon*, Dong-Ryeol Shin*
*School of Information and Communication Engineering, Sungkyunkwan University

요 약

사람, 컴퓨터, 그리고 사물이 유기적으로 연계되어 다양하고 편리한 서비스를 제공해 주는 컴퓨팅 기술에 대한 연구들이 진행됨에 따라 지능형 로봇 시스템 환경은 분산 환경과 웹이 융합된 새로운 환경으로 변화하고 있다. 이러한 변화를 로봇산업에 적용하기 위해 다양한 연구기관의 개발자들은 이를 지원할 수 있는 로봇 미들웨어 플랫폼의 개발이 많은 노력을 기울이고 있다. 하지만, 지금까지 개발되어진 플랫폼은 통신 오버헤드, 컴포넌트 모델 부재, 통신구조 등 다양한 문제점을 가지고 있다. 이러한 문제점은 부적합한 통신 모델과 상호작용 패턴을 고려하지 않아 파생된다. 본 논문에서는 다양한 로봇 미들웨어 플랫폼의 통신 구조를 “통신 모델”과 “상호작용 패턴”을 분석하고, 시뮬레이션을 통한 결과를 기반으로 로봇 미들웨어 플랫폼의 향후 방향을 제시한다.

1. 서론

최근 유비쿼터스 기술의 발달로 인하여 일상생활에 접목되어 삶의 질을 향상 시킬 수 있는 홈-네트워크 및 로봇산업 기술이 핵심 분야로 관심을 받게 되었으며, 많은 국가의 연구기관에서 해당 분야의 기술 연구를 진행 하였다. 90년대 초반 로봇 시스템은 시리얼, 필드버스 그리고 캔버스와 같은 형태로 제한적인 공간에서 명령어를 송/수신하는 형태로 발전되어 왔다. 그러나 인터넷의 보급과 네트워크 기술의 발달을 기반으로, 네트워크 서비스가 제공되는 어떠한 공간에서도 로봇을 제어할 수 있는 네트워크 로봇 시스템으로 변화하였다. 시스템 통합 기술이 분산 컴퓨팅 환경으로 변함에 따라, 다양한 서비스를 제공하는 지능형 로봇이 등장하였다[1].

로봇 산업은 대부분 하드웨어 위주로 연구되었고, 소프트웨어는 중요하게 인식되지 않았다. 그러나 최근 다양한 서비스를 제공하는 차세대 지능형 로봇이 관심을 끌면서, 하드웨어를 운용하는 소프트웨어의 역할이 매우 커짐에 따라, 로봇을 동작시키기 위해서 다양한 분야의 지식과 펌웨어 및 시스템 소프트웨어에서부터 응용 소프트웨어까지 다양한 계층의 프로그램이 필요하게 되었다. 이러한 소프트웨어는 연구자 서로 간에 공유하는 체제로 정비되어 있지 않기 때문에 로봇 연구자 및 기술자가 독립적으로 개발함에 따라 기술 및 자원의 중복 등 비효율적인 연구 개발을 수행하는 경우가 발생하고 있다. 이러한 상황에서 비용과 시간을 줄이기 위한 로봇 플랫폼은 로봇산업에서 중요한 기술로 부각되고 있다[2].

로봇 플랫폼은 로봇의 복잡한 기능(하드웨어, 센서 및 제어)을 지원하는 소프트웨어의 재사용성을 높이고, 각 기능간의 통신을 쉽고 안정적으로 할 수 있도록 하는 분산객체 기술과 차세대 로봇의 기능측면에서 로봇이 행동하기 위한 정보 및 지식을 네트워크와 정보교환을 통한 “로봇기술과 네트워크와의 통합”기술들이 중요한 이슈로 떠오르고 있다. 이러한 로봇 플랫폼에 대한 필요성으로 개발자와 연구자들은 로봇 플랫폼의 구조를 크게 4가지 계층구조를 기반으로 플랫폼을 구성하였다[그림 1].

[그림 1] 로봇 플랫폼의 계층구조

OROCOS, CORBA, Orca, ICE[3] 등 대부분의 플랫폼 연구들은 운영환경과 응용프로그램에 대한 유연성을 목표를 가지고 있으며, 이에 통신 계층과 서비스 계층의 구성이 로봇 플랫폼을 구성하는데 있어서 핵심 요소가 된다. 통신 계층을 구성하는데 있어서 기존에 안정성 및 유용성이 검증된 통신 플랫폼(CORBA, Ice 등)을 사용하는 방법과 독자적인 통신 구조(Publish/Subscribe, Shared memory 등)를 확립하는 두 가지 방법으로 나뉘어 진다.

[표 1] 기존 로봇 플랫폼의 통신구조

로봇 미들웨어 플랫폼	통신 구조
Miro	CORBA
RT	CORBA
MARIE	CORBA
RSCA	CORBA
Orca	Ice
ASEBA	Ad-hoc approach
AWARE	Publish/Subscribe
PEIS	Shard memory

[표 1]에서 보이는 바와 같이 지금까지 개발된 로봇 미들웨어 플랫폼은 효율성, 확장성, 안정성 등의 확실한 테스트를 거치지 않은 채 국제 표준 단체인 OMG가 표준으로 제시하고 있는 CORBA를 채택하여 사용하고 있다. 하지만 CORBA는 다양한 응용환경에의 적용을 고려하여 범용 분산 통신 플랫폼 구조를 취하기 때문에 로봇산업 분야에 사용되기에는 적합하지 않다. 또한, 04년 이후 Minimum CORBA, Real-time CORBA를 발표하였지만, 분산 미들웨어 플랫폼이 가지고 있는 고질적인 통합문제와 통신 문제점을 해결하지 못하고 있다. 이러한 문제점은 로봇 미들웨어 플랫폼에서 통신계층을 구성하는데 있어서 통신 모델, 상호작용 패턴, 그리고 컴포넌트 모델을 고려하지 않아 발생한다[4].

본 논문의 구성은 다음과 같다. 2장에서는, 기존 로봇 플랫폼의 통신모델과 상호작용 패턴을 기술하고, 3장에서 각 통신 모델의 성능을 측정한다. 마지막으로 결론을 맺고 향후 로봇 미들웨어 플랫폼의 연구 과제를 제시한다.

2. 로보틱스 미들웨어 플랫폼

로봇의 기능이 점점 복잡해짐에 따라 개발자를 위한 편리한 개발환경, 표준화된 부품, 로봇 컴포넌트 간의 통신기능 등이 매우 중요해졌다. 따라서 소프트웨어의 재사용성, 컴포넌트간의 안정성, 개발의 편의성 그리고 운영체제 상에서의 투명성을 지원하기 위해 컴포넌트 소프트웨어 공학 방법을 적용하였다. 하지만, 대부분의 컴포넌트 방법은 컴포넌트와 인터페이스를 구현 및 정의 하는데 있어 엄격한 절차 혹은 명세를 제공을 하지 않기 때문에 개발된 컴포넌트를 독립적으로 수정하는 것을 보장하지 못하며, 특정 도메인에 사용되는 컴포넌트의 재사용성과 안정성을 제공하는데 커다란 벽(Composition Gap)이 존재한다. 이러한 특성들은 통신구조의 상속 복잡성과 유연성으로부터 파생된다[5].

2.1. 통신모델

현재 개발된 로봇 미들웨어 플랫폼의 통신 모델은 [표 1]의 결과로부터 RPC, RMI, Publish/Subscribe, AMI, Ad-hoc approach 등으로 나뉘어 진다.

RPC(Remote Procedure Call)는 클라이언트/서버 기술을 기반으로 하며 지역 프로시저 호출과 같은 방법으로 동작이 이루어진다. 단지 실행되는 프로시저들이 지역적으로 분산되어 있다는 차이점만을 가진다. 즉, 호출된 프로시저는 네트워크에 전달이 되며, 사용자는 네트워크를 통한 RPC의 처리과정을 알 필요 없이 지역 프로시저 호출을 하는 것처럼 RPC를 이용하여 분산 시스템을 설계한다. 메시지 교환을 이용하여 통신상의 과부하가 적고 단순성, 투명성 때문에 분산 응용 설계 시 많이 이용되어진다. 이러한 장점들로 초반 로봇 미들웨어 플랫폼은 RPC를 기반 통신을 사용했다[6].

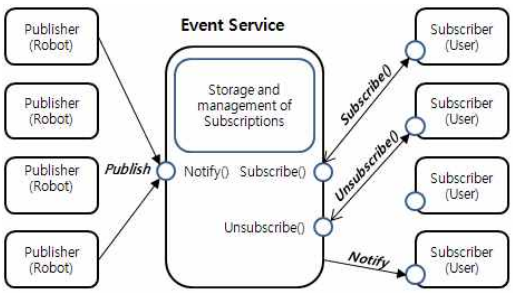
하지만, RPC는 복잡하고 다양한 기능의 최신 프로그램을 개발하거나, 이미 개발된 메커니즘을 적용하는데 있어 미흡한 점이 많고 다른 RPC간의 호환성이 결여되어 유지보수가 어려운 단점을 가지고 있으며, 클라이언트가 요청한 서비스를 서버에서 프로세싱 작업을 하기 때문에 서버의 자원을 낭비하게 되며, 서버의 작업이 종료될 때까지 클라이언트는 대기상태로 존재한다. 이를 해결하기 위해 비동기 RPC를 연구하였지만, 서버의 과부하 문제를 해결하지 못한다.

RMI(Remote Method Invocation)는 RPC의 객체처리의 복잡성과 관리의 어려움을 해결하기 위해 객체 그룹을 이용하는 클라이언트/서버 기반의 통신 모델이다. 서로 다른 런타임 환경에서 클라이언트가 원격의 객체를 로컬 객체처럼 자신의 런타임 환경에서 원하는 메소드를 호출할 수 있도록 하는 자바의 분산 객체 표준이다. 하지만, RMI는 클라이

언트 및 서버가 모두 자바 기반이여야 하며, *Method Invocation*으로 인하여 성능이 매우 떨어진다.

Publish/Subscribe는 이벤트(Topic) 생산자와 소비자의 중간에 이벤트 브로커를 둠으로써 생산자와 소비자 간 의존성을 없앴다. 따라서, 이벤트 생산자와 소비자는 동시에 실행될 필요도, 서로의 위치를 인지하고 있을 필요도 없게 되어 환경적 변경이 잦은 자연발생적 상호작용을 쉽게 지원할 수 있는 장점을 지니고 있다[7][8]. 시스템 통합 기술이 분산 컴퓨팅 환경으로 변화하고, 다양한 서비스를 제공하는 지능형 로봇이 등장함에 따라서, 사용자가 로봇의 다양한 서비스를 요구하고, 유비쿼터스 기술의 발전으로 주변 센서와의 상호운영성이 증대함에 따라서, 기존의 RPC, RMI와 같은 기술은 로봇 플랫폼의 통신 모델로 적용하기에 부적합하다. 이와 같은 문제점을 해결하기 위해서 로봇 미들웨어 플랫폼은 Publish/Subscribe 통신 모델을 채택하고 있다[그림 2].

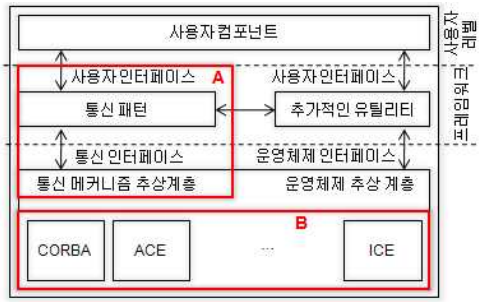
대표적인 분산 미들웨어 플랫폼 중의 하나인 CORBA는 AMI(Asynchronous Method Invocation), Publish/Subscribe 통신 모델을 지원하며, ICE (Internet Communication Engine) 통신 플랫폼 역시 RPC와 Publish/Subscribe 모델을 지원한다. 이러한 대표적인 통신 모델의 성능을 측정하기 위해 RPC와 RMI의 성능을 비교하며, CORBA 기반 그리고 ICE 기반 Publish/Subscribe 모델 성능을 3장에서 기술한다.



[그림 2] 로봇 미들웨어 플랫폼의 Pub/sub 통신 모델

2.2 상호작용 패턴

본 장에서는 로봇 미들웨어 플랫폼을 구성할 때 반드시 고려해야할 통신 패턴에 대한 중요성과 특징을 설명한다. [그림 3]에서 나타나는 바와 같이 통신 패턴은 통신 메커니즘 과 사용자 인터페이스의 중간에 위치하며, 서비스 요청자와 서비스 제공자를 분리하고, 수신 받은 요청 또는 결과를 처리하는데 있어 다양한 사용자 접근방식(동기방식, 비동기방식)을 제공한다. 이러한 상호작용 패턴은 크게 Client pattern과 Server patterns로 나뉘며 이들의 조합을 고려하여 상호작용 패턴을 구성하여야 한다.

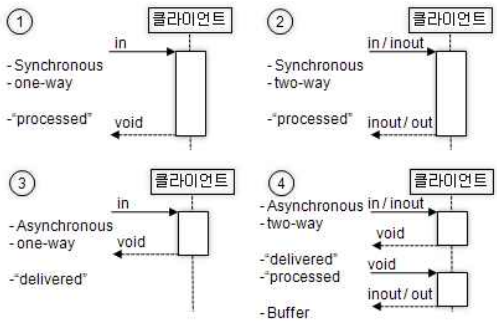


[그림 3] 로봇 플랫폼 내부 계층 구조

Client patterns

상호작용 패턴의 클라이언트 부분의 특성은 [그림 4]에 자세히 나타나 있다. 클라이언트 인터페이스들은 항상 멤버 함수 호출을 기반으로 하기 때문에, 통신 패턴의 접근 모드와 *direction, invocation*모드로 분류를 할 수 있다.

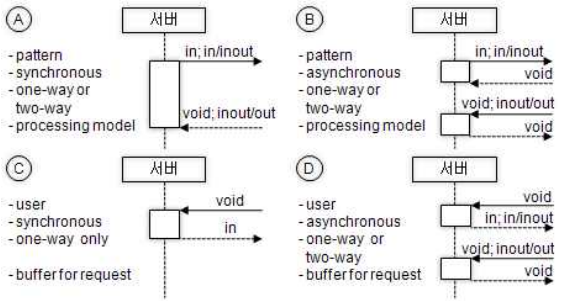
Synchronous one-way 특성①은 오로지 *in*매개변수를 가지며, 서버의 작업 처리가 끝날 때까지 블록되며, 서버의 작업의 처리가 끝나는 순간 *processed* 통지를 클라이언트에게 보내준다. 이 방법은 매개변수를 리턴 하지 않으며, Empty(void)를 리턴 한다. Synchronous two-way 특성②은 *in, inout*매개변수를 가지며, 서버의 작업 처리가 끝날 때까지 블록된다. ①의 방식과 차이점은 리턴 메시지가 Empty(void)가 아니며, *inout*과 *out*매개변수를 리턴 한다. Asynchronous one-way 특성③은 *in*매개변수를 가지며, 서버의 작업 처리와 상관없이 클라이언트는 블록 되지 않는다. 하지만, *unreliable send*정책으로 인하여 클라이언트는 서버의 반환 메시지를 잃어버릴 가능성이 존재한다. Asynchronous two-way 특성④은 요청 과 응답 메소드를 사용하기 때문에 클라이언트가 서버의 요청에 대한 결과를 받았을 때 이를 확인 하고 응답 메소드를 호출하기 위해 임시적으로 저장해야할 버퍼가 필요하다.



[그림 4] 상호작용 패턴의 클라이언트 부분 특성

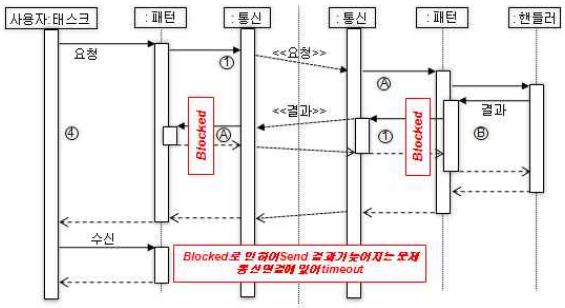
Server patterns

서버 패턴은 크게 *initiation*과 *invocation*모드로 나뉘어 진다. *initiation*은 *user*와 *pattern*으로, *invocation*은 동기와 비동기 모드로 세분화 된다. [그림 5]은 이러한 모드들의 조합에 따른 서버의 특성 유형들이다. Ⓐ는 상호작용 패턴으로부터 처리하는 것을 실행하고, *single upcall*과 함께 클라이언트 요청을 처리 한다. 이 클라이언트 요청은 *upcall*로부터 반환되는 즉시, 완료되며 클라이언트로 결과를 반환한다. Ⓑ는 클라이언트 요청을 위한 *upcall*을 실행하며, 해당 실행이 끝나는 즉시, 응답을 위한 *downcall*을 호출한다.



[그림 5] 상호작용 패턴의 서버 부분 특성

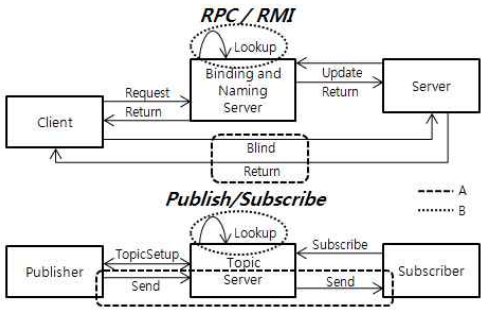
대부분의 로봇 미들웨어 플랫폼은 다음과 같은 조합으로 이루어져 있다. RPC 모델은 (②/Ⓐ)의 패턴으로, CORBA 모델은 (③/Ⓐ)와 (④/Ⓐ)로, 메일박스 시스템과 TCP소켓은 (③/Ⓒ)와 같은 패턴으로 구성되어 있다. 이러한 상호 작용 패턴들의 중요한 점은 각각의 Server pattern과 Client pattern 조합들을 고려하여야 한다. 각각의 조합 패턴들은 통신 패턴에 있어서 예기치 않은 오류 혹은 *block*으로 인한 *timeout* 현상을 파생할 수 있기 때문이다. [그림 6]은 통신 모델에서 발생할 수 있는 예기치 않은 오류에 대한 문제점에 대해서 간략하게 소개하고 있다.



[그림 6] 통신 패턴 조합으로 인한 오류

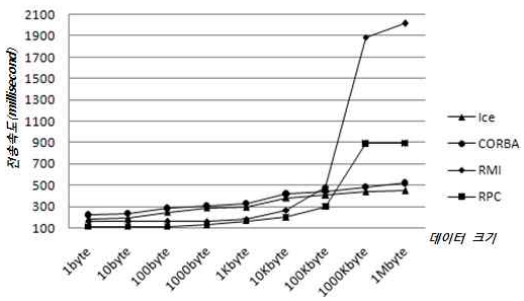
3. 성능측정

RPC, RMI, Publish/Subscribe(CORBA, Ice)의 성능 측정을 하기 위해 RPC와 RMI, CORBA와 Ice로 나눈다. RPC와 RMI는 최근에 분산컴퓨팅 환경에 적용하기 위하여 RPC는 Binding Server, RMI는 Naming Server를 두고 있으며, Publish/Subscribe(CORBA, Ice) 역시 Topic을 관리하는 Topic Server를 따로 둔다. 하지만, [그림 7]에서 'A'와 같이 통신 과정이 다르다.

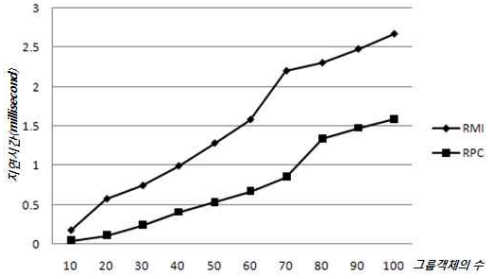


[그림 7] 통신 모델 루틴의 차이점

또한, RPC와 'B'와 같이 RMI는 객체그룹기술을 적용하고 있어 객체 수의 증/감량을 가지고 성능을 분석할 수 있지만, Publish/Subscribe모델은 각각의 객체가 여러 가지의 Topic을 요구할 수 있기 때문에, 객체 수의 증/감량으로 성능을 측정하기 불분명하다. 따라서, 성능 측정을 3단계로 나누어 성능을 측정한다. 간단한 작업을 하는 *Object*의 메시지의 크기를 증가시키고 이에 따른 처리 지연시간으로 각 통신 모델의 성능을 측정을 하고, RPC와 RMI의 성능은 그룹간의 객체의 수의 증가에 따른 지연시간으로 성능을 측정한다. 마지막으로, Publish/Subscribe 모델을 적용한 ICE와 CORBA와 *ad-hoc mechanism*의 성능은 Tpoic의 수를 증가시키면서 발생하는 처리 속도를 기반으로 성능을 측정한다.



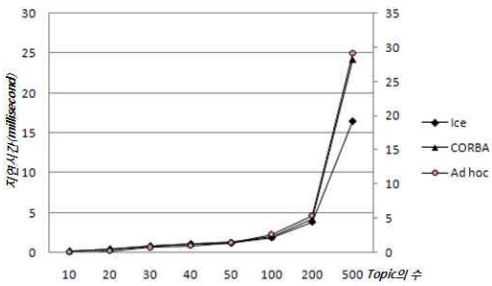
[그림 8] 각 통신 모델의 성능분석



[그림 9] RMI와 RPC의 성능 분석

[그림 8]은 데이터 크기를 증가시킴에 따른 각 통신 모델의 전송속도의 결과 그래프이다. CORBA와 Ice는 전송되는 데이터의 크기를 관리해주는 *ObjectManagement*가 존재하여, Service를 보다 안정적으로 제공하며, 단위시간 당 데이터 크기를 증가하였을 때 전송되는데 소요되는 시간이 순차적으로 증가하는 것을 볼 수 있다. 이와 달리, RMI와 RPC는 이러한 데이터의 크기를 관리해주는 interface가 기본적으로 있지 않기 때문에 초반에는 통신 성능이 좋지만, 100Kbyte를 기준으로 급격하게 증가하는 것을 볼 수 있다. [그림 9]의 결과와 같이 RMI는 기본적으로 데이터 마셜링 부분에서 많은 시간을 소요한다. 따라서, RMI는 RPC보다 좋지 않은 결과를 낸다.

[그림 10]는 Publish/Subscribe모델은 CORBA, ICE 그리고 임시적방법론을 적용한 통신의 성능을 Topic 수의 증가량에 따른 처리 지연시간을 측정한 결과이다. 실험 결과에 따르면, Topic의 수가 적을 경우 CORBA와 Ice의 성능은 크게 차이가 나지 않는다. 하지만, 100개(Topic)를 기점으로 CORBA의 성능은 Ice에 비해 현저하게 떨어진다. 이는 CORBA의 Object 식별방식에서 *Reference Pointer*의 오류에 의해 문제가 발생한다[9]. 물론 Ice역시 *Object model*을 사용하고 있는 문제점을 가지고 있다.



[그림 10] CORBA, Ice, Ad-hoc mechanism의 성능 분석

4. 결론

본 논문에서는 현재까지 개발된 로봇 미들웨어 플랫폼을 통신 모델과 패턴을 기반으로 성능을 측정하고, 로봇산업의 미들웨어 플랫폼을 설계할 때, 반드시 통신 모델과 패턴을 고려해야 하는 것을 실험을 통해 수 있었다.

IT(Information Technology) 기술을 바탕으로 다양한 서비스를 제공하는 차세대 지능형 로봇에 대한 관심이 고조되고 있는 현 시점에서, 지능형 로봇을 위한 플랫폼을 설계 시, 보다 엄격한 컴포넌트 개발 방법론을 제시해야하며, 통신 계층 부분에서도, 명확한 통신 모델을 채택하고, 통신 패턴을 분석하여, 예기치 않은 오류나 성능을 현저하게 떨어뜨릴 수 있는 문제에 대하여 대처해야 할 것이다. 이와 더불어 아직까지 CCM모델의 문제점이 제기되고 있는 가운데, 국내 컴포넌트 모델의 표준의 개발이 시급하다.

참고문헌

[1] Toshio HORI, HIRUKAWA, Takashi SUEHIRO, and Shigeoki HIRAI, "*Networked Robots as Distributed Object*", IEEE International Conference on Advanced Itelligent Mechatronics, September, 1999, pp. 61-66.

[2] Chirstopher D. Gill and William D. Smart, "*Middleware for Robots?*", In proceedings of the AAAI Spring Symposium on Intelligent Distributed and Embedded Systems, 2002.

[3] Nader Mohamed. Hameela A1-Jaroodi, and Imad Tawhar, "*Middleware for Robotics: A Survey*", In Processing of the IEEE International Conference on Robotics, Automation and Mechatronics, September, pp. 736-742, 2008.

[4] N. Wang, D. Schmidt, and C. O'Ryan, "*Overview of the CORBA component model*", In Component-based software engineering : putting the pieces together,, chapter 31, pages 557-571, Addison-Wesley, 2001.

[5] Brugali, D. and Agah, A and MacDonald, B. and Nesnas, I. and Smart W. "*Trends in component-based Robotics*", In D.Brugali(Ed.) Software Engineering for Experimental Robotics. Spinger STAR series, 2006.

[6] Andrew D, Birrell and Bruce Jay Nelson, "*Implementing Remote Procedure Calls*", ACM Transactions on Computer Systems, Vol. 2, No. 1, February 1984.

[7] Partrick TH. Eugster, Pascal A. Felber, Rachid Guerraoui and Anne-Mari Kermarrec, "*The many Faces of Publish/Subscribe*", ACM Computing Syrveys, Vol. 35, No. 2, June 2003.

[8] Matteo Matteucci, "*Publish/Subscribe Middleware for Robotics: Requirements and State of the Art*", Tech. Report 2003.3, Politecnico 야 Milano, Milan, Italy 2003.

[9] Michi Henning, "A New Approach to Object-Oriented Middleware", IEEE Internet Comuting, Vol. 8, No. 1, pp. 66-75, 2004.