# 메시지 기반 인터페이스 공동 개발을 위한 메시지 관리 시스템

## (A Message Management System for Cooperative Message-based Interface Development)

유 제 영 † 박 진 희 ††
(Je-young Yu) (Jinhee Park)

**요 약** 대형 시스템은 각 컴포넌트가 여러 다른 개발자에 의하여 개발된다. 개발 과정에서 컴포넌트 간의 인터페이스 관리의 효율성은 전체 개발 효율에 큰 영향을 미친다. 특히, 개발 초기 과정에서는 컴포넌트가 새롭게 정의되거나 변경되는 경우가 많다. 컴포넌트의 새로운 정의나 변경은 컴포넌트 간의 인터페이스의 생성이나 변경을 필요로 한다. 이러한 인터페이스의 변경에 대한 관리가 효율적으로 이루어지지 않는 경우, 관련된 컴포넌트를 개발하는 서로 다른 개발자들이 서로 다른 인터페이스를 기반으로 개발을 진행하는 경우가 발생한다. 이는 개발의 효율이 저하되는 문제를 일으킨다. 이러한 문제를 해결하기 위하여 메시지 관리 시스템은 메시지 기반의 인터페이스 관리를 위한 수단을 제공한다. 또한, 메시지에 대한 코드 생성을 통하여 구현의 편의성을 제공해 주며, 메시지 관리 시스템 내에 정의된 메시지 정보를 이용하여 인터페이스 설계 명세서(IDD)를 자동으로 생성해 준다. 문서 자동 생성은 메시지 변경으로 인한 실제 인터페이스와 문서와의 불일치를 방지한다. 또한, 메시지 관리 시스템은 전체 시스템의 각 컴포넌트 간 메시지를 관리함으로써 각 컴포넌트 간 메시지 흐름에 대한 종합적인 정보를 수집하는 것이 가능하다. 이러한 정보는 컴포넌트 간 메시지 흐름의 병목 현상의 발생을 미리 방지하거나, 전체 시스템 성능의 조율을 지원하는데 응용할 수도 있다.

· 이 논문은 제34회 추계학술대회에서 '메시지 기반 인터페이스 공동 개발을 위한 메시지 관리 시스템'의 제목으로 발표된 논문을 확장한 것임

† 정 회 원 : 삼성탈레스 SW2 그룹 선임연구원
jeyoung.yu@samsung.com
†† 비 회 원 : 삼성탈레스 SW2 그룹 연구원
jh91.park@samsung.com

논문접수 : 2007년 12월 14일
심사완료 : 2008년 5월 28일



정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제6호(2008.8)

키워드 : 메시지 관리, 인터페이스 관리, 메시지 기반 인터페이스

***Abstract*** In Large scale system, components are developed by many different developers. In such a development environment, efficiency of the development depends largely on effectiveness of interface management. In the early stage of development, many components are newly defined or modified quite often. These definitions and modifications of components cause the change of interfaces between components. If changes of interfaces are not properly managed, many developers may implement components based on different version of interfaces. This causes decrease in efficiency of development. "Message Definition and Management System (MDMS)" provides the means to cope with the inefficiency of unmanaged interface definitions and changes. MDMS automatically generates source code and Interface Design Description (IDD). The automatic generation of source code and IDD prevents the disagreement between code and documents. Furthermore, MDMS shows the overall view of message flow for a system. Based on this information, we can optimize the system identifying message bottleneck and apply to support for the performance tuning of the system.

**Key words** : Message Management, Interface Management, Message-based Interface

## 1. 서 론

대형 시스템은 일반적으로 수십 명 이상의 개발자가 협력하여 개발한다. 하나의 대형 시스템은 수많은 컴포넌트로 구성되며, 그 컴포넌트의 수는 시스템의 구성에 따라 수십 개에서 수천 개에 이르기도 한다. 각 컴포넌트는 전체 시스템의 기능을 수행하기 위하여 서로 명령이나 데이타를 주고 받는다. 이렇게 각 컴포넌트가 제공하는 서비스나 데이타의 형식과 내용을 정의한 것이 컴포넌트의 인터페이스이다.

대형 시스템은 각 컴포넌트가 여러 다른 개발자에 의하여 개발된다. 따라서, 개발 과정에서 이러한 컴포넌트 간의 인터페이스 관리의 효율성은 전체 개발 효율에 큰 영향을 미친다. 특히, 개발 초기 과정에는 컴포넌트가 새롭게 생성되거나 변경되는 일이 빈번하게 발생한다. 이에 따라 메시지 기반의 인터페이스를 사용하는 시스템에서는 컴포넌트 간의 인터페이스 역할을 하는 메시지가 새로 생성되거나, 제거되거나, 변경되는 일이 발생한다. 이러한 변경에 대한 관리가 효율적으로 이루어지지 않는 경우, 관련된 컴포넌트를 개발하는 서로 다른 개발자들이 서로 다른 인터페이스를 기반으로 개발을 진행하는 경우가 발생할 수 있다. 이는 바로 개발 및 시험에 직접

적인 영향을 미치며 공동작업의 효율을 떨어뜨리는 직접적인 원인이 된다. 또한, 컴포넌트 간 인터페이스의 정의 및 변경은 많은 문서 및 코드의 변경을 동반한다.

우리는 이러한 문제를 해결하기 위하여 메시지 관리 시스템을 개발하였다. 메시지 관리 시스템은 메시지 기반의 인터페이스 관리를 위한 수단을 제공한다. 또한, 메시지 구조와 해당 메시지를 처리하기 위한 기본적인 소스 코드를 자동 생성으로 줌으로써 개발의 편의성을 제공한다. 메시지 관리 시스템은 관리하는 메시지 정보를 이용하여 인터페이스 설계 명세서(Interface Design Description: IDD)를 자동으로 생성해 준다. 이는 문서 작성 작업을 간단하게 만들어 주며, 메시지 변경으로 인한 실제 인터페이스와 문서와의 불일치를 방지한다.

메시지 관리 시스템은 전체 시스템의 각 컴포넌트 간 메시지를 관리함으로써 각 컴포넌트 간 메시지 흐름에 대한 종합적인 정보를 수집하는 것이 가능하다. 이러한 정보를 바탕으로 컴포넌트 간 메시지 흐름의 병목 현상의 발생을 미리 방지하거나, 전체 시스템 성능의 조율을 지원하기 위하여 응용할 수 있다.

## 2. 관련 연구

최근에 가장 널리 사용되는 통신 미들웨어는 크게 객체 지향 미들웨어(OOM: Object Oriented Middleware)와 메시지 지향 미들웨어(MOM: Message Oriented Middleware)로 분류할 수 있다.

객체 지향 미들웨어에서 컴포넌트의 인터페이스는 컴포넌트가 정의하는 일련의 속성(Attributes)과 동작(Methods)으로 이루어진다. 객체 지향 미들웨어를 사용하는 경우 컴포넌트간의 통신은 다른 컴포넌트의 속성을 변경하거나, 동작을 실행함으로써 이루어진다. 객체 지향 미들웨어의 대표적인 예로는 CORBA[1]와 DCOM[2]이 있다. 메시지 지향 미들웨어(MOM: Message Oriented Middle-ware)는 컴포넌트의 인터페이스가 컴포넌트간 메시지로 이루어진다. 메시지 지향 미들웨어를 사용하는 경우 컴포넌트간 통신은 메시지의 전달로 이루어진다. 대표적인 메시지 지향 미들웨어는 DDS[3]가 있다.

객체 지향 미들웨어의 경우 컴포넌트 간의 인터페이스 정의는 다른 컴포넌트가 호출하여 사용할 수 있는 속성(Attributes)과 동작(Methods)을 정의하는 것이다. 컴포넌트의 속성과 동작의 정의를 위하여 일반적으로 널리 사용하고 있는 객체 지향 모델링 툴들을 적용할 수 있다. 특히 Rational Rose, Telelogic Rhapsody와 같은 도구들은 자동 코드 생성과 자동 문서 생성 등의 다양한 기능을 제공하고 있다.

메시지 기반의 미들웨어의 경우 컴포넌트간의 인터페이스 정의는 다른 컴포넌트로 전달하기 위한 메시지의 구조 및 메시지의 성격을 정의하는 것이다. 메시지 기반의 미들웨어 상의 시스템은 각 기능을 담당하는 컴포넌트들과 컴포넌트들 사이의 메시지들로 표현이 가능하다. 이러한 표현은 DFD(Data Flow Diagram)의 표현과 일치한다. 즉 메시지를 데이타로 표현하고 컴포넌트를 기능으로 표현한 후 DFD를 적용하면 메시지 기반의 미들웨어 상의 시스템이 표현된다. DFD 기반의 모델링 도구로는 Cadre TeamWork이 있다. Cadre TeamWork은 DFD 기반의 CASE 도구로써 DFD 기반의 모델링을 지원하며, Function 간의 메시지의 형식 및 성격을 정의할 수 있는 기능을 제공한다. 이렇게 정의한 내용은 코드나 문서로 자동으로 생성된다. 하지만, TeamWork은 2000년도 이후 단종되어 더 이상 사용이 불가능하다.

기존의 인터페이스를 정의하기 위한 도구는 모델링 툴의 부수적인 기능으로 존재한다. 따라서 인터페이스 정의 기능은 존재하지만, 인터페이스 변경에 대한 프로세스나, 인터페이스의 형상관리 기능 등은 제공되지 않는다. 인터페이스 변경 프로세스나 형상관리 기능은 널리 사용되는 형상관리 시스템과 모델링 도구를 결합하는 방법이 있다. 하지만, 기존의 형상관리 시스템은 소스코드나 문서에 대한 형상관리를 위하여 개발되었다. 따라서, 인터페이스나 모델링 정보에 대한 형상 관리를 위해서는 인터페이스 정의나, 모델링 정보를 소스 코드나 문서로 변환한 후, 관리 시스템을 이용하여 간접적으로 관리하는 방법을 이용한다.

간접적인 인터페이스 형상관리는 효율성이 떨어지며, 인터페이스 변경 절차의 부재로 관리에 많은 노력이 요구된다. 또한, 간접적인 관리의 경우 정확하게 이루어지지 않으면, 관리가 전혀 이루어지지 않는 것과 동일한 결과를 초래한다.

## 3. 설계 고려 사항

메시지 관리 시스템의 설계 시 가장 중요한 고려사항은 메시지 관리 프로세스의 수립이다. 메시지는 컴포넌트 간의 인터페이스로써 다수의 컴포넌트와 연관되어 있다. 따라서 메시지의 정의 프로세스와, 변경 프로세스가 명확해야 시스템의 효율성을 보장 받을 수 있다.

메시지 관리 시스템은 메시지 관리 프로세스의 수립을 위하여 OMG에 의하여 표준으로 채택되어 최근에 많이 사용되고 있는 출판-구독(Publish-Subscribe) 기반의 메시지 기반 통신 미들웨어인 DDS를 사용하는 것으로 가정한다.

출판-구독 기반의 통신 미들웨어는 메시지 기반의 인터페이스를 사용하여 어떠한 컴포넌트가 특정한 정보나, 서비스에 대한 요청을 메시지로 네트워크에 출판(Publish)하고, 메시지를 구독(Subscribe)하는 컴포넌트가 정보를 사용하거나, 요청을 처리하는 방식으로 통신이 이루어진다. 출판-구독 기반의 통신의 가장 큰 특징은 출

판하는 컴포넌트는 누가 구독하는지 그리고 얼마나 많은 컴포넌트가 구독하는지에 대하여 미리 알 필요가 없고, 구독하는 컴포넌트 역시 구독하는 정보를 누가 출판하는지에 얼마나 많은 컴포넌트가 해당 정보를 출판하는지를 미리 알 필요가 없다는 것이다. 이것은 메시지 통신을 단순화함으로써 대형 시스템 설계를 단순화하고, 새로운 컴포넌트의 추가에 의한 기존 시스템의 영향을 최소화함으로써 확장성(Scalability)를 극대화 할 수 있다. DDS에서는 메시지의 정의를 OMG Interface Definition Language(IDL)[1]을 이용하여 정의한다.

그림 1에서 그림 3은 각각 메시지 정의 절차와 메시지 변경 절차, 메시지 삭제 절차를 나타낸다. 회색으로 표시된 단계에서는 히스토리 정보가 별도로 저장되어 메시지의 변경 히스토리를 참조할 수 있게 한다. 각 절차에서 메시지는 그림 4와 같이 *Wait, Confirm, Fix*의 세가지 상태 중 하나의 상태를 가지게 된다.

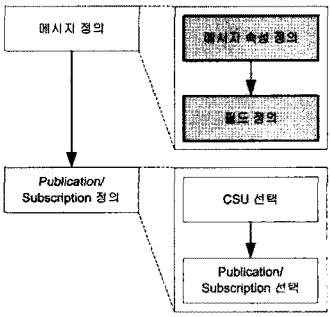


그림 1 메시지 정의 절차

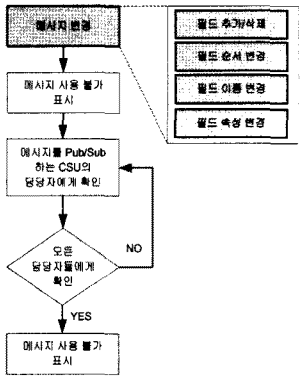


그림 2 메시지 변경 절차

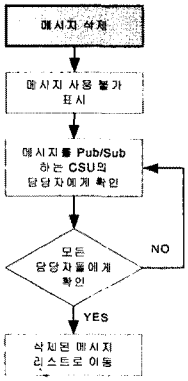


그림 3 메시지 삭제 절차

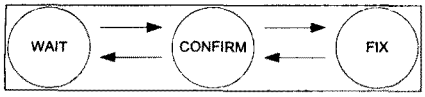


그림 4 메시지 상태 변경

*Wait* 상태는 메시지가 누군가에 의하여 수정이 발생한 상태이다. *Wait* 상태의 메시지의 경우 수정이나, IDD 생성, IDL의 생성 등의 작업이 불가능하다. 메시지를 수정이 가능한 상태로 다시 만들려면 관련된 모든 컴포넌트 담당자들에게 확인을 받거나 관리자에게 확인을 받아 *Confirm* 상태로 변경해야 한다.

메시지가 처음 생성되거나, 메시지의 변경 후 모든 관련 컴포넌트 담당자, 또는 관리자에게 확인을 받으면 *Confirm* 상태가 된다. *Confirm* 상태의 메시지의 경우, 메시지의 변경, IDD 생성, IDL 생성 등의 모든 작업이 가능하다.

개발 프로세스상 베이스라인이 설정 등의 이유로 메시지에 대한 더 이상의 변경이 불필요하거나 변경을 금지 시키기를 원하는 경우, 메시지를 *Fix* 상태로 변경할 수 있다. *Fix* 상태의 메시지는 IDD 생성, IDL 생성 등의 작업은 가능하지만, 메시지의 수정은 불가능하다. 메시지를 *Fix* 상태로 변경하거나, *Fix* 상태의 메시지를 *Confirm* 상태로 변경하는 것은 프로젝트 관리자 만이 가능하다. 이와 같은 프로세스를 제공하기 위해서는 추가로 사용자 관리와 컴포넌트 이름 및 컴포넌트 담당자 관리 등의 기능이 필요하다.

메시지 관리 프로세스 외에 중요한 고려사항으로는 접근성과 공동 개발 지원이다. 메시지 관리 시스템은 효과적인 접근성 및 공동 개발 지원을 위하여 웹 어플리케이션으로 개발한다. 또한 앞에서 이미 언급한 히스토리 정보의 저장이나, 메시지 변경 관리 기능 역시 공동 개발 지원을 위하여 필수적인 기능이다.

## 4. 구 현

개발의 효율성을 높이기 위하여 메시지 관리 시스템은 Ruby 언어와 Rails 웹 프레임워크[4]를 사용하여 개발하였다. Rails 웹 프레임워크는 전형적인 MVC(Model-View-Controller) 아키텍처로 구성되어 있다.

그림 5와 같이 메시지 관리 시스템(Message Definition & Management System - MDMS)은 데이타베이스에 대한 ORM(Object-Relation Mapping) 레이어로 구성된 모델과 사용자 인터페이스를 제공하는 뷰, 그리고 뷰에서 입력된 정보와 사용자 명령에 따라 모델을 처리하고, 처리된 결과에 따라 뷰를 구성하는 컨트롤러로 구성되어 있다.

모델은 MDMS의 데이타베이스에 대한 ORM 레이어를 제공한다. ORM 레이어에 의하여 각 테이블은 하나의 클래스로 나타나며, 테이블의 레코드는 각 클래스의 하나의 객체로써 표현된다.

컨트롤러는 대표적인 5개의 컨트롤러 클래스로 구성되며, 각각은 다음과 같은 기능을 수행한다.

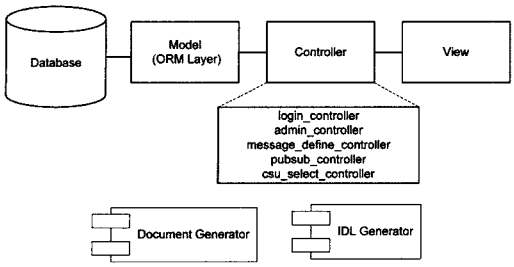


그림 5 MDMS 소프트웨어 구조

- login_controller: 사용자에 대한 인증 및 세션 관리
- admin_controller: 사용자 관리, 프로젝트 관리
- message_define_controller
  - 메시지 정의, 변경, 삭제
  - Publication 및 Subscription 관리
  - 변경 내역 정보 기록, 참조
  - Document Generator와 IDL Generator를 이용 IDD 문서와 IDL 파일 생성
  - View 제어
- pubsub_controller
  - publication 정의 및 변경
  - subscription 정의 및 변경
- csu_select_controller
  - 컴포넌트의 정의 및 선택

MDMS의 뷰는 그림 6과 같이 구성된다. 각 상자는 대표적인 웹 페이지를 나타낸다.

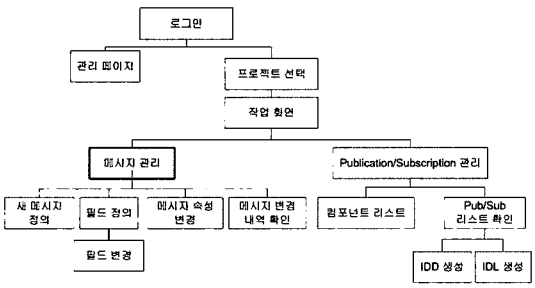


그림 6 MDMS 뷰 구성

사용자는 그림 7의 로그인 페이지를 통하여 MDMS를 사용할 수 있다. 사용자가 MDMS 관리자의 아이디로 로그인을 할 경우, 그림 8의 관리 페이지로 접근하여 사용자, 프로젝트 등을 관리할 수 있다. 일반 사용자의 경우 로그인 시 그림 9의 프로젝트 선택 페이지가 나타나며, 일반 사용자가 속해있는 프로젝트의 리스트에서 원하는 프로젝트를 선택할 수 있다.

프로젝트를 선택하면 그림 10과 같은 작업 화면이 나타난다. 작업화면의 좌측은 Publication/Subscription과

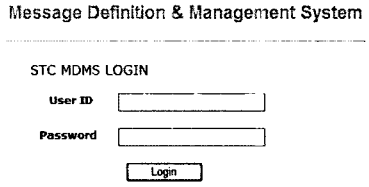


그림 7 MDMS 로그인 화면

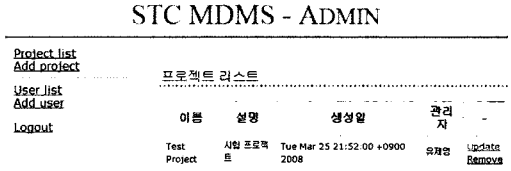


그림 8 MDMS 관리자 화면

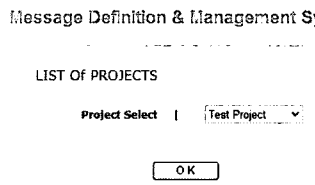


그림 9 프로젝트 선택 화면

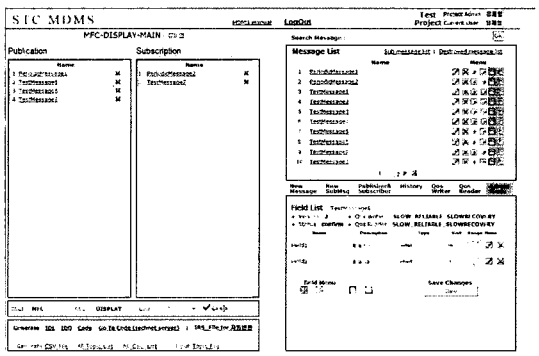


그림 10 MDMS 작업 화면

컴포넌트 생성/선택, 각 컴포넌트에 대한 IDL 파일/IDD 문서 자동 생성 등의 기능을 선택할 수 있도록 구성되어 있다. 작업 화면 우측에서는 각 메시지에 대한 작업을 수행할 수 있다. 메시지에 대한 작업은 메시지 리스트 보기, 메시지의 생성 및 수정, 메시지 필드의 정의, 각 메시지의 변경 이력 보기 등을 포함한다.

그림 11은 소스 코드 생성의 예로써 IDL 파일 자동 생성의 결과를 보여준다. 그림 12는 MDMS가 자동으로 생성한 IDD 문서의 예이다. 코드나 문서 자동 생성은 필요에 따라 다양한 형식을 지원할 수 있다.

```
//-----------------------------------------------------
// Message definition for MFC_DISPLAY_MAIN by IDL
//
// This file is generated automatically by STC MDMS
// copyright (c) 2007 Samsung Thales Co., Inc.
// All rights reserved
//-----------------------------------------------------
//*****************************************************
// Publication messages for MFC_DISPLAY_MAIN
//*****************************************************
//-----------------------------------------------------
// 주기적 메시지 1
//-----------------------------------------------------
struct PeriodicMessage1 {
        octet Field1;
        short Field2;
};

..... (생략)

//*****************************************************
// Subscription messages for MAIN
//*****************************************************
//-----------------------------------------------------
// 주기적 메시지 2
//-----------------------------------------------------
struct PeriodicMessage2 {
        octet Field1;
        short Field2;
};

..... (생략)
```

그림 11 IDL 자동 생성 예

**Publications Message List**

**TestMessage5**

시험용 메시지 5

| Field Name | Field Type | Description |
|---|---|---|
| Field1 | octet | 시험 필드 1 |
| Field2 | short | 시험 필드2 |
| NewField | float | 새로운 필드 추가 |

**Subscription Message List**

**PeriodicMessage2**

주기적 메시지 2

| Field Name | Field Type | Description |
|---|---|---|
| TestField1 | char | 테스트 필드입니다. |

그림 12 IDD 자동 생성 예

## 4. 결론 및 향후 연구

많은 개발자가 참여하는 대형 시스템의 개발 과정 상의 효율성 증대를 위해서는 컴포넌트 간 인터페이스의 효과적인 관리가 필수적이다. 메시지 관리 시스템은 컴포넌트간의 인터페이스의 정의 및 변경 이력 관리, 인터페이스 기술서 및 소스 코드 자동 생성 등의 기능을 간결하고 명확한 절차를 통하여 제공한다.

본 논문에서 제시한 메시지 관리 시스템은 개발이 완료되어 사업에 적용하여 개발의 효율성이 증대되었다. 메시지 관리 시스템의 적용으로 얻은 장점은 다음과 같다.

- 컴포넌트 간의 인터페이스 정의 절차 단순화
- 컴포넌트 간의 인터페이스 변경 절차 단순화
- 인터페이스 변경 내역 추적
- 서로 다른 인터페이스 버전의 사용으로 인한 시간과 노력의 낭비 방지
- 인터페이스 기술서 자동 생성으로 문서화 단순화
- 컴포넌트간 인터페이스 관련 코드 자동 생성으로 표준화된 인터페이스 코드 사용 지원

현재까지 개발된 MDMS는 표준화된 인터페이스 처리 코드 지원과, 컴포넌트간 메시지 및 컨트롤 흐름 분석 부분은 지원하지 않는다. 따라서, 다음과 같은 향후 연구 과제가 남아있다. 첫째, 표준화된 인터페이스 처리 코드를 자동으로 생성하여 인터페이스 처리 부분을 표준화 및 단순화함으로써 개발 효율을 증가시키는 방법의 연구를 진행할 예정이다. 둘째, 전체 시스템의 메시지 및 컨트롤 흐름을 분석하여, 컴포넌트 간의 네트워크 트래픽 분석 및 컨트롤 흐름에 대한 도식화를 제공함으로써 시스템 분석 도구로 활용하기 위한 방법에 대한 연구가 필요하다.

또한, 메시지 기반의 인터페이스 정의뿐만 아니라, 분산 객체 기반의 인터페이스 정의에도 적용할 수 있는 인터페이스 정의 및 관리 시스템으로의 발전 역시 향후 연구 과제이다.

## 참 고 문 헌


[1] Object Management Group, "Common Object Request Broker Architecture: Core Specification Version 3.0.3," Mar 2004.

[2] Microsoft, "The Component Object Model Specification," Microsoft Development Library, CD-14, Jan 1996.

[3] Object Management Group, "Data Distribution Service for Real Time System Version 1.2," Jan 2007.

[4] Micheal Bächle and Paul Kirchberg, "Ruby on Rails," IEEE Software, Mov 2007.