논 문 22-1-3

윈도우즈 플랫폼에서의 프로그램간 실행시간 상호연동 기술에 대한 조사

Interoperation Techniques On Windows Platform

신승철 노상훈 권민혁 임용수 김성민
한국기술교육대학교 정보미디어공학과
{scshin; noulne; minhyuk; gninraw; smilefe2}@kut.ac.kr

프로그램 상호연동 기술은 웹브라우져, 문서편집기 등 다양한 종류의 응용프로그램에서 사용되며 그 필요성이 높아지고 있다. 프로그램간 상호연동은 컴파일 시간에 이미 결정되기도 하지만, 내장 스크립트 언어를 통한 상호연동 처럼 실행 시간에만 결정할 수 있는 경우도 있다. 본 논문에서는 실행 시간에 상호연동 할 수 있는 방법에 초점을 맞추어 설명한다. 또한 국내 PC에서 널리 사용되고 있는 윈도우즈 운영체제를 중심으로 여러 플랫폼에 실제로 구현된 상호연동 기술의 사례를 조사하였다. 그리고 마지막으로 이 상호연동 기술들이 인터프리터 언어에서의 상호연동에 어떻게 응용될 수 있는지 제시한다.

1 서 론

서로 다른 언어로 작성되고 독립적으로 개발된 프로그램의 상호연동 필요성이 높아지고 있다. 국내의 컴퓨터 환경에서 쉽게 접할 수 있는 예는 인터넷 익스플로러이다. 비록 보안 문제들로 인해 비판을 받고 있긴 하지만 ActiveX라는 형태로 다양한 프로그램이 인터넷 익스플로러와 상호연동하며 동작하고 있다. 이 상호연동 기술을 통해 인터넷 익스플로러 안에서 바로 동영상을 재생하고 플래시를 재생한다. 또한, PDF나 HWP와 같은 다양한 문서파일을 바로 열어 볼 수 있는 것 역시 상호연동 기술을 통한 것이다. 만약 이러한 상호연동 기술이 없었다면, 인터넷 익스플로러의 기능 확장을 통해 얻을 수 있는 편리함을 취하기 어려웠을 것이다.

근래에는 엑셀과 같은 문서작성 프로그램에서부터 게임에 이르기까지 다양한 분야에서 인터프리터를 내장하고 이를 통해 여러 컴포넌트와 상호연동하는 응용도 이루어지고 있다. 특히 이러한 경우는 인터프리터를 내장하는 프로그램을 작성할 때 어떤 컴포넌트와 어떻게 상호연동할지 알 수 없으며, 실행시간에만 결정할 수 있다. 본 논문에서는 이러한 동적인 환경을 위해 어떤 상호연동 기술이 있는지 조사하였다.

상호연동에 있어서 어려운 점은 서로 다른 프로그래밍 언어로 작성된 프로그램간에 이루어진다는 점이다. 직관적으로는 두 프로그램의 프로그래밍 언어가 무엇이냐에 따라 다른 상호연동기술을 필요로 할 것 같지만, 상호연동 기술의 선택은 오히려 플랫폼의 종류에 의해 결정된다.

현재 국내 PC 운영체제의 대부분을 차지하고 있는 것은 윈도우즈이다. 본 논문에서는 윈도우즈 운영체제에서 많이 사용되고 있는 플랫폼인 WIN32, JVM 그리고 .NET 을 중심으로 상호연동 기술들을 살펴본다.

본 논문의 나머지 부분은 다음과 같이 구성된다. 2절에서는 상호연동 기술의 요소를 알아보고, 3절에서는 여러 상호연동 기술들을 소개한다. 그리고 마지막 절은 결론이다.

2 상호연동 기술의 요소

상호연동 기술을 고안할 때 고려할 점은 크게 두가지이다. 하나는 어떻게 연동하는 프로그램의 함수를 호출할 것인가이고, 또하나는 주고 받는 데이터의 호환성을 어떻게 부여할 것인가 이다.

함수를 호출하는 방법은 컴퓨터내 상호연동 인지 컴퓨터간 상호연동 인지에 따라 달라진다. 컴퓨터내 상호연동은 효율성을 중요시 하여 함수의 포인터를 이용한다. 호출하는 프로그램에서는 어떻게든 메모리상에 올라가 있는 이 함수포인터를 얻어내 직접 호출을 하는 방식을 취한다. 때문에 반드시 호출자(caller)와 피호출자(callee)는 반드시 같은 컴퓨터에 속해있어야 한다. 반면, 컴퓨터간 상호연동은 메시지를 전달하는 간접적인 방식을 사용한다. 많은 경우 이 메시지는 소켓을 통해 전달하며, 호출자는 피호출자가 다루는 — 메모리와 같은— 자원에 직접적인 접근을 하지 않는다. 이는 호출자와 피호출자를 완전히 분리시켜 주며, 동작하는 플랫폼이 완전히 분리되어 있더라도 무리 없이 동작 가능하다. 이는 분산처리와 관련된 특징이며 이러한 특성 때문에 이러한 컴포넌트를 분산 컴포넌트라고 한다.

분산 컴포넌트 여부에 따라서 상호연동 기술을 분류하면 **그림1** 과 같다.

Distributed Component	Non-distributed Component
CORBA DCOM Java RMI WCF	COM DLL/Managed DLL

[1] 분산 여부별 상호연동 기술 분류

컴퓨터간 상호연동 기술은 플랫폼 독립적인 특성을 갖지만, 컴퓨터내 상호연동 기술은 서론에서 언급한 것과 같이 플랫폼에 종속된다. 때문에 컴퓨터내 상호연동 기술은 **그림2** 와 같이 플랫폼에 따라 분류할 수 있다.

상호연동 기술의 요소중 데이터의 호환성과 관련된 문제는 마샬링(marshalling)/언마샬링(unmarshalling) 이라는 방법을 통해 해결한다. 두 상호연동 하는 프로그램이 서로 다른 프로그래밍 언어로 작성되어 있다면 한쪽 언어에는 존재하지만 다른 한쪽 에는 존재하지 않는 데이터타입이 존재할 가능성이 높다. 이는 단순한 데이터타입의 값일 수도 있지만, 포인터로 연결된 그래프 구조체와 같이 메모리 여기저기 흩어져 있는 복잡한 데이터가 될 수도 있다. 이런 데이터를 두 프로그램이 주고받으며 사용하려고 한다면, 두 언어가 호환할 수 있는 타입으로 인코드/디코드 하는 과정이 필요하다. 이러한 과정을 마샬링/언마샬링이라고 한다. 상호연동 기술에 따라 이에 대해서는 전혀 제한하고 있지 않은 기술도 있으며, 제한된 표준 데이터타입을 제시하여 항상 제시된 표준 데이터타입으로 변환하도록 하기도 한다.

Caller \ Callee	WIN32	.NET	JVM
WIN32	DLL COM Socket	Managed DLL COM Socket	DLL(JNI) COM(JCB) Socket
.NET	DLL COM WCF Socket	Managed DLL COM WCF Socket	COM(JCB) Socket
JVM	DLL(JNI) COM(JCB) Socket	COM(JCB) Socket	RMI Socket

[2] 플랫폼별 상호연동 기술 분류

3 상호연동 기술

앞 절에서 상호연동 기술의 두 요소에 대해서 살펴보았다. 이 절에서는 실제 사용되고 있는 상호연동 기술과 그 사용례를 살펴본다.

3.1 DLL

DLL(Dynamic Link Library)은 상호연동을 위해 WIN32 플랫폼에서 가장 기본적으로 사용되는 기술이다. 현재 다양한 플랫폼에서 가장 널리 사용되고 있는 언어는 C/C++ 언어이다. 그리고 이 플랫폼을 위해 만들어지는 실용적인 언어는 보통 C/C++ 언어와 연동을 고려한다. 이는 C/C++로 작성된 방대한 라이브러리를 활용할 수 있게하는 이점을 준다. WIN32 플랫폼에서는 이를 위해 DLL을 사용하며, 그 예로 Java, Python, OCaml 등을 들 수 있다.

DLL은 호출의 방법으로 함수 포인터를 이용한다. DLL 사용과 관련된 WIN32 API로는 LoadLibrary 와 GetProcAddress 가 있다. LoadLibrary는 동적으로 DLL을 로드하는 역할을 하며, GetProcAddress는 함수의 이름으로 인수로 주면 함수의 포인터를 알려준다.

이렇게 얻어낸 함수 포인터를 통해 직접 해당 함수를 호출할 수 있다. 단, 여기서의 함수 포인터는 함수의 코드 위치만을 의미하며, 함수의 형식인수가 어떤모양이며 반환타입이 어떻게 되는지는 전혀 반영하지 않는다. 그것은 전적으로 프로그래머에 의해 처리되어야 한다.

즉, 완전히 동적으로 DLL의 함수를 호출하려고 한다면 이 부분에 대해서는 타입 시스템의 유용성을 포기해야 한다. 인수를 잘못 넣은채 호출하는 것이 허용되며 이는 해당 프로그램의 심각한 문제를 초래할 수 있다.

이는 마샬링 문제에도 동일하게 적용된다. DLL을 통한 상호연동에서는 어떠한 기준도 제공되지 않으며 전적으로 상호연동하는 두 프로그램에 책임이 있다.

3.2 COM

COM(Component Object Model)역시 WIN32 플랫폼에서 기본적으로 제공되는 기술이다. DLL이 함수 수준의 상호연동을 제공하는 것과 달리 COM은 클래스 수준의 상호연동을 제공한다. DLL에서도 인스턴스를 만드는 함수를 이용하는 편법으로 클래스를 사용할 수 있다. 하지만 이 경우 두 프로그래밍 언어의 클래스가 호환이 되어야만 한다.

COM 오브젝트를 사용하기 위해서도 마찬가지로 COM 오브젝트와의 호환성이 필요하긴 하지만, 이 경우는 COM 기술 자체가 일종의 표준으로 쓰이기 때문에 불특정한 프로그래밍

언어로 작성된 프로그램간의 상호연동에도 사용할 수 있다.

DLL이 함수포인터를 사용하는 것과 같이 COM은 메소드의 호출을 위해 가상함수테이블(virtual function table)을 사용하여 직접적인 호출 방법을 제공한다. COM 오브젝트를 사용할 때 필요한 API는 COM 오브젝트를 생성하는 CoCreateInstance라는 함수 단 하나이다. COM 클래스는 특정한 매커니즘에 의해 생성한 GUID라는 전세계적으로 고유한 식별자를 갖으며, 이는 윈도우즈 레지스트리에 등록되어 COM 오브젝트 생성에 사용된다. CoCreateInstance로 주어지는 이 식별자는 같은 객체에 한해서는 전 세계적으로 고유하므로 단순히 DLL을 사용하는 경우 처럼 어떤 DLL을 로드할지 결정하기 위해 LoadLibrary를 하는 작업은 불필요하다.

모든 COM 오브젝트는 IUnknown 인터페이스를 구현함으로써 만들어진다. IUnknown 인터페이스에는 QueryInterface, AddRef, Release의 세 개의 메소드가 존재한다. 뒤의 두개는 레퍼런스 카운팅 목적의 메소드이며, QueryInterface는 일종의 다운캐스팅과 같은 역할을 한다. 이 인터페이스를 통해 특정한 메소드를 포함하는 다른 인터페이스의 클래스로 변환할 수가 있다. COM 의 기본 인터페이스에는 비주얼베이직과 같은 인터프리터 언어를 지원하기 위해 IDispatch 인터페이스가 있다. 이 인터페이스에는 GetIDsOfNames와 Invoke라는 두 메소드가 있다. 각각은 메소드의 이름으로 메소드 아이디를 얻어내고, 메소드 아이디로 해당 메소드의 호출을 대행해주는 메소드이다. 때문에, 이 COM 오브젝트에 이 메소드가 구현되어 있다면 QueryInterface를 통해 이 인터페이스를 얻어내어 완전히 동적인 시간에 메소드를 사용할 수 있다.

COM 은 이러한 메소드의 호출방법에 관한 측면만을 정의하고 있지 않다. COM 오브젝트의 정의는 유한한 개수의 데이터타입만으로 정의가능하다. COM 오브젝트를 이용하는 프로그램들은 때문에 적절하게 마샬링/언마샬링을 해야한다. 하지만 DLL 에서와 달리 이 데이터타입들은 일종의 표준으로 사용될 수 있기 때문에 불특정한 프로그래밍 언어로 작성된 프로그램과도 무리없이 상호연동할 수 있다.

3.3 Java RMI

윈도우즈 상에서 동작하는 자바 가상기계는 JNI를 통해 윈도우즈의 DLL과 상호연동을 제공해준다. 이러한 기술은 운영체제에 종속되게 되며, 자바가 갖는 플랫폼 독립적인 특성을 잃게한다.

자바는 Java RMI라는 플랫폼 독립적인 분산 컴포넌트 기술을 사용하고 있다. 이를 사용하기 위한 과정은 다음과 같다.

1. 서비스 할 원격 객체의 인터페이스 작성
2. 원격 객체및 서버프로그램 작성
3. 원격 객체를 사용 할 클라이언트 프로그램 작성
4. 스텁(stub)과 스켈레톤(skeleton) 생성
5. RMI 레지스트리 실행
6. 서버/클라이언트 실행

여기서 주목할 만한 점은 스텁과 스켈레톤이다. 이는 원격 객체의 프로그램이 있으면 툴을 이용해 자동으로 생성할 수 있는데, 스텁은 클라이언트에서 사용하고 스켈레톤은 서버—즉, 원격 객체—에서 사용한다.

스텁은 원격 객체와 동일한 인터페이스를 가지고 있어, 클라이언트는 원격 객체의 메소드를 마치 일반 객체의 메소드를 호출하는 것 처럼 사용할 수 있다. 단지, 그 호출 대상이 원격 객체 자체가 아니라 스텁 객체라는 점이 다를 뿐이다. 스텁 객체는 TCP를 통해 서버쪽의 스켈레톤 객체와 연결되어 있어, 스텁 객체의 메소드가 호출되면 이 연결을 통해 스켈레톤

객체에 메소드 호출을 요청한다. 스켈레톤 객체는 TCP를 통해 메소드 호출 요청이 들어오면 해당 메소드에 대응하는 원격 객체의 메소드를 대신 호출해 준다.

이와 같은 구조는 클라이언트의 입장에서 마치 로컬 컴퓨터에 존재하는 메소드를 호출하는 것처럼 원격 객체를 다룰 수 있게한다.

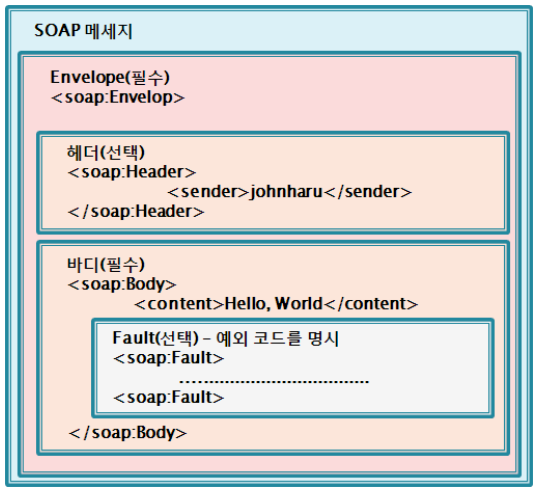
두 객체간의 통신에는 마샬링/언마샬링의 과정을 거치며 이는 바이트스트림의 형태로 이루어진다. Java RMI 는 같은 Java 프로그래밍 언어간의 상호연동임에도 불구하고 TCP 통신을 통해 호출 메시지가 전달되어야 하므로 이러한 인코딩/디코딩 과정이 필요하다. 이는 분산 컴포넌트가 비분산 컴포넌트에 비해 성능적 약점을 갖음을 보여준다.

Java RMI는 이렇게 분산환경에서 상호연동을 할 수 있는 장점을 갖고 있지만, 메소드 호출 과정의 간접성과 복잡성에 의해 상당한 성능하락을 가져온다.

3.4 WCF

WCF(Windows Communication Foundation)는 .NET 프레임웍 3.0 이상에서 제공해주는 상호연동 기술이다. 이 기술은 SOA(Service Oriented Architecture)를 기반으로 하며, SOAP(Simple Object Access Protocol) 메시지를 이용해 상호간에 정보를 주고받는다.

SOAP 메시지는 **그림3**과 같은 형태의 XML로 구성된다. SOAP Header는 전송할 서비스의 주소, 라우팅과 같은 메시지 처리에 필요한 정보들을 기술하며, SOAP Body는 실질적인 메시지를 기술한다.



[3] SOAP 메시지의 구조

SOAP 메시지는 http나 smtp와 같은 채널을 통해 전달이 된다. 이 방식의 장점은 로컬 컴퓨터에 종속되지 않는다는 점이다. 앞에서 언급되었던 DLL이나 —DCOM이 아닌—COM과 같은 상호연동기술은 코드가 메모리에 올라가고 그 코드에 대한 포인터를 통해 접근하기 때문에, 반드시 같은 컴퓨터에 있어야만 정상적으로 사용할 수 있다.

XML은 매우 복잡한 자료구조도 일관된 방식으로 표현할 수 있다. 때문에 SOAP 메시지가 XML의 형태로 표현된다는 점은 불특정 언어와의 상호연동의 마샬링/언마샬링의 측면에서 큰 장점으로 볼 수있다. 하지만 이러한 형태로의 인코딩/디코딩은 파싱 단계를 포함해야 하기 때문에 상당한 성능의 하락을 수반한다.

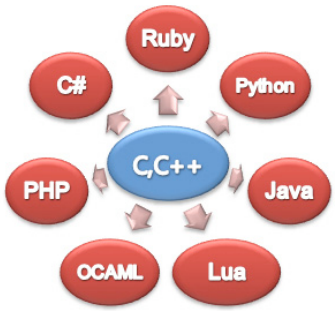
[I]기본적인 타입 처리

int, long, short	→	target integers
oat, double	→	target oats
char, char	→	target strings
bool	→	target boolean(target language 가 지원하지 않을 경우 integer 0,1)
structure, array, class	→	pointer
long long, long double	→	unsupported

3.5 SWIG

앞에서는 상호연동 기술에 따른 호출 방식의 차이와 마샬링/언마샬링이 어떻게 다루어지는지에 대해서 살펴보았다.

SWIG(Simpli ed Wrapper and Interface Generator)는 앞에서 소개되었던 상호연동 기술과는 달리 마샬링/언마샬링에 관한 문제만을 다룬다. 이 툴은 **그림4**과 같이 C/C++ 로 작성된 프로그램을 다른 다양한 스크립트 언어들에서 사용할 수 있도록 도와준다.



[4] SOAP 메시지의 구조

비록 COM과 같은 상호연동 기술이 일종의 표준 데이터타입을 제공해준다고 하여도 마샬러/언마샬러의 구현은 프로그래머가 직접 해주어야 한다. 프로그램간 상호연동에서 발생하는 버그의 상당수가 마샬러/언마샬러의 잘못된 구현에 기인한다. 프로그램을 넘어 상호연동을 하기 위해 다른 데이터타입의 값으로 변환하는 과정은 타입검사를 통한 안전성의 확보를 무력화 시킨다.

SWIG는 C/C++의 원시프로그램을 입력받아 마샬러/언마샬러를 자동으로 생성해 주어 이러한 문제를 경감시켜준다.

SWIG는 **표I**에서 보여주는 것과 같이 C/C++의 타입을 목표 스크립트 언어에서 가장 유사한 타입으로 변환하는 것을 가정하지만, 정교한 수준의 마샬링/언마샬링을 제공하지는 않는다.

4 상호 연동 기술의 인터프리터로의 적용

앞에서는 여러가지 종류의 상호연동 기술에 대해서 조사하였다. 이를 인터프리터로의 적용하기 위해서는 다음과 같은 두가지 경우를 고려할 수 있다.

1. 기존에 존재하는 인터프리터를 사용한다.
2. 새로운 인터프리터를 구현한다.

1의 방법을 사용하기 위해서는 기본적으로 해당 인터프리터가 상호연동할 수 있는 방법을 이미 내장하고 있어야 한다. 윈도우즈 플랫폼을 위한 인터프리터는 일반적으로 DLL 사용방법을 제공하므로 이를 이용할 수 있다. 만약 DLL을 사용할 수 있다면 다른 COM/CORBA와 같은 다른 상호연동 기술도 간접적인 방법으로 사용할 수 있다. 예를 들면 C로 COM 오브젝트를 조정할 수 있는 래퍼함수를 만들 수 있다. 이를 DLL로 만들어 인터프리터에서 사용을 한다면, 함수 호출의 형태로 COM 오브젝트를 사용할 수 있다. COM 오브젝트와의 상호연동의 경우 IDispatch 인터페이스를 사용한다면 특정 COM 오브젝트가 아닌 IDispatch 인터페이스로 구현한 모든 COM 오브젝트를 다룰 수 있으므로 이를 래퍼함수로 만들어 사용하면 된다. 만약 인터프리터가 DLL은 지원하지 않지만 소켓기능을 제공한다면 CORBA의 구현 원리를 이용할 수도 있다. DLL의 경우와 같이 C로 래퍼함수를 만들되 서버프로그램의 형태로 만든다. 이 프로그램이 패킷에 의해 래퍼함수를 호출할 수 있는 방법을 제공해 준다면 DLL로 호출하는 경우와 동일한 능력을 갖는다. 최악의 경우는 오직 기본입출력(Standard I/O)만을 제공하는 경우이다. 통합개발환경과 컴파일러간의 상호연동에 많이 사용되는 방법으로 컴파일러의 에러와 같은 콘솔 출력메시지를 통합개발환경에서 GUI로 표현한다. 소켓 대신 기본입출력을 호출 의뢰의 채널로 삼으면 앞에서 언급한 방법들로 다른 상호연동 기술을 사용할 수 있다.

2의 방법을 사용하는 것은 1번 방법의 경우와 비교하면 매우 사소하다고 볼 수 있다. 해당 인터프리터를 개발하는 언어에서 상호연동하는 기술을 가지고 있으면 직접 구현하면 된다.

COM 오브젝트와 같이 오브젝트를 인터프리터에서 사용하기 위해서는 또다시 두가지 경우를 고려해볼 수 있다.

1. 인터프리터가 클래스를 제공하지 않는다.
2. 인터프리터가 제공하는 클래스가 COM 오브젝트와 호환되지 않는다.
3. 인터프리터가 제공하는 클래스가 COM 오브젝트와 호환되는 제공한다.

1의 경우는 앞서 서술했던 것 처럼 래퍼함수를 통한 간접적인 방법을 사용해야 한다. 2의 경우 인터프리터는 동적으로 클래스를 생성할 수 있으므로 인터프리터 쪽에서는 래퍼 클래스를, DLL 쪽에서는 래퍼함수를 만들어 사용할 수 있다. 3의 경우는 COM 오브젝트를 직접 사용한다.

5 결 론

상호연동 기술은 상호연동이 어떤 프로그래밍 언어간에 이루어지는지 보다 어떤 플랫폼간에 이루어지는지에 따라 분류된다. 본 논문에서는 윈도우즈 운영체제에서 많이 사용되는 플랫폼인 WIN32, .NET 그리고 JVM 에 대해 어떤 상호연동 기술들이 있는지 조사하였다.

상호연동 기술을 구성하는 두가지 요소인 호출 방법과 데이터 호환의 속성이 실제 구현된 상호연동 기술에서 어떻게 나타나는지를 알아보았다. 호출 방법의 차이는 메모리 포인터와 TCP 통신의 두가지로 나뉘며 이는 분산 컴포넌트의 가능성을 결정한다. DLL과 COM은 로컬 컴퓨터에서의 상호연동에 이용할 수 있으며, WCF와 Java RMI는 분산환경에서 상호연동에 이용할 수 있다. 하지만 분산환경에서의 호출 방식은 로컬 컴퓨터에서의 호출보다 더 간접적이고 복잡한 특성으로 인해 상대적으로 성능이 떨어지는 단점을 갖는다. 데이터 호환의 측면은 프로그래밍 언어간에 데이터타입이 호환되지 않을 가능성이 높기 때문에 발생하며, 바이트스트림이나 XML과 같은 형태로 마샬링/언마샬링 하여 해결한다. 일반적

으로는 마샬러/언마샬러를 프로그래머가 직접 구현해야하지만 이 과정에서 많은 상호연동의 문제를 야기하며, 이러한 문제는 SWIG와 같은 마샬러/언마샬러 생성툴을 이용하여 경감시킬 수 있다. 하지만, 현재로써는 그 한계가 있음을 확인 할 수 있었다.

마지막으로 인터프리터 언어에서 상호연동을 사용하려고 할 때 대상 인터프리터 언어 혹은 구현하는 언어가 클래스나 COM 오브젝트를 지원하는지에 따른 여러 상황이 있음을 보이고, 각 상황에서 앞 서 소개한 상호연동 기술들을 어떻게 적용할 수 있는지를 제시하였다.

[1] Interoperability, MSDN, Microsoft, 2008, http://msdn.microsoft.com/en-us/library/ms172270(VS.80).aspx

[2] Interop Marsahlling, MSDN, Microsoft, 2008, http://msdn.microsoft.com/en-us/library/04fy9ya1(VS.80).aspx

[3] DLLs, MSDN, Microsoft, 2008, http://msdn.microsoft.com/en-us/library/1ez7dh12(en-us,VS.80).aspx

[4] Win32 and COM Development, MSDN, Microsoft, 2008, http://msdn.microsoft.com/en-us/library/aa139672.aspx

[5] Windows Communication Foundation, MSDN, Microsoft, 2008, http://msdn.microsoft.com/en-us/netframework/aa663324.aspx?PHPSESSID=0fbb7c61

[6] Using WCF, MSDN, Microsoft, 2008, http://msdn.microsoft.com/en-us/library/ms735119.aspx

[7] Getting Started Tutorial, MSDN, Microsoft, 2008, http://msdn.microsoft.com/en-us/library/ms734712.aspx

[8] The Java Native Interface Programmer's Guide and speci cation, Sun Microsystems, 2008, http://java.sun.com/docs/books/jni/html/jniTOC.html

[9] George T. Heineman, William T. Councill, Component-Based Software Engineering: Putting the Pieces Together, Addison-Wesley Professional, 2001

[10] Enterprise JavaBeans Technology Home page, Sun Microsystems, 2008, http://java.sun.com/products/ejb/index.jsp

[11] Sheng Liang, The Java Native Interface Programmer's Guide and Speci cation, Addison wesley, 1999

[12] David M.Beazley, SWIG Users Manual, Departmaent of Computer Science, University of Utah, 1997

신승철

– 1992-1996 인하대 전자계산학과 박사
– 1999-2000 캔사스 주립대 연구원
– 1996-2006 동양대 컴퓨터학부 부교수
– 2006-현재 한국기술교육대 인터넷미디어학부 조교수

<> 프로그래밍 언어, 프로그램 분석 및 검증, 소프트웨어 보안, 수리 논리

노상훈

– 2006-2008 한국기술교육대 석사 2008-현재 한국기술교육대 박사과정

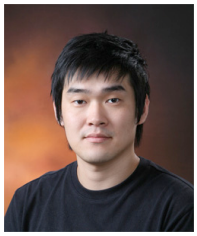
<> 프로그래밍 언어, 프로그램 분석 및 검증, 소프트웨어 보안, 수리 논리

권민혁

– 2007-현재 한국기술교육대 석사과정

<> 프로그래밍 언어, 프로그램 분석 및 검증

임용수



– 2001-2008 한국기술교육대 학사

<> 프로그래밍 언어, 프로그램 분석 및 검증

김성민

– 2001-2008 한국기술교육대 학사

<> 프로그래밍 언어, 프로그램 분석 및 검증