정보
산업동향

# C# : 컴포넌트 지향 언어

홍 영 준*

● 목 차 ●



## 1. 서 론

지난 20년간 C/C++는 소프트웨어 개발 시 가장 널리 사용된 프로그래밍 언어였다. 이 들은 하위 수준 접근으로서의 뛰어난 유연성을 위해 생산성 및 안정성을 희생시킨 대표적인 예라고 할 수 있다 [1]. 개발자들은 현재의 짧은 소프트웨어 개발주기와 하드웨어의 발전을 근거로 생산성과 안정성을 보장하는 새로운 언어를 요구하기 시작했다. JAVA를 비롯한 많은 언어들이 이러한 요구를 충족시키기 위해 만들어졌으나 많은 경우 이들은 생산성과 안정성을 목표로 하였으며 결과적으로 C/C++가 가지는 유연성 및 편리성을 확보하지는 못했다. 그러나 이러한 유연성은 시스템에 대한 세부적인 제어 뿐 아니라 이전 시스템과의 연동에 있어서도 중요한 부분이다.

2000년 8월, 마이크로소프트사는 이후 그 이름이 .NET Framework로 변경된 NGWS (New Generation Windows Service)와 함께 C#이라는 프로그래밍 언어를 발표하였다. C#은 전체적으로 컴포넌트 지향 언어이면서 C/C++의 유연성과 JAVA의 생산성 및 안정성 사이에 균형을 유지하는 언어라고 할 수 있다. .NET Framework는 .NET 실행 환경인 CLR (Common Language Runtime) 및 네트워크, 웹, 데이터 접근, XML, XML 웹 서비스 및 서버 자원에 관련된 풍부한 클래스 라이브러리를 포함하고 있다. 본 논문의 기술 순서는 먼저 .NET Framework및 C#의 기능을 특징 별로 소개하고 표준화 과정과 함께 현재와 미래에 대한 방향을 소개하고 마지막으로 결론을 맺는다.

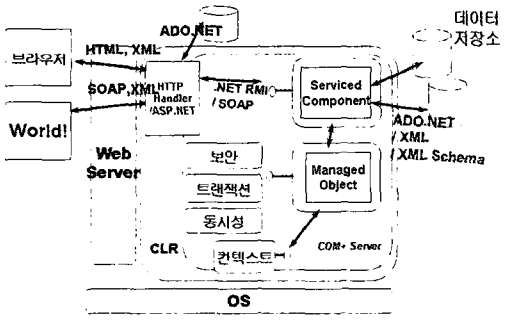
* 한국 마이크로소프트(주) 닷넷 에반젤리스트

## 2. .NET Framework에 대한 소개

.NET Framework는 JAVA와 마찬가지로 플랫폼 독립성을 제공하도록 설계되었다. JAVA VM에 해당하는 CLR과 JAVA 바이트 코드에 해당하는 IL(Intermediate Language)이 그것이다. 즉, 실행 파일은 IL로 컴파일 되며 CLR상에서 동작한다. 물론 필요한 기능들은 각종 클래스 라이브러리를 통해 제공한다. .NET Framework와 JAVA와의 차이점은 언어 독립성이라고 할 수 있다. 컴파일러가 CLS (Common Language Specification)을 만족하도록 작성된 언어라면 어떤 언어에서도 사용 가능하다. 현재 .NET Framework 상에서 동작 가능하거나 컴파

일러가 작성중인 언어로는 C#, VB.NET, C++, Python, COBOL, FORTRAN, Scheme, Haskell과 같은 언어들이 있다[2]. 각 언어로 작성된 컴포넌트는 상속 및 위임에 의한 재 사용이 가능할 뿐 아니라 .NET Framework가 지원하는 모든 환경에서 공통적으로 보장된다. (그림 1)은 .NET Framework의 전체 아키텍처를 도식화한 것이다.

.NET Framework의 또 다른 특징은 XML에 기반했다는 점이다. 즉, XML에 대한 지원을 부가적인 툴 킷을 이용하는 것이 아니라 데이터 접근, 메세징, XML Web Service에 있어 XML을 기초로 설계되었다. 프로그래머는 XML에 대한 지식 없이 (그림 1)에 보이는 각 계층간의 메시징과 데이터 교환에 있어 XML을 이용할 수 있다. 즉, JAVA가 플랫폼간의 통합을 위해 CORBA의 IIOP를 선택했다면 .NET은 XML 기반의 SOAP과 XML을 선택했다. 또한 JAVA의 경우 JSE(JAVA 2 Platform, Standard Edition), JME(JAVA 2 Platform, Micro Edition), J2EE(JAVA 2 Platform, Enterprise Edition)가 발표된 반면, .NET의 경우 .NET Framework와 .NET Compact Framework가 있으며 .NET Framework는 JAVA의 J2EE와 J2SE에 해당하며 .NET Compact Framework는 JME에 해당한다.




(그림 1) .NET Framework의 구조

## 3. C# 언어에 대한 소개

C#은 .NET Framework의 우선적인 언어로서 CLR을 직접 반영한다. C#은 먼저 컴포넌트 지향 언어로서 설계되었다. 컴포넌트 구현 시 필요한 개념들을 일 등급 구성자로 취급한다. 이를 위해 프로퍼티, 인덱서, 이벤트, 실행 및 디자인 시 사용할 수 있는 어트리뷰트(attribute)를 지원한다. 또한 자가 설명적인 컴포넌트를 위해서 XML 주석을 지원한다. 이러한 구성 요소들을 이용하여 컴포넌트 구현 시 필요한 개념들을 직접 표현할 수 있으며 분산 환경에서조차 별도의 헤더 파일이나 IDL (Interface Definition Language)을 작성할 필요가 없다. 또한 .NET 프레임워크의 언어 독립적인 특성과 결합하여 재사용 성을 향상시킬 수 있다. 어트리뷰트는 실행환경과 RAD 툴에 필요한 정보를 선언적으로 제공하기 위하여 사용된다. 어트리뷰트를 이용하여 컴포넌트가 동작하는데 필요한 설정 사항을 실행환경에게 요구할 수 있으며 RAD(Rapid Application Development) 툴에서 작성된 컴포넌트를 사용할 경우 필요한 정보를 전달할 수 있다[3,4].

C#은 SmallTalk와 마찬가지로 모든 타입을 최상위 클래스인 object로부터 파생된 객체로 다룬다. C/C++, JAVA에서의 내장 형(built-in type)은 별도의 값 타입(value type)으로서 객체로 취급할 수 없다. 이는 모든 타입을 객체로 다루게 될 경우 발생할 성능 비용을 줄이기 위해서 선택된 방식이다. C#은 통합 자료 형 시스템(Unified Type System)으로 추가적인 성능 비용 없이 내장 형과 같은 값 타입(Value Type)과 참조 타입(reference type)을 통합하였다. 이를 위하여 C#은 Boxing/Unboxing이란 기능을 이용한다. Boxing/Unboxing은 값 타입과 참조 타입 간의 자동 변환 기능으로 CLR에서 직접 지원한다. 예를 들어서 아래의 프로그램을 보자.

```
object o = 1;       // Boxing
int i = (int) o;  // Unboxing
```

이 경우 1은 정수 내장 형으로 값 타입이다. 이를 참조 타입인 object로 형 변환 할 경우 변환 과정을 Boxing이라 하며 그 반대의 경우 객체 o를 다시 내장 형 int로 형 변환 할 경우 필요한 변환 과정을 Unboxing이라 한다. 또한 C#은 JAVA와 달리 사용자 정의 값 타입을 제공한다. 즉, 구조체와 열거 형을 이용하여 내장 형 외에 값 타입을 정의할 수 있다.

C#은 위임 형(delegate)을 제공한다. 위임 형은 일종의 함수 포인터로 생각할 수 있으나 C/C++와 달리 안전하다. 위임 형은 이벤트와 결합하여 이벤트 처리 코드를 작성하거나 쓰레드 생성, .NET Framework의 비동기 프로그래밍 패턴, 기존의 C/C++ 함수 호출 시 실제 함수 포인터로 사용할 수 있는 등 그 활용 예가 매우 다양하다. 동일한 기능을 구현할 경우 이러한 위임 형을 이용한다면 JAVA, C/C++의 경우에 비해 코드가 상당 부분 간결해진다.

C#은 신뢰성과 안전한 프로그램을 개발을 위하여 설계된 언어로 가비지 컬렉션, 구조적 예외 처리, 버전 관리 기능을 제공한다. 가비지 컬렉션과 구조적 예외 처리는 JAVA와 유사하다[5]. 그러나 가비지 컬렉션을 사용할 때 발생할 수 있는 자원 낭비를 막기 위하여 C#은 using 블록과 IDisposable 인터페이스를 제공한다. 즉, using 블록 내에서 선언된 객체가 IDisposable 인터페이스를 구현하였다면 using 블록의 실행이 종료하면 IDispose 인터페이스의 Dispose() 메소드 호출을 보장한다.

C#은 메소드 오버라이드를 위하여 반드시 virtual, override 예약어를 사용해야 하는데 이는 가상 메소드 테이블의 크기에도 영향을 미치지만 무엇보다도 버전 관리 기능을 제공하기 위함이다. JAVA, C/C++의 경우 부모 클래스에서 새로운 가상 메소드의 추가 시, 추가된 가상 메소드의 시그너처가 자식 클래스의 메소드와 반환 형만 다를 경우 컴파일 혹은 실행 시 예외를 발생 시킨다[5,6]. C#은 어떠한 상황에서도 처음 사용자가 의도했던 메소드가 사용된다.

C/C++의 포인터는 강력하지만 위험한 기능이다. C/C++의 문제점은 배열 다루기부터 로우 레벨 접근, 다형적 변수 사용에 이르기까지 전체적으로 포인터 사용을 강요한다는 점이다[1]. C#은 포인터 사용을 하위수준 접근 혹은 기존 시스템과의 연동을 목적으로 그 사용을 제한할 수 있다. unsafe로 선언된 메소드 내에서만 포인터 사용이 가능하며 C/C++에서 사용 가능한 포인터 형, 포인터 연산, 형 변환 연산자의 대부분을 사용할 수 있다. C#의 포인터 형은 함수 포인터로 사용이 가능한 위임 형, 사용자 정의 값 타입의 물리적 메모리 레이아웃을 조정하기 위한 StructLayout 어트리뷰트, 그리고 가비지 컬렉션으로 하여금 특정 메모리 영역을 보호하게 하는 fixed 블록과 함께 사용될 수 있다. 이것은 일종의 임베디드 C라고 할 수 있으며 C/C++의 유연성을 제공한다. 이러한 unsafe 선언 구문은 IL에 직접 포함되므로 실행 환경은 작성된 메소드의 unsafe 선언 유무를 조사할 수 있다. 또한 잘못된 포인터 사용을 구조적 예외처리로 다루므로 C#의 포인터 연산은 C/C++만큼 위험하지 않다.

기타 C#은 논리적인 클래스 구성을 가능하게 하는 네임스페이스, #if, #elseif, #define같은 컴파일러 지시어, 연산자 오버로딩, JAVA 및 C/C++의 Jagged 배열과 함께 다차원 배열을 포함하고 있다.

마이크로소프트사의 캠브리지 연구소는 Haskell, Eiffel과 유사한 형태의 패러미트릭 폴리모피즘(parametric polymorphism)에 대한 CLR 지원을 연구하고 있다. 이들은 IL및 CLR자체를 확장하고 있으며 이미 구현을 마친 상태이다[7]. 이러한 연구 결과는 향후 .NET Framework에 반영될 예정이며 매우 긍정적으로 평가되고 있다.

## 4. C# 언어의 현재와 미래

C#과 .NET Framework의 관계는 C++와 UNIX에 비교할 수 있다. .NET Framework및 CLS를 지원하는 모든 언어가 .NET Framework기반에서 동작한다. 그러나 C#은 .NET Framework의 우선적인 언어로서 CLR을 직접 반영하고 있으며 .NET Framework의 많은 부분이 C#으로 작성되어 있다. C#, .NET Framework에 관련된 표준은 ECMA에 제출되었으며 TC39의 TG2에서 C# 표준을 TG3에서 CLI(Common Language Interface)및 표준 클래스에 대한 표준을 개발 중이다[4].

현재 C#은 Microsoft.NET에서만 동작하지만, 곧 .NET과 함께 모든 플랫폼에 이식 될 예정이다. FSF(Free Software Foundation)은 리눅스 기반의 .NET Framework라고 할 수 있는 MONO 프로젝트를 지원하고 있다. MONO 프로젝트는 리눅스 기반의 C# 컴파일러 역시 포함하고 있으며 GPL 라이센스를 따르고 있어 모든 소스가 공개되고 있다. 마이크로소프트사는 현재 FreeBSD에 .NET Framework및 C# 컴파일러를 이식 중이며 Microsoft Shared Source Framework에 따라 소스를 공개하게 될 것이다. Halcyon사는 JAVA.NET을 곧 발표할 예정이다. JAVA.NET은 IL코드를 JAVA 플랫폼에서 동작시킨다. 이것이 발표되면 C#으로 작성된 프로그램은 메인 프레임에서 PC까지 JAVA 플랫폼이 동작하는 모든 환경에서 동작하게 된다. 또한 OMG는 CORBA-C# 매핑을 지원할 것이라고 밝힌바 있다. Evans Data사는 북미 지역의 개발자 15%가 2001년도에 C#을 사용해 볼 것이라는 조사 자료를 발표했다. 또한 ZDNET KOREA사의 조사에 따르면 설문 응답자의 19.44%가 C#이 JAVA보다 많이 사용되는 언어가 될 것이라고 응답했다.

## 5. 결 론

C#은 전체적으로 C++과 JAVA의 중간 정도에 위치한 언어라고 할 수 있다. C++의 경우 많은 기능을 제공하지만 복잡할 뿐 아니라 각 언어 구성요소들이 서로 충돌하기 때문에 의미의 모호함이 심각하다. JAVA는 단순함을 목표하였다[5]. 그러나 언어의 단순함은 오히려 학습과 코드의 복잡함을 유도할 수 있다. C#은 기본적으로 JAVA의 아이디어와 함께 C++의 필요한 기능들을 모호함 없이 정제하여 받아들인 것이 그 특징이라고 할 수 있다.

C#이 발표 된지 1년이 되지 않아 다양한 플랫폼 및 언어들이 함께 발표되고 있으며 개발자 사회의 관심 또한 매우 크다고 할 수 있다. C#을 마이크로소프트사가 발표했다는 이유 외에도 이 언어가 가지는 장점이 잘 부각되고 있기 때문이라고 생각한다. 현재의 추세가 계속된다면 C#은 다양한 플랫폼에 이식될 것이다. 또한 C#은 .NET Framework와 독립적으로 발전할 가능성이 있으며 결국에는 컴포넌트 시대의 C++과 같은 역할을 수행하게 될 것이다.

## 참고문헌

[1] Sakkinen, M., "The darker side of C++ revisited" Structured Programming, 13(4): 155-177, 1992.

[2] Microsoft Corp., "Third Party .NET Developer Resources", MSDN Online, http://msdn.microsoft.com/net/thirdparty/default.asp.

[3] Eric Gunnerson, "A Programmer's Introduction to C#" Apress, 2000.

[4] Microsoft Corp., "Draft C# Language Specification", MSDN Online, http://msdn.microsoft.com/net/ecma/, March 2000.

[5] David Holmes, James Gosling, Ken Arnold, "The Java Programming Language Third Edition", p65,

p195, Addison Wesley, 2000

[6] Bjarne Stroustrup, "The C++ Programming Language Third Edition", p389, Addison Wesley, 1997.

[7] Andrew Kennedy, Don Syme, "The Design and Implementation of Generics for the .NET Common Language Runtime", Microsoft Research, http://research.microsoft.com/projects/clrgen/, May 2001.

## 저자약력

**홍 영 준**

1998년 동아대학교 컴퓨터공학과 (공학사)
2000년 동아대학교 컴퓨터공학과 (공학석사)
2001년 동아대학교 컴퓨터공학과 박사과정
2001년 한국 마이크로소프트㈜ 닷넷 에반젤리스트
관심분야 : 닷넷, C# Generic, 데이터베이스, XML Web Services
e-mail : yjhong@microsoft.com