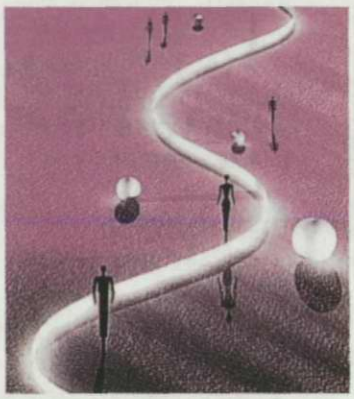
객체지향기술을 어떻게 볼 것인가



객체지향기술(OOT)이 주목을 받으면서 다양한 이론들이 발표되었고, 산업계에서 적용이 시도되어 왔으나 그 인기에 비해 OOT의 저변을 확대시키지는 못하고 있다. OOT의 도입, 확산에 걸림돌이 되고 있는 오해들을 살펴봄으로써, 객체기술을 도입하고자 하는 기업들의 올바른 이해를 돕고자 한다.

김영온 / LG-EDS시스템 기술연구부문 객체기술팀 차장

90년대에 접어들면서 정보기술분야에서 주목을 받아오던 대표적인 기술 중 하나인 객체지향기술(OOT:Object-Oriented Technology)은, 인터네트의 등장과 함께 인터네트/인트라네트 애플리케이션 개발에 있어서 핵심기술로서 다시 주목을 받게 되었다.

OOT가 주목을 받으면서 다양한 이론들이 발표되었고, 산업계에서 적용이 시도되어 왔으나 그 인기에 비하여 OOT의 저변을 확대시키지는 못하였다. OOT의 도입, 확산에 걸림돌이 되고 있는 오해들을 살펴봄으로써, 객체기술을 도입하고자 하는 조직들의 올바른 이해를 돕고자 한다.

OOT를 적용하면 높은 생산성 향상을 얻을 수 있다

새로운 기술을 조직에 도입하기 위해서는 추가적인 투자와 노력이 필요하다. 일반적으로 OOT는 구조적 방법론/정보공학 또는 절차형 프로그래밍(Procedural Programming)보다 생산성 향상을 위한 좋은 도구임에는 틀림없다.

그러나 조직의 성숙도(조직 문화와 구성원의 기술 수준 등)를 고려한 적절한 투자와 노력이 있어야만 OOT 도입을 통한 효과를 얻을 수 있다. 즉, 생산성은 첫번째 프로젝트에서부터 얻기는 어려우나, 기술이 축적되어 감에 따라 그 이후의 프로젝트에서 중·장기적으로 얻게 될 것이다.

OOPL을 사용하지 않으면 OOT를 적용하는 것이 아니다

상속(Inheritance), 다형성(Polymorphism), 캡슐화(Encapsulation), 정보은닉(Information Hiding) 등 객체지향 패러다임을 지원하는 객체지향 프로그래밍 언어(OOPL: Object-Oriented Programming Language)는 C++, 자바, 스몰토크, ADA'95, OO코볼 등이 있으며, 객체지향 4세대 언어(OO 4GL:Object-Oriented Fourth Generation Language)로는 포르테, 델파이, 비주얼 베이직, 파워빌더 등이 있다.

C++가 가장 많이 사용되고 있는데, C++는 C를 확장하여 객체지향을 지원하도록 만든 언어(Hybrid Language)이다. C++같은 하이브리드 언어는 절차형 프로그래밍을 허용함으로써 객체지향의 장점을 희생시킬 수 있으므로, C++를 객체지향에서 효과적으로 사용하기 위해서는 철저한 표준 및 절차에 따라 프로젝트가 관리되어야 한다.

스몰토크는 자바의 등장으로 사용이 점차 감소하는 추세이며, 자바는 현재 가장 주목받고 있는 프로그래밍 언어로서 자리를 잡아가고 있으나, 실제 애플리케이션에서 본격적으로 이용되기까지는 다소

시간이 걸릴 것이다. 그러나 자바가 향후 주축언어로서 자리잡을 것이 확실시되므로, 자바에 대한 준비를 생각할 때이다.

OO 4GL은 중·소규모의 애플리케이션이나, 빠른 납기를 요구하는 시스템 또는 프로토타입 개발에 주로 사용되고 있는데, 시스템의 확장성과 성능, 기능 등에서 제한적이다. 그러나, OO 4GL의 기능이 개선되면서 점차 사용 영역이 증가하여 왔고, 앞으로도 꾸준히 사용이 증가할 것이다. 현재 사용되고 있는 대부분의 4GL이 OO 4GL이다. 포르테와 델파이는 본래부터 객체지향 패러다임하에서 개발되었으며, 비주얼 베이직과 파워빌더는 최근 버전에서 객체기술을 지원하기 시작하였다.

성능, 사용하는 하드웨어 및 소프트웨어 등 환경의 제약에 따라 OOPL의 사용이 어려울 수도 있다. 이상적으로는 OOPL만을 사용하는 것이 바람직하나, 개발하려는 애플리케이션에 따라 C와 같은 3세대 언어를 사용하거나, 생산성 향상을 위하여 4GL을 병행하여 사용하는 것이 바람직하다.

OOT는 패러다임의 변화를 필요로 한다

패러다임의 변화는 OOT를 말할 때 가장 많이 언급되는 말이고, OOT의 도입이 어렵다는 이유로서 잘못 받아들여지고 있는 말이다. 패러다임의 변화 정도는 OOT를 받아들이려고 하는 조직의 문화 및 기술수준에 따라 다르다.

예를 들면, 새로 정보기술을 배우는 사람은 변화시켜야 할 패러다임이 거의 없으므로 패러다임의 변화가 큰 이슈가 될 수 없다. 구조적 기법이나 정보공학 방법론을 적용하고 있는 조직은 패러다임의 변화가 필요하지만, 개발 방법론에 대한 기술을 보유하고 있지 않은 조직보다는 패러다임을 쉽게 변화시킬 수 있을 것이다.

그러나 스파게티 프로그램을 개발하고, 소프트웨어 공학에 따른 체계적인 개발방법론(구조적 기법이나 정보공학)을 적용하고 있지 않는 조직은 OOT 도입으로 인한 패러다임의 변화보다는, 개발 방법론 등의 인프라스트럭처의 정착을 위한 패러다임의 변화가 더 어려운 과제가 될 것이다.

애플리케이션 개발 프로젝트에서 OOT이 중요한 요소일 수는 있지만 전부는 아니다. 따라서, OOT의 도입으로 인한 패러다임의 변화보다는 프로젝트에서 필요로 하는 개발 경험, 비즈니스에 대한 지식/경험, 사용자와의 커뮤니케이션 능력 등의 요소가 더욱 중요하다.

OODB를 적용하지 않으면 객체지향이 아니다

객체를 있는 그대로 저장하고, 꺼내 쓰는 가장 바람직한 DBMS는 객체지향 데이터베이스(OODB)이기는 하나, 현실적으로 성능, 사용자의 수 등의 제약때문에 대부분의 객체기술 프로젝트에서 관계형 데이터베이스가 주로 쓰이고 있다.

OODB는 멀티미디어 정보나 데이터의 구조가 복잡할 경우에 적합한데, 인터네트가 각광을 받으면서 인터네트/인트라네트에 적합한 DBMS로서 주목을 받고 있으며, 앞으로 점차 사용이 확대될 것이다.

그러나, 당분간은 OOT를 적용하는 프로젝트에서 관계형 데이터베이스(RDB)가 주로 쓰이면서, IBM, ORACLE 등 주요 RDB 공급자들이 발표한 객체관계형 데이터베이스(ORDB)에 대한 사용도 증가할 것이다. DBMS(RDB, ORDB, OODB간)의 선정은 개발하는 애플리케이션의 성격에 따라 결정되어야 하며, 모든 애플리케이션 환경에 맞는 하나의 DBMS는 존재하지 않을 것이다.

OOT는 아직 성숙하지 않았기 때문에 몇 년 후에나 적용할 수 있는 기술이다

OOT는 계속 진화하여 왔으며, 앞으로도 계속 진화할 것이다. 이러한 특성으로 인하여 OOT가 성숙하지 않았다는 오해가 생기게 되었다. 그러나, OOT는 구조적 방법론 또는 정보공학의 문제점을 극복하면서 발전되어 왔기 때문에, 실 프로젝트 적용에 있어서 기존의 기술(구조적 기법/정보공학)에 비교하여 이론적으로 우위에 있으며, 구현하고자 하는 업무를 보다 자연스럽게 표현할 수 있게 함으로써 적용을 용이하게 하여 준다.

새로운 기술의 도입이 그러하듯이 OOT를 도입할 때 이상 'State of Art'만을 추구해서는 안되고, 항상 현실 'State of Market'을 고려하는 것이 바람직하다. 즉, 현실을 위해서는 일정 부분 OOT의

적용을 포기하고, 가능한 다른 솔루션을 혼용한다면, OOT가 미성숙한 부분이 있더라도 이 부분을 충분히 보완할 수 있을 것이다.

OOT는 정보기술 경력자보다 경험이 없는 사람에게 유리하다

단순하게 OOT만을 고려하면, 정보기술에 대한 경험이 없는 사람이 OOT를 보다 쉽게 이해할 수도 있다. 그러나 정보기술에 대한 경험이 없는 사람이 설사 OOT를 쉽고, 빠르게 받아들인다 하더라도, 비즈니스에 대한 지식/경험 및 애플리케이션 개발 경험 부족으로 실제 적용은 어려울 것이다.

경험 유무보다는 개인이 새로운 것을 얼마나 편견없이 적극적으로 열린 마음으로 받아들이느냐가 더 중요할 것이다. 프로젝트의 성공을 위해서는 정보기술에 대한 경험이 충분한 전문가와 경험이 없는 인원이 혼합된 팀의 구성이 바람직하다.

OOT는 매우 어렵다

OOT는 정보기술과 함께 발달되어 왔다. 따라서, 기존의 정보기술에 대한 인프라스트럭처(예, 윈도우 프로그래밍, UNIX, RDB 등)가 갖추어져 있어야만 OOT가 성공적으로 도입될 수 있다. 그러하지 못한 조직이 OOT를 도입할 경우 인프라스트럭처에 관련된 기술을 한꺼번에 도입하는 것이 불가피하므로 OOT의 도입이 상대적으로 어려울 수 밖에 없다.

OOT는 요술 방망이가 아니며 다양한 정보기술의 일부분을 차지할 뿐이다. 따라서 프로젝트에서 OOT 이외의 기술이 추가로 요구될 경우에는 이러한 기술도입에 대한 투자와 노력을 추가로 고려하여야 한다.

UML은 방법론이다, "방법론 전쟁은 끝났다"

OO방법론은 럼버의 OMT, 야곱슨의 OOSE, 부치, HP의 퓨전(Fusion), 마틴-오델의 OOIE, 쉴러-멜러, 코오드-요든, 워프-브록(Wirfs-Brock)을 비롯하여 40~50여종이 상호보완 관계를 유지하면서 발전되어 왔다. 이와 같이 수많은 방법론이 사용됨에 따라서, OOT를 도입하려는 회사들이 방법론을 선택하는 것을 어렵게 하였으며, OOT이 어렵다는 오해를 불러오는 한 요인이 되기도 하였다.

이러한 문제점을 극복하기 위하여 94년부터 OO 방법론의 통합작업이 미국의 래쇼날 소프트웨어(Rational Software)를 중심으로 이루어져 UML(Unified Modeling Language)이 개발, 97년 9월초에 버전 1.1이 발표되었다.

이 작업은 래쇼날 소프트웨어에 근무하고 있는 3명의 OO방법론 전문가(Three Amigos)인 그래디 부치, OMT를 개발한 제임스 럼버, 그리고 사용예(Use case)를 개발한 아이바 야곱슨(Ivar Jacobson)이 주축이 되어 IBM, 마이크로소프트, 오라클 등과 함께 개발되었으며, 객체관리그룹(OMG:Object Management Group)의 OOA/D 표준으로 97년 9월초에 제출되어 현재 심사를 진행하고 있다. UML은 사실상 산업계의 표준으로 받아들여지고 있으며, OMG의 표준으로 채택이 확실시되고 있다.

방법론은 1)모델링에서 사용되는 다이어그램(Notation)과 이 다이어그램에 사용되는 문법(Semantics), 2)프로세스로 구성된다.

UML은 Notation과 Semantics만을 정의하고 있으므로 모델링을 위한 언어이며, 프로세스는 정의하고 있지 않다. UML에서 프로세스는 조직, 문화, 업무영역별로 차별화되어야 하므로 표준화된 프로세스를 제안하는 대신, 단지 Use-case중심이고, 아키텍쳐 중심이며, 반복적이며, 점진적인 프로세스(A use-case driven, architecture-centric, iterative, and incremental process)를 사용할 것을 추천하고 있다.

따라서 UML은 방법론의 중요한 부분인 프로세스가 없기 때문에 방법론이 아니다. 그리고, Notation과 Semantics은 통합할 것이 확실시 되고 있으나 프로세스가 없으므로 방법론에 대한 전쟁이 끝난 것이 아니고, 프로세스 및 프로세스에 관련된 기법에 관련된 방법론 개발에 대한 경쟁이 이제부터 시작될 것이다.

UML을 지원하는 프로세스는 객체지향 CASE를 개발하는 회사나 방법론에 관한 자문을 하는 회사들에 의하여 개발되고 있는데, UML을 주도적으로 개발한 래쇼날 소프트웨어의 Objectory 프로세스가

될 주요한 프로세스중 하나가 될 것으로 보이며, 금년 중으로 발표될 것으로 예상된다.

DCOM이 CORBA를 대체할 것이다

주로 DCOM을 선호하는 전문가에 의하여 잡지에 가끔 언급되는 소수 의견이다. DCOM과 CORBA는 원격지에 있는 객체간에 메시지를 주고 받는 솔루션인데 아직까지 널리 사용되고 있지는 않다. CORBA는 이기종간에 위치하는 객체간에 메시지를 주고 받는 솔루션으로, DCOM은 NT나 윈도우즈95환경에서의 솔루션으로 받아들여지고 있다.

일반적으로 CORBA가 DCOM에 비해서는 많이 사용되고 있으며, DCOM은 제한된 영역에서 사용되기 시작하였다. DCOM은 지속적으로 CORBA와 함께 사용될 것이나 CORBA 중심으로 재편될 것이라는 의견이 보다 다수의 의견이다.

이상의 OOT에 대한 이해를 토대로 OOT를 도입하기로 결정되었다면, 적절한 교육과 자문을 통한 시행착오를 최소화하여야 한다.

적절한 교육과 자문이 필수적

OOT는 업무의 분석, 설계, 구현 방법에서의 변화를 요구한다. 따라서 OOT를 성공적으로 도입하기 위해서는 애플리케이션 개발에 관계되는 모든 사람 즉, 관리자, 시스템 분석가, 설계가 및 개발자들을 대상으로 실습(Workshop)을 포함한 적절한 교육이 필요하다. 교육 내용은 교육 참가자의 요구 및 숙련도(프로그래밍 언어, 분석방법, 설계기법 등)에 따라서 탄력적으로 조정(Customization)되어야 하고, 기존의 전통적인 개발방법을 잊게 하는 교육이 함께 병행되어야 한다(Training with untraining). 아울러, 실제 적용경험(해당 업무분야에 대한 경험도 고려되어야 함)이 있는 전문가(Mentor)의 지원하에 프로젝트를 수행하는 것이 바람직할 것이다.

OOT의 숙달을 위해서는 상당한 기간이 필요한데, IBM의 경우 기술적으로 충분한 경험을 갖고 있는 전문가가 충분한 교육과 자문하에서 완전한 생산성을 얻는데 6개월에서 8개월이 소요되었고, 유럽의 시스템 통합 사업자인 데이터 사이언스사의 경우는 전통적인 개발방법에 익숙한 전문가가 효과적인 객체지향 설계자로 전환되는데 6개월에서 12개월이 소요되었고, 교육 후 3개월 정도의 세심한 자문(Mentoring)이 필요하였다고 보고되었다.

데이터 사이언스사의 경우 전통적인 개발자의 절반은 쉽게 객체지향으로 전환하였고 나머지 25%는 결국 객체지향으로 성공적으로 전환하였다. 그러나 나머지 25%는 객체지향 개념을 이해하는데 실패하였다. 객체지향에 대한 열정만으로는 충분하지 않으며 객체지향으로 전환하는데 실패한 팀원은 조기에 발견하여 프로젝트에서 격리시키는 것이 매우 중요하다. OOT 도입에 있어서 조직의 성숙도를 고려한 도입전략이 수립되지 않으면 전문가들에게 좌절과 함께 낮은 생산성을 초래하여, 프로젝트는 실패할 것이다.

맺는 말

OOT이 성숙되었는가, 아닌가에 대한 논쟁과 상관없이 OOT는 이미 정보기술의 중요한 인프라로서 자리잡아가고 있다. 현재 주로 이용되고 있는 대부분의 4GL(포르테, 델파이, 비주얼 베이직, 파워빌더), 그리고 주요한 프로그래밍 언어(C++, 자바)가 객체지향 패러다임을 지원하고 있다. 따라서, 이러한 OOPL이나 OO 4GL을 사용할 경우에는 객체지향 패러다임 하에서 애플리케이션을 개발하는 것이 훨씬 효율적이고, 바람직하다.

OOT의 성공적인 도입을 위해서는 객체기술을 정확하게 이해하고, 기업의 실정에 맞는 적절한 전략(기업문화에 맞는 기술선택, 팀 구성 등)이 수립되어야 한다. OOT로부터 얻을 수 있는 효과는 공짜로 얻어지는 것이 아니고, 적절한 노력과 투자를 통해서만 얻어질 수 있다. 이제는 OOT를 도입할 것인가에 대해 망설일 때가 아니고, 직접 경험하면서 도입을 서둘러야 할 시기가 아닐까 생각된다.

참고문헌:

1. Gartner Group - Object Technology: A Manager's View외 다수(1995)
2. Communication of the ACM - The Promise and the Cost of Object Technology: A Five-Year Forecast(October 1995)
3. Journal of Object-Oriented Programming - The Need for Flexibility in Object-Oriented Training(March-April 1994)
4. UML 1.1 - Rational Software(1997/9)

OT 기사 97-10-21 11:02 AM