Service Oriented Architecture와 재사용

특집 04

최 은 만
(동국대학교)

목 차

1. 서 론

소프트웨어 엔지니어들은 소프트웨어 IC (integrated circuit)라는 꿈을 향한 긴 여정을 달리고 있다. 새로운 소프트웨어 개발 기술이 출현할 때마다 재사용 가능한 컴포넌트를 이용하여 조립 개발할 수 있고 변경이 많은 요구에 대하여 쉽게 적응할 수 있는 약속을 펴 보였다. 간단한 함수의 라이브러리를 시작으로, 파서 또는 애플리케이션 생성기, 객체 중심 프로그래밍, 애플리케이션 프레임워크, 미들웨어를 통한 분산 컴포넌트, 도메인 엔지니어링, 프로덕트 라인 등이 소개되었다. 모두 재사용을 목표로 하고 있지만 접근 방법이나 재사용 수준에 큰 차이를 보이고 있다.

최근 소프트웨어 재사용을 서비스 재사용과 연결하려는 시도가 있다. 비즈니스 환경의 변화에 따라 애플리케이션이 서비스 중심으로 재편되고 있어 서비스 인터페이스를 정의하여 이를 토대로 신속하게 소프트웨어를 구축하려는 방법이다. 즉 SOA(Service Oriented Architecture)는 단순히 코드의 재사용에 초점을 두지 않고 그런 기능을 제공하는 서비스의 실행을 이용하자는 개념이다. SOA는 웹 서비스 기술을 기반으로 실현되고 있는데 대부분의 재사용 기술은 재사용 코드를 새 시스템 안에 포함하는 밀접한 관계를 가지고 있으나 웹 서비스는 원격에서 실행되는 서비스가 느슨한 관계를 유지하고 있어 재사용을 용이하게 한다.

이 논문에서는 상당히 긴 여정 동안의 재사용을 위한 기술 발전 추이를 살펴보고 재사용의 원리를 소개한 후 최근에 핫이슈가 되고 있는 서비스 단위의 재사용 방법에 대하여 자세히 알아본다. 또한 SOA를 재사용이라는 관점으로 조명해 보고 재사용의 방해요소들과 이를 타개하기 위한 방향에 대하여 소개한다.

2. 소프트웨어 재사용 원리

소프트웨어 재사용은 하드웨어 IC라는 꿈을 가지고 시작하였지만 하드웨어 재사용과는 근본적으로 차이가 있다. 하드웨어 재사용은 동일한

프로덕트를 만들기 위하여 동일한 도구를 반복하여 사용한다. 예를 들어 건축에 사용되는 재사용 컴포넌트인 못이나 문짝. 하드웨어 IC 등은 계속 같은 부품이 반복적으로 사용된다. 한편 소프트웨어 재사용은 처음부터 무에서 시작하는 것이 아니라 현재 존재하는 것을 기반으로 발전적으로 사용된다.

2.1 재사용의 목표

소프트웨어를 재사용 하는 목적은 두 가지이다. 첫째는 비용 절감이다. 소프트웨어 문제 자체가 복잡해지고 사용 기술이 다양해지면서 소프트웨어 개발에 드는 노력과 비용은 계속 늘어간다. 개발 과정에 단순한 프로그래밍만이 아니라 설계, 테스팅 등 다양한 작업이 필요하고 대규모 소프트웨어를 개발할 때는 코딩 이외의 작업에 많은 노력이 든다.

두 번째 목적은 버그를 줄이는 것이다. 새로 작성하는 프로그램보다 잘 실행되고 있는 증명된 프로그램을 이용하려는 것이다. 결국 소프트웨어의 목표는 처음부터 새로 만들어내는 소프트웨어 제작 관행을 리사이클링 방식으로 바꾸어 소프트웨어 생산 비용을 줄이려는 노력이다.

2.2 재사용의 요소

재사용 원리를 찾아내려면 우선 어떤 문제들이 재사용을 어렵게 하고 있는지를 살펴보아야 한다. Prieto-Diaz에 의하면 다음과 같은 문제들이 재사용을 방해하는 요소이며 좋은 재사용 방법이 되려면 이들 문제를 잘 해결해 주어야 한다고 한다[1].

- 재사용 단위를 식별하는 일
- 재사용에 관련된 지식을 잘 저장하는 일
- 목적에 맞는 재사용 대상을 찾아내는 일
- 재사용 대상을 새로운 환경에 맞도록 수정하는 일
- 재사용 대상을 프로젝트에 맞게 통합하는 일

처음 두 가지 요소는 재사용 대상을 미리 준비할 때(build for reuse) 필요한 작업이며 세 번째 요소부터 마지막까지는 재사용 대상을 이용하여 시스템을 구성하는(build with reuse) 작업들이다. 결국 위에 설명한 다섯 가지 요소를 잘 제공하여야 좋은 재사용 방법이라 할 수 있다. Krueger는 이를 좋은 재사용의 다섯 가지 차원으로 <표 1>과 같이 제시하였다[2].

<표 1> 좋은 재사용 방법의 다섯 가지 차원

항목	차원	설명
Build for reuse	추상성 (abstraction)	재사용 대상의 단위를 식별하고 이를 추상적인 형태로 정확히 표현하는 차원
	분류 (classification)	재사용 대상을 지식 베이스에 저장하고 잘 분류 인덱스화 하는 차원
Build with reuse	선택 (selection)	재사용 대상을 파라미터를 가진 함수 형태로 질의하는 차원
	커스텀화 (specialization)	재사용 대상을 파라미터의 값을 지정하듯 수정하여 커스텀화 할 수 있는 차원
	통합 (integration)	재사용 대상을 새로운 프로젝트에 잘 통합하는 차원

3. 전통적인 재사용 방법들

소프트웨어 재사용 기술은 전산학의 상당히 광범위한 분야의 기술과 연결되어 있다. 프로그래밍 언어의 설계에 재사용 아이디어가 포함되어 있고 설계에 관한 이론 중 디자인 패턴이나 소프트웨어 아키텍처에서도 재사용이 중요한 관건이 된다. 또한 반복되는 컴파일러 구성 작업을 덜어준 생성 기반의 도구도 재사용 기법의 일종이다. 소프트웨어 기술 발전과 함께 출현한 다양한 재사용 사례를 정리하면 <표 2>와 같다.

이제까지 출현한 재사용 기술들은 두 가지 측면에서 분류할 수 있다. 첫째는 재사용 대상이 무엇인가가 분류 기준이 된다. 원시코드를 재사용하는 경우가 대부분이나 바이너리 코드를 직접 재사용하는 기법도 있으며 미시적인 원시코드의 재사용보다는 거시적인 도메인 지식이나 설계 또는 프레임워크와 아키텍처를 다시 이용하는

〈표 2〉 전통적인 재사용 기술 사례

재사용 기술	사례	특징
High-level language	Java, SQL	- 추상수준이 높아지면서 재사용 효과 가짐
함수 라이브러리	Math.lib	- 함수 단위의 재사용
파서 생성기 및 애플리케이션 생성기	YACC, JavaCC, ANTLR, automake, Eclipse	- 컴파일러 생성 분야의 재사용 탁월 - 최근 도메인 특수 언어, generative programming으로 발전
메뉴/테이블기반 재사용	GUI Widgets	- GUI 생성에 재사용
애플리케이션 프레임워크	Smalltalk, Motif, Swing, AWT	- 객체지향의 상속, 위임에 의한 재사용
도큐멘트 생성기	JavaDoc/XDoclet, DocBook, LaTeX, CSS, RSS, XSLT	- 프로그램을 다른 프로그램에 의하여 처리되는 자료로 간주
4-세대 언어	SQL, Wizards, templates, MIL/ADL	- 높은 추상성에 의한 재사용
Plug-in, Skins, Themes, Macros, Extensions	Eclips, Word, WinAmp	- 바이너리 코드의 재사용
도메인 엔지니어링 애플리케이션 생성	SAP	- 유사한 응용분야의 프로덕트 패밀리에 대한 재사용
도메인 중심 언어	Draco, TXL	- 도메인의 상세화
미들웨어 컴포넌트	CORBA, EJB, .NET	- 분리 배치할 수 있는 실행 가능 바이너리 컴포넌트
웹 서비스	WSDL, UDDI	- 낮은 결합력을 가진 컴포넌트들의 합성

기법이 최근에 많이 이용되어 왔다.

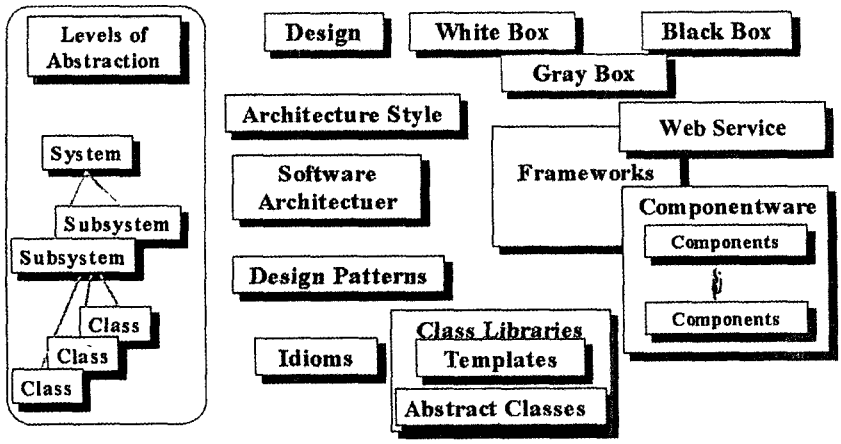
둘째는 재사용 대상을 합성하여 시스템을 구성해 나가는 빌딩 블록 접근 방법과 개발 과정의 일부를 자동화 하여 더 높은 추상 형태로부터 코드를 자동 생성하는 생성 접근 방법이 있다. 재사용 가능한 컴포넌트를 만들고 이를 이용하여 빌딩 블록을 구성해 나가는 방법은 CORBA, EJB, .NET과 같은 분산 미들웨어 컴포넌트로 발전되었다. 특히 이들 미들웨어 컴포넌트는 실행 가능 바이너리 컴포넌트로 인터페이스가 잘 정의되어 있고 컴포넌트를 기본 빌딩 블록으로 복잡한 시스템으로 발전시킬 수 있으며 분리 배치할 수 있는 특성을 가지고 있다.

4. 소프트웨어 재사용을 위한 새로운 시도

앞서 언급한 것처럼 재사용이 활발히 이루어지려면 다섯 가지 측면, 추상성, 분류, 선택, 커스텀화, 통합이 용이하여야 한다. 재사용 대상을 간단히 표현하여 다시 사용하려는 입장에서 이해가 쉬워야 하며 잘 분류 저장되어 선택이 용이하여야 하며 새로운 환경에 맞도록 쉽게 바꿀 수 있어야 하고 새로운 애플리케이션 안으로 통합할 수 있어야 한다.

이러한 관점에서 이제까지 많이 사용되어 온 방법을 고찰해보자. 재사용 함수는 간단한 인터페이스로 정의되어 추상성이 뛰어나지만 분류 및 선택 방법이 제공되지 않아 재사용 하여 개발하는 입장에서 불편함이 있다. 꼭 맞는 인터페이스와 기능이 아니면 재사용이 어려워 커스텀화에 대한 지원은 전혀 없다. 무엇보다 함수 재사용의 단점은 함수가 데이터와 분리되어 있고 데이터가 단일 네이밍 스페이스에 완전히 오픈 되어 쉽게 변경될 수 있다는 점이다. 따라서 프로그램의 규모가 커지면 매우 많은 함수가 자료에 커플링되어 다시 사용하기가 어렵게 된다.

객체지향 방법은 자료와 함수가 캡슐화 된 객체가 재사용 단위가 된다. 객체 재사용의 장점은 클래스나 패키지 등과 같은 네임 스페이스를 정할 수 있다는 점이다. 따라서 데이터의 접근을 제한 할 수 있고 함수와 데이터의 커플링이 로컬화 된다. 또한 인터페이스와 구현이 분리되어 추상성이 높고 서브클래스와 함수 오버로딩으로 커



(그림 1) 재사용 단위와 추상 수준

스템화와 통합에 도움이 된다.

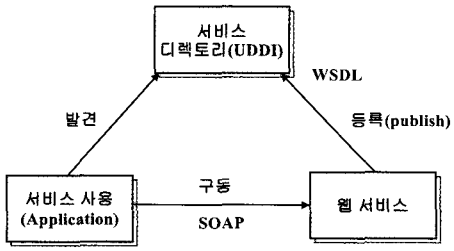
객체지향은 원시코드만이 아니라 설계 패턴이 정립되면서 재사용 측면에서 상당한 효과를 보여 왔다. 그러나 클래스라는 단위가 재사용이라는 측면에서 하드웨어 IC의 규모는 되지 못한다. 즉 Account라는 클래스 단위의 재사용 보다는 Account, Profile, Address, 등이 모두 포함되어 있는 Customer라는 컴포넌트 단위의 재사용이 더 효과적이다. 즉 그림 1에 나타낸 것처럼 추상의 단위가 커질수록 재사용의 효과가 더 크다. 컴포넌트는 분산 환경에서 보다 완전한 재사용 단위가 되기 위하여 자원, 트랜잭션, 배치 서술부(deployment descriptor) 등이 확장되어 사용되어 왔다. 컴포넌트는 내부 구현을 감추어 재사용할 수 있는 블랙박스 형태이며 재사용을 위한 좋은 인터페이스를 가지고 있다. 구현이 변경되더라도 인터페이스가 유지된다면 재사용 클라이언트 측에는 영향이 없다. 반면에 재사용 컴포넌트를 잘 분류하고 선택, 통합할 수 있어야 한다는 측면에서는 부족한 면이 있다. 시스템의 규모가 계속 커지고 비즈니스 시스템이 다양화, 세분화되면서 객체지향을 기초로 한 분산 컴포넌트는 다이나믹한 시장요구를 충족시키는 데 한계를 드러내고 있다.

이질적인 환경의 복잡한 시스템을 웹 환경에서 통합하는 문제를 해결하는 패러다임 킬러 기술인 웹 서비스가 이러한 문제의 열쇠로 등장하였다. 웹 서비스의 컴포넌트는 WSDL로 잘 정의되어 있고 윈도우 레지스트리에 등록된 프로그램처럼 UDDI에 등록되며 XML과 SOAP에 의하여 통신할 수 있다. 웹 서비스에서 유래된 SOA는 소프트웨어의 재사용에서 차원을 높여 서비스의 재사용이라는 접근 방식이다. 즉 Customer 컴포넌트를 재사용하는 대신에 고객을 생성, 변경, 삭제, 로그인, 취향 탐색 등의 기능을 제공해 주는 Customer 서비스를 재사용 하는 것이다. 웹 서비스 기술은 분산 컴포넌트처럼 블랙박스 형태이나 재사용 컴포넌트가 강하게 결합되지 않아 통합하기 쉽고 복잡한 배치와 선택을 일일이 신경 쓰지 않아도 재사용 할만 한 것인지 판단하여 쉽게 사용할 수 있다.

5. SOA

SOA의 빌딩 블록은 서비스이다. 서비스는 정해진 작업, 예를 들면 고객의 신용 정보를 검증하는 작업을 수행하는 독립된(self-contained) 소프트웨어 모듈이다. 이러한 서비스 컴포넌트를 결합하되 분산 컴포넌트 방법보다는 약한 결합력

으로, 언어 및 플랫폼에 관계없이 소프트웨어를 구성할 수 있는 아키텍처 스타일이 SOA이다. 중요한 특징 중의 하나는 그림 2와 같이 서비스가 잘 정의된 인터페이스로 등록되어 발표된다는 (publish) 점이다. 또한 서비스를 사용하는 측에서 원하는 서비스를 찾아낼 수 있고 찾은 서비스들을 복합적으로 구성하여 구동하게 함으로써 전체 시스템을 구성할 수 있다.



(그림 2) SOA

분산 컴포넌트의 경우 객체나 컴포넌트의 로직이 애플리케이션 안에 너무 강력한 결합으로 통합되어 있는데 반하여 SOA는 특정 기술이나 컴포넌트의 내부 로직에 관여하지 않고 인터페이스만을 이해하면 재사용할 수 있다. 예를 들어 고객이 신용카드로 지불하는 기능을 갖추려면 먼저 이런 서비스를 만들고 웹 서비스 컴포넌트로 등록한 후에는 여러 애플리케이션에서 이 서비스를 동적으로 사용할 수 있다.

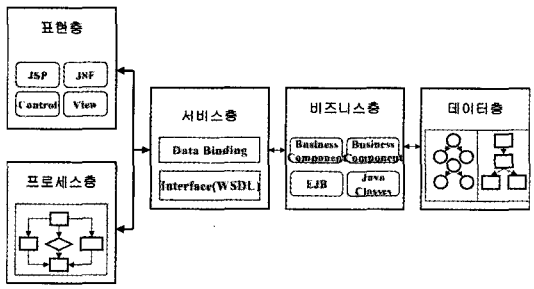
5.1 SOA를 가능하게 한 기술

SOA를 가능하게 한 기술은 무엇보다 웹 서비스이다. SOA와 웹 서비스를 같은 뜻으로 혼용하는 경우가 있는데 정확히 말하면 SOA는 설계이며 웹 서비스는 구현 기술이다. 웹 서비스를 사용하지 않고 Java RMI와 같은 것으로도 SOA를 구축할 수 있다. SOA의 주된 원리는 분산된 적당한 모듈을 찾아내어 모듈 사이에 결합이 낮은 형태로 플랫폼 독립적인 웹 환경에서 통합하는 것이다.

웹 서비스는 HTTP, XML, SOAP, UDDI와 같은 플랫폼 독립적인 표준을 기초로 하고 있다. 따라서 레거시 시스템만이 아니라 J2EE, .NET과 같은 이질적인 기술 사이에 상호운용성을 가져다 주는 장점이 있다.

5.2 SOA를 이루는 층

서비스 중심 애플리케이션도 분산 애플리케이션과 같이 프레젠테이션, 비즈니스 로직, 데이터 모델층 등 여러 층으로 구성된다. (그림 3)에 나타낸 것이 SOA의 구조이다. SOA에서 중요한 층은 서비스 층과 비즈니스 프로세스 층이다.



(그림 3) 서비스 중심 애플리케이션을 이루는 계층

서비스는 SOA의 빌딩 블록이다. 따라서 자바의 객체나 EJB의 컴포넌트와 어느 정도 유사하지만 객체와는 달리 내부 상태가 잘 정의된 하나의 인터페이스 안에 캡슐화 되고 일체화 되어 있다. 따라서 SOA를 구축하는 데 가장 중요하고 어려운 일은 적절한 추상 수준을 가진 인터페이스를 정하는 일이다. 예를 들어 구매 주문을 처리하는 서비스는 적당한 크기와 추상성을 가진 서비스 컴포넌트라 할 수 있다. 컴포넌트와는 달리 주문 처리 서비스 안에는 구매의 속성 변경과 관리에 대한 사항이 포함되어 일체화 되어 있다.

서비스의 가장 중요한 측면은 서비스를 명세화 하는 방법이다. 웹 서비스를 이용한다면 WSDL을 사용하여 웹서비스의 입출력 메시지,

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MyTimeService"
  targetNamespace="urn:oracle-ws" xmlns:tns="urn:oracle-ws"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
<types/>
<message name="TimeService_getDateTime">
<part name="String_1" type="xsd:string"/></message>
<message name="TimeService_getDateTimeResponse">
<part name="result" type="xsd:string"/></message><portType name="TimeService">
<operation name="getDateTime" parameterOrder="String_1">
<input message="tns:TimeService_getDateTime"/>
<output message="tns:TimeService_getDateTimeResponse"/>
</operation></portType><binding name="TimeServiceBinding" type="tns:TimeService">
<operation name="getDateTime">

<input>
<soap:body
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  use="encoded" namespace="urn:oracle-ws"/>
</input>

<output>
<soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded" namespace="urn:oracle-ws"/>
</output>

<soap:operation soapAction=""/></operation>
<soap:binding
  transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
</binding>

<service name="MyTimeService">

<port
  name="TimeServicePort"
  binding="tns:TimeServiceBinding">
  <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
</port>
</service>
</definitions>
```

(그림 4) TimeService 웹 서비스의 WSDL 명세

타입, 오퍼레이션을 정의한다.

SOA는 현재 등록된 서비스로부터 새로운 애플리케이션을 구축할 수 있고 따라서 비즈니스 프로세스 층에 대한 모델링이 중요하다. 레거시 시스템들을 웹 서비스로 만들고 이들 서비스들을 잘 편성하면 효율적으로 재사용하여 시스템을 구축할 수 있기 때문이다. 최근 SOA 플랫폼 벤더들은 비즈니스 프로세스를 설계하고 실행시킬 수 있는 도구를 출시하고 있는데 BPEL (Business Process Execution Language)이라는 표준을 제공하고 있다. BPEL 정의에는 프로세스의 파트너에 대한 링크, 조작하는 변수, 관련되는 메시지, 오류 처리, 문제 발생시 보충, 이벤트 처리 등에 대한 사항을 XML 형태로 기술한다.

표현층은 사용자 입장에서 매우 중요한 부분이다. JSP, JSF, Java 클라이언트 등으로 구축되며 Model-View-Controller 프레임워크와 같이 다른 층에 대하여 느슨한 결합을 가져야 한다. 표현층의 문제점은 JSP, Java 클라이언트, EJB, 웹 서비스 등 서로 다른 클라이언트 사이에 데이터 교환을 위한 바인딩이 표준화 되어 있지 않다는 점이다. 예를 들어 웹 서비스나 EJB를 사용하여 구현한 서비스들이 데이터를 교환하는 인터페이

스가 각기 다르다.

비즈니스층을 구성하는 서비스 컴포넌트는 레거시 컴포넌트를 포함하여 EJB, Java 클래스 등 여러 가지 방법으로 구성할 수 있다. 이것이 바로 SOA에서 재사용되는 단위이다. 비즈니스 로직을 수행하는 Stateless 세션 EJB나 Java 클래스를 웹 서비스 컴포넌트로 이용하려면 WDSL로 정의하는 것 이외에도 Java-to-WDSL 매핑을 자세히 설명하는 XML 파일과 웹 서비스 배치 디스크립터, 벤더 고유의 배치 디스크립터가 필요하다.

5.3 SOA의 도전과제

이제까지 언급한 것과 같이 SOA는 비즈니스 애플리케이션을 구성하기 위하여 제삼의 기관에서 제공한 재사용 서비스들을 통합한 것이라 할 수 있다. 통합을 위하여 범용적인 프로토콜을 사용하고 느슨한 결합을 가짐으로 빠른 개발을 촉진하였다할 수 있다. 그러나 SOA 기술의 수용을 빨리 이루려면 다음과 같은 도전과제를 해결하여야 한다.

재사용 서비스들은 대부분 외부의 조직에 의하여 만들어진 컴포넌트들이며 애플리케이션의 운용과 관리라는 입장에서 취약한 링크를 가지고 있다. 제3의 기관에서 제공하고 운용하는 웹서비스에 대하여 신뢰도나 서비스의 질(QoS), 성능에 대한 보증이 필요하다. 또한 애플리케이션과 협력하는 트레이딩 파트너의 웹서비스는 특별한 변경 없이 계속 사용될 수 있는 축소확장성(scalability)이 있어야 하고 다양한 사용을 위하여 애플리케이션에 영향을 주지 않고 커스텀화 수 있어야 하며 유지보수 할 수 있어야 한다[9].

결국 SOA는 재사용하는 서비스가 계속 변경되는 환경에 있어 무빙 타켓을 잘 관리하고 제어하는 것이 관건이다. 이런 문제를 위하여 웹서비스의 동적 스워핑 기술과 브로커 아키텍처에 대한 연구가 이어지고 있다.

6. 결 론

이제까지 걸어온 소프트웨어 재사용을 위한 여정을 뒤 돌아보면 재사용 단위가 커진다는 추세와 재사용 컴포넌트를 결합하는 방법이 더욱 자유롭고 간편하게 이루어지는 방향으로 발전하고 있다는 것을 알 수 있다. 최근 비즈니스 환경의 변화에 따라 서비스 단위로 업무가 재편되면서 웹서비스로 컴포넌트화 되는 SOA 기술은 재사용에 효과적이라 할 수 있다. 특정 애플리케이션에 사용되기 위한 코드를 일반화하기는 쉽지 않았지만 웹서비스는 특정 애플리케이션의 요구라는 차원을 넘어 범용성을 고려한 것이므로 재사용이 자연스럽다. 그러나 SOA는 재사용이 느슨한 결합을 이루고 있어 전체 애플리케이션의 품질 저하의 원인이 될 수 있다. 이를 해결하기 위한 여러 가지 도전과제들을 잘 해결하고 품질 기반의 재사용 엔지니어링이 확립된다면 애플리케이션 개발을 위한 재사용 핵심 기술이 될 전망이다.

참고문헌

[1] R. Prito-Diaz, "Status report: software reusability", IEEE Software, 10(3), pp.61-66, 1993.

[2] C. W. Kreuger, "Software Reuse", ACM Computing Survey, pp.131-183, 1992.

[3] Hafedh Mili, Fatma Mili, and Ali Mili, "Reusing software: Issue and research directions", IEEE Trans. on Software Engineering, Vol. 21, No. 6, pp.528-562, June 1995.

[4] William B. Frakes and Kyo Kang, "Software reuse research: Status and future", IEEE Trans. on Software

Engineering, Vol. 31, No. 7, pp.529-536, July 2005.

[5] Mahesh H. Dodani, "From Object to Services: A journey in search of component reuse nirvana" Journal of Object Technology, Vol. 3, No. 8, September-October, pp.49-54, 2004.

[6] Rich Rogers, Reuse engineering for SOA, http://www-123.ibm.com/developerworks/webservices/library/ws-reuse/soa.html, 2004.

[7] Thomas Erl, Service-Oriented Architecture (SOA): concept, technology, and design, Prentice Hall, 2005.

[8] Debu Panda, An introduction to service-oriented architecture from a java developer perspective, http://www.onjava.com/pub/onjava/2005/01/26/soa-intro.html, 2005.

[9] Sandeep Chatterjee, James Webber, Developing enterprise web services, Prentice Hall, 2004.

[10] B. W. Frakes and S. Isoda, "Success factors of systematic reuse", IEEE Software, 11(9), pp.15-19, 1994.

[11] D. Sprott, R. Veryard, Component-based Development for the .Net Environment. CBDI Forum Report, http://www.cbdiforum.com, December 2000 .

[12] do Prado Leite, J. C. S. Yu, Y. Liu, L, Yu, E. S. K. Mylopouslos "Quality based software reuse", Lecture Note in Computer Science, Vol. 3520, pp.525-550, 2005.

저자약력

최 은 만

1982년 동국대학교 전산학과(학사)
1985년 한국과학기술원 전산학과(공학석사)
1993년 일리노이 공대 전산학과(공학박사)
1985년~1988년 한국표준연구소 연구원
1988년~1989년 데이콤 주임연구원
1998년~2004년 한국정보과학회 소프트웨어공학연구회 운영위원
2000년 콜로라도 주립대 전산학과 방문교수
2002년 카네기멜론대학 소프트웨어공학 과정 연수
1993년- 현재 동국대학교 컴퓨터멀티미디어공학과 교수
관심분야 : 객체지향 설계, 소프트웨어 테스트, 프로세스와 메트릭, Program Comprehension
이 메 일 : emchoi@dgu.ac.kr