소프트웨어 컴포넌트 용어 표준(Software Component Terminology Standard)

1. 개요

1.1 목적

본 표준은 컴포넌트 기반 소프트웨어 개발(component based software development) 에서 사용되고 있는 용어들에 대한 일관된 한글 표기법과 그 의미를 정의하기 위한 것이다. 본 표준은 본 표준과 관련된 방법론 및 기술을 활용하여 컴포넌트를 교육, 개발, 자문하 는 기관에서 일관된 용어의 사용으로 관련자들간의 원활한 의견 교환을 가능케 하기 위한 것이며 본 용어 정의는 컴포넌트 기반 기술이나 방법론을 제안한 공식 기관의 원문을 바탕 으로 우리나라 개발자들의 이해를 증진하기 위한 목적으로 이루어졌다. 또한 동일한 용어에 대해서 상이해 보이는 정의 같지만 실질적으로는 본질적인 의미는 동일함을 인식해야 한다. 그러므로 용어가 사용되는 상황을 무시하고 하나의 용어만을 표준으로 채택하는 것을 목적 으로 하지 않는다.

1.2 범위

본 표준에서 사용하고 있는 용어는 현재 국내 산.학.연을 통해 가장 널리 퍼져 있는 컴 포넌트 기반 개발(CBD:Component Based Development) 방법론과 구현기술을 범위로 하였 다. CBD에 관한 일반 용어 및 CBD 방법론에 관한 용어, 대표적인 구현 기술로 선(Sun)의 EJB, 마이크로소프트의 닷넷(.NET)을 포함하고 있다.

각 용어는 상기 선정 범위에 해당하는 내용에 대해 공식 용어 해설집(Glossary)을 기반 으로 하고 있다. 또한 각 용어는 다음의 원칙에 따라 작성되었다.

- 용어 순서는 영어원어의 알파벳 순으로 배열하였다.

- 표준안 작성은 영어 원어를 기준으로 하여 분류영역, 영어정의, 한글용어, 참고자료 항목으로 기술하였다. 분류 영역은 용어의 정의가 활용될 수 있는 영역을 표시한다.

- 한글 용어는 번역시 원어를 가급적 그대로 유지하며 일반적이고 보편적으로 통용되 는 언어를 사용하였다.

- 한글 용어 표현에 있어서는 원어의 음절길이가 한글번역 용어 음절의 두 배 이상 길 어 사용의 간편성이 요구되는 경우나 한글 용어가 널리 사용되는 용어가 있을 경우 이를 한글화 하였다.

본 용어 정의는 완전한 것이 아니며 향후 새로운 기술, 사용 환경 등을 반영하여 지속 적으로 확장되고 개선되어야 한다.

2. 참고문헌

본 표준에서 참고하고 있는 대표적인 참고문헌들은 다음과 같다.

. UML Guide

. J2EE Tutorial

. MS .NET Library

. CSG : UN/CEFACT Steering Group

주) UN/CEFACT : UnitedNations Center for Trade Facilitation and Electronic Business

3. 용어 정의

각각의 용어를 해당 항목의 정의, 설명 및 주석의 세부분으로 나누어 기술하고, 항목의 정의는 볼드체로, 주석은 이탤릭체로 표기하였다. [ ]안은 해당 항목의 설명 및 주석의 출처를 나타낸다.

3.1 일반 용어

3.1.1 애플리케이션 아키텍처(Application Architecture)

넓게는 관심대상의 영역/응용프로그램의 아키텍처를 이르며, 좁게는 특정 영역을 위한 응용 프로그램 생성기를 일컬음.

주. 워크플로우 및 CASE 도구가 양쪽 범주에 다 해당된다.

[CSG]

3.1.2 애플리케이션 조립자(Application Assembler)

컴포넌트와 모듈들을 결합하여 전개/배치 가능한 응용 프로그램의 단위로 만드는 사람

[J2EE Tutorial]

3.1.3아키텍처(Architecture)

1) 시스템을 구성하는 소프트웨어 시스템의 조직, 구조적 요소의 선정 및 이들의 인터페이스 등에 관한 중요한 결정 사항들의 집합 ([UML Guide])

2) 소프트웨어적 관점에서의 아키텍처 : 소프트웨어 시스템의 형태와, 아키텍처상에 있는 하위 시스템 또는 컴포넌트 부품들을 조립하여 소프트웨어 시스템을 만드는데 필요한 협약, 정책 및 메커니즘을 제공해주는 정적 프레임워크 또는 골격(구조 또는 협약의 집합)을 말함 ([CSG])

주1) 아키텍처에는 요소간 협력 관계에 명시한 행동, 이들 구조적 요소와 행동 요소들의 점 차적으로 더 큰 서브 시스템으로의 조립, 이러한 조직-이들 요소와 그들의 인터페이스-의 지침이 되는 아키텍처 스타일, 이들의 협력 관계 및 구성 등에 대한 내용이 포함된다.

주2) 소프트웨어 아키텍처는 구조와 행동에만 국한되지 않고 사용관계(usage), 기능성, 성능, 탄력성, 재사용, 이해도, 경제적 기술적 제약사항 및 이들의 장단점, 그리고 미적 문제까지 관련된다.

주3) 소프트웨어 아키텍처는 구성 요소들간 상호 관계 및 이들 관계를 제어하는 제약 조건을 정의한다. 추상적 프레임워크은 어떤 구체적인 특정 하위 시스템을 명시하지 않은 프레임워크를 말하며, 구체적 프레임워크은 추상적 프레임워크가 실행시에 특정한 요소와의 바인딩이 결 정됨으로써 특정 하위 시스템으로 실체화된 프레임워크를 말한다. 시스템을 부품 (예, 아키텍처 및 이를 구성하는 컴포넌트들)으로 나누면 이들 부품 간의 내부 통신 방식을 정의하는 인터페이스들이 있다. 아키텍처는 단지 하위 시스템들을 특수하게 합성한 것일 수 있으나, 주로 다른 하위 시스템들과 인터페이스 하는 하나의 특정한 하위 시스템일 경우가 대부분이다. 후 자의 경우 아키텍처는 이를 사용하여 구축한 시스템에 대하여 일정한 보증을 유지하는 아키텍처 특성(예, 안전성. 확장성, 결함 허용, 위치에 대한 투명성, ...)을 가질 수 있다. 이러한 아키텍처의 특성들은 가끔 규칙이나 협약(예, C++로 작성되어야 함) 또는 제약 조건(예, C++이나 자바 사용)으로 명시될 수 있다.

3.1.4 아키텍처 중심 (Architecture-centric)

소프트웨어 개발 생명주기 문맥에서 소프트웨어 아키텍처의 초기 개발 및 기준선 잡기에 중 점을 두는 프로세스를 일컬음.

주) 시스템 아키텍처를 주 산출물로 사용하여 개발 대상 시스템을 개념화하고, 구축하며, 관 리하고, 발전시킨다.

[UML Guide]

3.1.5 바인딩 시점(Binding Time)

어떤 결정이 내려져 인스턴스가 생성되는 시간.

주) 소프트웨어에서는 개념에서부터 설계, 코딩, 컴파일, 실행에 이르기까지 바인딩 시점이 달라진다. 정적 바인딩은 컴파일 시에 일어나고, 특정한 유형(type) 정보가 사용되며 때로는 버려지기도 한다. 동적 바인딩은 실행 시에 일어난다.

3.1.6 비즈니스 컴포넌트(Business Component)

반드시 가시적 요소(사용자 인터페이스)를 포함할 필요가 없으나 특정 비즈니스 논리 (Logic)를 내포시킨 컴포넌트.

주) 한 예로, 문자열이나 파일을 받으면 이를 암호화하게 되는 "암호화(encryption)" 컴포넌트를 들 수 있는데 여기에는 가시적 인터페이스가 필요 없다. 사용자 인터페이스 컴포넌트가 반드시 가시적인 요소를 반드시 포함해야 하는 것에 비하여, 비즈니스 컴포넌트는 가시적 요소를 포함할 수도 있다.

[Component Source]

3.1.7 비즈니스 메소드(Business Method)

응용 프로그램의 규칙이나 비즈니스 논리를 구현한 엔터프라이즈 빈의 메소드

[J2EE Tutorial]

3.1.8 비즈니스 객체(Business Object)

회사와 이해 관계가 있는 객체. 예를 들어, 문서, 고객 또는 발주

[CIO]

3.1.9 클라이언트(Client)

다른 분류자에게 서비스를 요청하는 분류자

주) 3.2 모형화 용어의 “분류자(classifier)” 참조

[UML Guide]

3.1.10 클라이언트쪽 컴포넌트(Client-side Component)

응용 프로그램과 같은 기계에서 실행되도록 설계된 컴포넌트

주) 클리이언트쪽 컴포넌트는 가시적(예; 사용자 인터페이스가 있음)일 수도 있고 그렇지 않 을 수도 있다. 클라이언트쪽 컴포넌트는 응용 프로그램과 함께 패키지화 되어 전개/배치되 어야 하며, 응용 프로그램이 설치되어 있는 모든 기계에 설치되어야 한다. [Component Source]

3.1.11 컴포넌트(Component)

1) 컨테이너에 의해 관리되는 응용 프로그램 수준의 소프트웨어 단위 ([J2EE Tutorial])

2) 인터페이스를 제공하여 적응시킬 수 있는, 물리적으로 교체 가능한 시스템의 한 부분 ([UML Guide])

3) 미리 결정된 기능을 자체내에 갖추고 있고, 알려진 인터페이스를 통하여 접근할 수 있는 객체([Component Source])

4) 분해될 수 있고, 잠재적으로 표준화될 수 있거나 재사용 될 수 있는 공개된 인터페이스를 가지고 있는 모든 소프트웨어 (하부)시스템 ([CSG])

주) 1. 컴포넌트는 전개/배치 시점에서 그 형상을 결정할 수 있다. J2EE에서는 엔터프라이즈 빈, 웹 컴포넌트, 애플릿 및 애플리케이션 클라이언트의 4가지 컴포넌트 유형을 정의하고 있다.

주) 2. 컴포넌트의 예로는 ActiveX 컴포넌트, 자바빈즈, 엔터프라이즈 자바빈즈, VCL 등이 있다.

주) 3. SW 아키텍처내의 컴포넌트는 여러 다른 추상화 수준에서 식별될 수 있으며, 이렇게 각기 다른 수준에서 식별된 컴포넌트는 일대일로 대응이 되지 않을 수도 있다. 예를 들어, 추상화 수준 단계에서 아키텍처를 보면 객체 서비스들을 컴포넌트로 식별할 수 있다. 동일 한 아키텍처를 보다 구체적 수준의 단계에서 보면, 하나의 주어진 서비스가 여러 개의 독립된 소프트웨어 모듈로 구현될 수 있는데, 이들 소프트웨어 모듈이 각각 개별 컴포넌트로 식별될 수 있게 된다.

3.1.12 컴포넌트도(Component Diagram)

컴포넌트 집합들의 조직 및 이들간 의존관계를 나타내는 다이어그램

주) 컴포넌트도는 시스템의 정적 구현 뷰를 표현한다. [UML Guide]

3.1.13 합성(Composition)

일반적으로 부품(parts) 자체를 변경하지 않고, 주로 몇가지 구성 방식 또는 규칙을 통해 부 품들을 종합하여 더 큰 제품으로 만드는 것

주 1. 특정한 목적의 합성과 일반적인 목적의 합성이 있을 수 있는데, 이 두 가지 모두, 생 성된 시스템은 컴포넌트 부품(또는 이들의 인터페이스 및 일부 구현과 같은 이들의 최소한 기능)에 의존하게 된다. 이러한 의미에서 합성된 시스템은 이를 구성한 부품보다 상위 수준이라고 할 수 있다. 만약 일반적인 목적의 합성이라면, 상위 수준의 추상화된 시스템은 하위은 수준의 추상화 단위에 의존적이 된다.

주 2. 합성에서 또 다른 중요한 사항은 바인딩 시점으로, 어떤 합성은 정적으로 이루어지며 어떤 합성은 동적으로 이루어질 수도 있다. 원시적인 객체 서비스들로부터 합성될 수 있는 것으로 DBMS, 워크플로우, 지식기반 관리 시스템, 정보 저장소 등이 있다 [CSG]

3.1.14 커넥터(Connector)

전사적 정보 시스템과의 연계성을 위해 컨테이너가 제공하는 표준 확장 체계

주) 커넥터는 각 전사적 정보 시스템 마다 다를 수 있으며, 전사적 정보 시스템과 연계를 위한 하나의 자원 어댑터와 응용 프로그램 개발 도구들로 이루어져 있다. 자원 어댑터는 컨테이너에 플러그인 되어, 커넥터 아키텍처에 정의된 시스템 수준의 계약을 충족시킨다. [J2EE Tutorial]

3.1.15 계약(Contract)

소프트웨어 부품이 제공하는 행동 및 상태 정보로서 해당 부품을 활용하는 클라이언트가 갖 고 있기를 기대하는 바와 일치함

주) 계약은 그 클래스의 모든 공용 필드, 메소드, 성질 및 이벤트들에 대한 서명에 의해서 부분적으로 표현된다. 이것은 각 메소드의 행위 및 각 필드 또는 성질이 나타내고 있는 것에 대한 설명(일반적으로 간단한 텍스트로 기술)이 보강되어 있다. [MS .NET Library]

3.1.16 DAO (Data Access Objects)

데이터베이스에 접근할 수 있도록 해주는 객체 지향 API

주1. 마이크로소프트 DAO는 마이크로소프트 제트 데이터베이스 엔진을 통해 데이터에 접 근하는 객체지향 접근 방법을 제공한다.

주. DAO는 마이크로소프트 오피스의 공유 컴포넌트이며, 자동화를 지원하는 어떤 응용프로그램에서나 사용할 수 있다. [Component Source]

3.1.17 위임(Delegation)

하나의 클래스가 자신에게 필요한 서비스를 다른 클래스에 정의된 메소드 호출을 통해서 얻어 내는 것

주1. 상속처럼 다른 클래스의 정의에 의해서 클래스가 정의되지만, 반드시 정적인 클래스 계층도를 통해서가 아니라, 가끔 실행 시 동적으로 정의되어 새로운 의존관계 (dependencies)가 추가될 수 있다.

주2. 마이크로소프트의 COM은 상속은 지원하지 않지만, 작동하고 있는 시스템에 새로운 행 동들을 추가하는 것을 허용하는 위임의 형태를 지원한다. [CSG]

3.1.18 영역(Domain)

지식이나 활동의 한 분야 주.

영역은 그 분야의 실무자들이 이해하고 있는 개념 및 용어의 집합으로 특징지어진다. [UML Guide]

3.1.19 영역 모형(Domain Models)

하나의 영역에서 중요한 의미를 지닌 개체들을 표준화된 상호 교환 가능한 방법으로 암호화함으로써 데이터를 여러 시스템이나 조직에 걸쳐 공유할 수 있도록 한 일반적인 클래스 라이브러리 또는 객체 모형들 [CSG]

3.1.20 전사적 정보 시스템(Enterprise Information System)

회사 전체에 걸쳐 필요한 정보를 처리하는 기업의 기존 시스템을 구성하고 있으며, 기업에 정보 기반 구조를 제공하는 응용 프로그램.

주. 전사적 정보 시스템은 잘 정의된 서비스를 클라이언트에게 제공하며, 클라이언트는 이 들 서비스를 기업 내부의 인터페이스 또는 원격 인터페이스에 의해 받는다. 이런 전사적 정 보 시스템의 예로 전사적 자원 관리 시스템(ERP), 주전산기의 트랜잭션 처리 시스템, 데이 터베이스 시스템 등을 들 수 있다. [J2EE Tutorial]

3.1.21 전사적 정보 시스템 자원(Enterprise Information System Resource)

전사적 정보 시스템 고유의 기능성을 클라이언트에게 제공하는 개체(엔티티)

주. 예를 들면, 데이터 베이스 시스템에서의 레코드 또는 레코드 집합, ERP 시스템에서의 비즈니스 객체 및 트랜잭션 처리 시스템에서의 트랜잭션 프로그램 등이 여기에 속한다. [J2EE Tutorial]

3.1.22 상호작용(Interaction)

어떤 목적을 성취하기 위해 특정한 문맥(context)내에서 객체들 간에 교환되는 메시지 집합으로 구성된 행동. [UML Guide]

3.1.23 인터페이스(Interface)

클래스나 컴포넌트의 서비스를 명시하는데 사용되는 연산(operation)의 모음 [UML Guide]

3.1.24 인터페이스 상속(Interface Inheritance)

하나 이상의 이전 인터페이스에 있는 모든 메소드들로 새로운 인터페이스를 정의하는 프로세스로서, 객체지향의 상속과 달리 메소드 및 속성의 이름만을 상속받고 메소드의 구현은 상속에 포함하지 않음.

주1. 객체지향의 상속에서는 메소드들의 구현을 포함하여 원래 클래스의 모든 메소드 및 속 성이 상속된다.

주2. 상속 모형을 기술하는 상속 다이어그램에서와 같이 격자 그래프를 사용하여 인터페이스 상속을 모형화 한다. 격자 그래프는 클래스 이름들이 인터페이스 이름들로 대체된 것을 제외하고는 OOP 상속 다이어그램과 같다. [UML Guide]

3.1.25 반복(Iteration)

내부 또는 외부에 출시(release)된 기본 계획 및 평가 기준을 갖추고 있는 활동들의 독립된 집합. [UML Guide]

3.1.26 반복 프로세스(Iterative Process)

소프트웨어 개발 생명 주기 문맥에서, 실행 가능한 출시품(release)의 흐름을 관리하는 프로세스 [UML Guide]

3.1.27 SOAP(Simple Object Access Protocol)

웹 상에서 구조화된 형식(type)에 대한 정보를 주고 받는데 활용되는 XML 기반 프로토콜

주. 이 프로토콜은 응용 프로그램이나 전송에 관한 정보를 포함하고 있지 않아 모듈성과 확 장성이 뛰어나다. [MS .NET Library]

3.1.28 웹 서버(Web Server)

인터넷, 인트라넷 또는 익스트라넷에 접근할 수 있는 서비스를 제공하는 소프트웨어

주1. 웹 서버는 웹 사이트를 주관하고, HTTP 및 다른 프로토콜을 지원하며, 정해진 기능들 을 수행하는 서버 측의 프로그램들(예; 서블릿)을 실행시킨다.

주2. J2EE 아키텍처에서는 웹서버가 웹 컨테이너에 서비스를 제공하고 있다. 예를 들어, 웹 컨테이너는 전형적으로 HTTP 메시지를 처리를 웹 서버에 의존하여 처리한다. 또한, 웹 컨테이너가 동일한 제조사의 웹 서버에 설치된다고 가정하고 있기 때문에 둘 간에 필요한 계약 을 정의하지 않는다. 물론 웹 서버에는 하나 이상의 웹 컨테이너를 설치할 수 있다. [J2EE Tutorial]

3.1.29 웹 서비스(Web Service)

웹 사용자 또는 웹에 연결된 다른 프로그램이 비즈니스 웹 서버로부터 이용할 수 있도록 만 든 서비스(보통 프로그래밍 및 데이터의 일부 조합을 포함하나 가능하면 인적 자원도 포함 할 수 있음)

주. 웹 서비스는 표준 웹 프로토콜을 통해 접근할 수 있는 프로그래밍 가능한 로직으로 SOAP를 이용하여 서비스에 대한 인터페이스를 기술함으로써 서로 다른 플랫폼 상의 인터넷 클라이언트가 XML 프로토콜을 통해 그 로직을 사용할 수 있게 한다. 웹 서비스 제공자는 일반적으로 ASP(응용서비스 제공자)로 알려져 있다. [Component Source]

3.2 모형화 용어

3.2.1 추상 클래스(Abstract Class)

직접적으로 실체화될 수 없어 인스턴스를 가질 수 없는 클래스로서 하위클래스가 상속을 통 해 실체화될 수 있는 기능성을 제공함

주. 추상 클래스의 실체화는 상속받은 하위클래스가 인스턴스 객체를 생성함으로써 간접적으로 이루어진다. 즉, 추상클래스는 직접적인 인스턴스 객체 생성이 불가능하나, 하위클래스를 통해 실체화 시킬 속성과 행위들을 정의하고 이를 상속 시킨다. [UML Guide]

3.2.2 추상화(Abstraction)

개체들을 다른 종류의 개체들로부터 구분시켜 주는 이들의 필수적인 공통 특성을 식별함으로써, 개체들간 유사성을 찾는 것. [UML Guide]

3.2.3 활동도(Activity Diagram)

활동 간의 제어 흐름을 보여주는 다이어그램으로 시스템의 동적 뷰를 표현함

주. 이것은 상태 다이어그램의 특별한 경우로서 대부분의 상태가 활동 상태이며, 한 활동에 서 다른 활동으로의 전이는 대부분 원래 상태에 있던 활동이 종료됨으로써 일어난다. 활동 은 객체의 생명주기 동안 거쳐가는 상태들의 변화를 일으키는 여러 개의 실행 가능한 연산을 의미한다. [UML Guide]

3.2.4 행위자/액터(Actor)

시스템이 수행하는 일련의 순차적인 활동을 설명한 유스케이스와 상호작용할 때에 유스케이스 사용자가 수행하는 역할들의 집합으로 기능성의 중요한 부분을 설명함 [UML Guide]

3.2.5 집합관계(Aggregation)

"전체-부분"를 명시한 연관관계(association)의 한 형태

주1. 집합관계에서는 하나의 구성요소가 다른 요소들의 집합인 더 큰 "전체"의 "부분"이다. 연관관계(association) 참조. [UML Guide]

3.2.6 연관관계(Association)

객체들간의 연결을 표현하는 링크들의 집합을 설명하는 구조적 관계; 행위적인 특징이나 구 조적인 특성을 나타내는 시스템의 추상적 요소인 클래스, 인터페이스, 컴포넌트, 유스케이스 등의 분류자 사이에 존재하는 의미적 관계 [UML Guide]

3.2.7 연관 클래스(Association Class)

연관관계 및 클래스의 속성을 모두 가지고 있는 모형화 요소

주. 연관관계 클래스는 클래스 성질을 가지고 있는 연관관계로 볼 수 있고, 또한 연관관계 성질을 가진 클래스로 볼 수도 있다. 연관관계 참조. [UML Guide]

3.2.8 속성(Attribute)

행위적인 특징이나 구조적인 성질을 나타내는 시스템의 추상적 요소인 클래스, 인터페이스, 컴포넌트, 유스케이스 등과 같은 분류자의 성질의 명칭으로, 인스턴스들이 그 성질에 대하 여 가질 수 있는 값의 범위를 기술함. [UML Guide]

3.2.9 행동(Behavior)

하나의 연산 또는 사건(event)의 결과들을 포함하여 이들의 관찰 가능한 효과들 [UML Guide]

3.2.10 클래스(Class)

동일한 속성, 연산, 관계 및 의미를 공유하고 있는 객체들의 집합을 명명하여 기술한 것 [UML Guide]

3.2.11 클래스도(Class Diagram)

클래스, 인스턴스 및 협력의 집합, 그리고 이들 요소 간의 관계를 나타내는 다이어그램

주. 클래스도는 시스템의 정적인 뷰를 표현한다. [UML Guide]

3.2.12 분류자(Classifier)

행위적인 특징이나 구조적인 특성을 나타내는 시스템의 모형 요소

주. 클래스, 인터페이스, 데이터 유형, 신호, 컴포넌트, 노드, 유스케이스 및 서브시스템이 여기에 포함된다. [UML Guide]

3.2.13 협력도(Collaboration Diagram)

메시지를 주고받는 객체들의 구조적 조직에 초점을 둔 상호작용 다이어그램; 인스턴스들 및 그들 서로 간의 링크들에 대한 상호작용을 나타내는 다이어그램 [UML Guide]

3.2.14 포함관계(Composition)

부분과 전체가 동일한 생명주기를 갖고 있으며 소유관계가 강한 집합관계(aggregation)의 한 형태

주. 포함관계가 이미 존재해도 비고정 다중성을 가진 부분들이 생성될 수 있으나, 이들은 일단 생성되고 나면 포함관계와 생사를 같이한다. 이러한 부분들은 포함관계의 소멸전에 명 시적으로 제거될 수도 있다. [UML Guide]

3.2.15 구체 클래스(Concrete Class)

직접적으로 실체화를 통해 인스턴스를 가질 수 있는 클래스 [UML Guide]

3.2.16 의존성, 의존관계(Dependency)

하나의 개체(독립적 개체)에 대한 변경이 또 다른 개체(의존 개체)의 의미에 영향을 끼칠 수 있는 두개의 개체들 간의 관계 [UML Guide]

3.2.17 전개도/배치도(Deployment Diagram)

실행 시, 처리를 담당하는 노드의 형상과 그 형상속에 존재하는 컴포넌트들을 보여주는 다이어그램.

주. 전개도/배치도는 시스템의 정적 배치뷰를 나타낸다. [UML Guide]

3.2.18 배치뷰(Deployment View)

시스템이 실행되는 하드웨어 토폴로지를 형성하는 노드들로 구성되는 시스템의 아키텍처 뷰

주. 분산된 시스템 내의 노드와 각 노드에 저장된 컴포넌트, 그리고 이들 컴포넌트에 포함된 객체들을 표현하며, 물리적 시스템을 구성하는 부품들의 배포, 인도 및 설치에 초점을 둔다.

[UML Guide]

3.2.19 정련단계(Elaboration)

제품의 비전과 제품의 아키텍처가 정의되는 소프트웨어 개발 생명 주기의 두 번째 단계

[UML Guide]

3.2.20 확장 메커니즘(Extensibility Mechanism)

통제 가능한 형태로 UML을 확장할 수 있도록 지원하는 세 가지 메커니즘(스테레오 유형, 꼬리표 값, 제약사항) 중 하나 [UML Guide]

3.2.21 일반화(Generalization)

보다 특수화된 요소(예; 기업고객, 개인 고객)가 보다 일반적인 요소(예; 고객)를 대체할 수 있는 특수화/일반화 관계 [UML Guide]

3.2.22 초기단계(Inception)

제안된 시스템에 대한 업무의 사례가 개발되고 제시되는 소프트웨어 개발 생명 주기의 첫 번째 단계

주. 초기단계의 주 목표는 새로운 시스템이 달성해야 할 사항, 이 시스템을 구축하는데 필 요한 시간 및 비용에 대하여 이해당사자들 간의 동의를 얻어내기 위한 것이다. [UML Guide]

3.2.23 상호작용도(Interaction Diagram)

객체 집합과 그들 사이에 주고받는 메시지를 포함하는 관계로 구성되는 상호작용을 나타낸 다이어그램으로 시스템의 동적인 뷰를 설명함.

주. 협력도, 순서도 및 활동도 등과 같이 객체들의 상호 작용을 강조하는 여러 유형의 다이어그램에 적용할 수 있는 일반적인 용어이다. [UML Guide]

3.2.24 메타클래스(Metaclass)

인스턴스들이 클래스인 클래스 [UML Guide]

3.2.25 메소드(Method)

연산의 구현 [UML Guide]

3.2.26 모형(Model)

개발할 시스템의 이해를 높이기 위해 시스템의 현실 셰계를 단순화 한 것. 의미적으로는 한 시스템에 대한 폐쇄적 추상화 개념임 [UML Guide]

3.2.27 객체(Object)

추상화를 구체적으로 나타낸 것으로, 경계가 잘 정의되어 명확하고 삭별성을 갖고 있으며 상태 및 행동을 캡슐화 하고 있는 개체. 클래스의 인스턴스 [UML Guide]

3.2.28 객체도(Object Diagram)

어떤 시점에 있는 객체들의 집합과 그들 간의 관계를 나타낸 다이어그램.

주. 객체도는 한 시스템의 정적 설계 뷰 또는 정적 프로세스 뷰를 설명한다.

[UML Guide]

3.2.29 UML(Unified Modeling Languge)

소프트웨어 개발 관련 산출물들을 가시화하고, 명세화 하며, 구축하고, 문서화하기 위한 모형화 언어 [UML Guide]

3.2.30 사용관계(Usage)

한 요소가 올바른 기능의 수행 또는 구현을 위해 다른 요소의 존재를 요구하는 의존성

[UML Guide]

3.2.31유스케이스(Use Case)

시스템이 수행하는 일련의 조치사항(sequences of actions)들을 설명한 것

주. 시스템이 이들 조치사항을 수행함으로써 행위자에게 가시적인 결과를 제공한다.

[UML Guide]

3.2.32 유스케이스도(Use Case Diagram)

유스케이스, 액터 및 이들간의 관계를 나타내는 다이어그램.

주. 유스케이스도는 시스템의 정적 유스케이스 뷰를 설명한다.

[UML Guide]

3.2.33 유스케이스 뷰(Use Case View)

시스템의 최종 사용자, 분석가 및 시험 수행자가 바라보는 대로 시스템의 행동을 기술하고 있는 유스케이스들을 내포하는 시스템 아키텍처에 관한 뷰 [UML Guide]

3.2.34 유스케이스 중심(Use Case-driven)

소프트웨어 개발 생명주기에서 요망하는 시스템의 행동을 수립하고, 시스템 아키텍처를 검 증 및 확인하며, 테스팅 및 프로잭트 이해 당사자간의 원할한 의사소통을 위하여 유스케이스를 주된 산출물로 사용하는 프로세스 [UML Guide]

3.2.35 가시성(Visibility)

특정 이름을 다른 것들이 볼 수 있고 사용할 수 있도록 하는 방법 [UML Guide]

3.3 J2EE 관련 용어

3.3.1 애플릿(Applet)

자체적으로 실행될 수 없고 웹 브라우저와 같은 응용 프로그램에 포함되어 사용되는 작은 프로그램

주. 웹 브라우저 뿐만 아니라 애플릿 프로그래밍 모형을 지원하는, 다른 응용 프로그램 또는 장치에서도 실행될 수 있다. [J2EE Tutorial]

3.3.2 애플릿 컨테이너(Applet Container)

애플릿 프로그램 모형을 지원하는 컨테이너로서 애플릿을 실행할 수 있는 브라우저. [J2EE Tutorial]

3.3.3 응용 컴포넌트 제공자(Application Component Provider)

컴포넌트의 메소드를 구현한 자바 클래스, JSP 페이지 정의서 및 필요한 배치 기술서 등을 제공하는 회사 [J2EE Tutorial]

3.3.4 애플리케이션 클라이언트(Application Client)

J2EE 플랫폼과 별도의 자바 가상 머신에서 실행되는 클라이언트 프로그램.

주. 애플리케이션 클라이언트는 J2EE 플랫폼 API(JNDI, JDBC, RMI-IIOP, JMS등)를 이용하 여 EJB 컴포넌트에 접근한다.

[J2EE Tutorial]

3.3.5 애플리케이션 클라이언트 컨테이너(Application Client Container)

애플리케이션 클라이언트 컴포넌트를 지원하는 컨테이너 [J2EE Tutorial]

3.3.6 애플리케이션 클라이언트 모듈(Application Client Module)

애플리케이션 클라이언트 배치 기술서와 하나 이상의 클래스들로 구성되는 소프트웨어 단위 [J2EE Tutorial]

3.3.7 권한(Authorization) 메소드 또는 자원에 접근할 수 있는지를 판단하는 프로세스.

주. J2EE 플랫폼에서의 권한은 인증을 통해서 요청을 한 대표 사용자(principal)가 사전에 정의된 보안 역할이 있는지에 따라 결정된다. 보안 역할이란 응용 프로그램 컴포넌트 제공 자 또는 조립자가 정의한 사용 권한의 유형에 따른 사용자의 논리적 그룹을 말하며, 배치 담당자는 이들 보안 역할을 각각 보안 식별자에게 지정한다. 보안 식별자는 운영 환경에서 의 대표 사용자(principals) 또는 그룹일 수 있다.

[J2EE Tutorial]

3.3.8 빈 관리 트랜잭션(Bean-managed Transaction)

엔터프라이즈 빈의 코드에서 범위가 정해지는 트랜잭션 [J2EE Tutorial]

3.3.9 커넥터 아키텍처(Connector Architecture)

J2EE 제품과 기존의 전사적 정보 시스템을 통합을 위한 아키텍처

주. 이 아키텍처는 전사적 정보 시스템 판매사가 제공하는 자원 어댑터와 이 어댑터가 플러그인 될 수 있는 J2EE 제품으로 구성되며, 자원 어댑터가 J2EE 제품(예; 트랜젝션, 보안 및 자원 관리 등)에 플러그인 되기 위해 지원해야 할 일련의 계약을 정의하고 있다. [J2EE Tutorial]

3.3.10 컨테이너(Container)

생명주기 관리, 보안, 배치 및 실행환경 서비스를 컴포넌트에 제공하는 개체.

주. 컨테이너에는 EJB 컨테이너, 웹 컨테이너, JSP 컨테이너, 서블릿 컨테이너, 애플릿 컨테 이너, 애플리케이션 클라이언트 컨테이너 등이 있으며, 이들 각 컨테이너의 유형별로 컴포 넌트에 적합한 서비스를 제공한다.

[J2EE Tutorial]

3.3.11 컨테이너 관리 트랜잭션(Container-managed Transaction)

범위가 EJB 컨테이너에 의해 정의되어 있는 트랜잭션 주. 엔티티 빈은 '컨테이너 관리 트랜잭션'을 사용하여야 한다 [J2EE Tutorial]

3.3.12 전개/배치(Deployment)

소프트웨어를 운영 환경에 설치하는 프로세스 [J2EE Tutorial]

3.3.13 전개/배치 기술서(Deployment Descriptor)

EJB 모듈및 응용 프로그램의 전개/배치 방법에 관한 정보를 가지고 있는 XML 파일

주. 이 배치 기술서는 EJB의 속성을 손쉽게 변경할 수 있게 해 줌으로써 전개/배치 도구가 특정 컨테이너 선택 사양에 맞게 모듈이나 응용 프로그램을 전개/배치할 수 있도록 해준다.

[J2EE Tutorial]

3.3.14 EAR 파일(EAR File)

J2EE 응용 프로그램을 포함하고 있는 JAR 파일 [J2EE Tutorial]

3.3.15 EJB 컨테이너(EJB Container)

J2EE 아키텍처의 EJB 컴포넌트가 지켜야 할 계약을 구현한 컨테이너

주. EJB 컴포넌트의 계약은 EJB의 실행 환경을 명시한 것으로, 보안, 동시성, 생명주기 관 리, 트랜잭션, 전개/배치, 명명 등의 서비스들을 포함하고 있다. EJB 컨테이너는 EJB나 J2EE 서버에 의해 제공된다.

[J2EE Tutorial]

3.3.16 EJB 문맥(EJB Context)

트랜잭션 및 컨테이너의 환경 속성, 호출자에 대한 정보 등에 엔터프라이즈 빈이 접근할 수 있도록 관련 서비스를 제공하는 객체 [J2EE Tutorial]

3.3.17 EJB 홈 객체(EJB Home Object)

엔터프라이즈 빈을 위하여 생명주기 연산(생성, 삭제, 검색)을 제공하는 객체

주. EJB 홈 객체의 클래스는 컨테이너의 전개/배치 도구에 의해 생성되며, EJB 홈 객체는 엔터프라이즈 빈의 홈 인터페이스를 구현한다. 클라이언트는 EJB 홈 객체를 참조하여 EJB 객체에 대하여 생명주기 연산을 수행하고, JNDI를 이용하여 EJB 홈 객체의 위치를 결정한다.

[J2EE Tutorial]

3.3.18 EJB JAR 파일(EJB JAR File)

EJB 모듈을 포함하고 있는 JAR 파일 [J2EE Tutorial]

3.3.19 EJB 모듈(EJB Module)

하나 이상의 엔터프라이즈 빈과 전개/배치 기술서로 이루어진 소프트웨어 단위 [J2EE Tutorial]

3.3.20 EJB 객체(EJB Object)

클래스가 엔터프라이즈 빈의 원격 인터페이스를 구현하고 있는 객체

주. 클라이언트가 엔터프라이즈 빈 인스턴스를 직접 참조하여 사용하는 경우는 없고, 항상 EJB 객체를 사용하게 된다. EJB 객체의 클래스는 컨테이너의 전개/배치 도구에 의해 생성된다.

[J2EE Tutorial]

3.3.21 EJB 서버(EJB Server)

EJB 컨테이너에게 서비스를 제공하는 소프트웨어

주. 일반적으로 EJB 컨테이너는 EJB 서버의 일부인 트랜잭션 관리자에 의존해서 참여하고 있는 모든 자원 관리자 전반에 걸쳐 2단계 커밋(precommit, commit)을 수행한다. J2EE 아키텍처에서는 EJB 컨테이너는 동일한 회사의 EJB서버에 의해 지원받는다는 것을 가정하기 때문에, 컨테이너와 서버 간에 필요한 계약을 명시하지 않는다. EJB 서버는 하나 이상의 EJB 컨테이너를 주관할 수 있다.

[J2EE Tutorial]

3.3.22 엔터프라이즈 빈(Enterprise Bean)

비즈니스 작업 또는 비즈니스 개체(엔티티)를 구현하며 서버 측의 EJB 컨테이너에 위치한 컴포넌트

주. 엔티티 빈, 세션 빈, 메시지 빈 등이 여기에 속한다.

[J2EE Tutorial]

3.3.23 엔터프라이즈 빈 제공자(Enterprise Bean Provider)

엔터프라이즈 빈 클래스, 원격 및 홈 인터페이스, 전개/배치 기술서 파일을 만들고, EJB JAR 파일에서 이들을 패키지화 하는 응용 프로그램 작성자 [J2EE Tutorial]

3.3.24 홈 인터페이스(Home Interface)

엔터프라이즈 빈에 정의되어야 하는 두가지 인터페이스 중의 하나로 엔터프라이즈 빈을 관 리하는 데 필요한 메소드를 정의.

주1. 홈 인터페이스는 하나도 없을 수도 있다. 세션 빈의 홈 인터페이스는 생성과 삭제 메소드를 정의하고 있으며, 엔티티 빈의 홈 인터페이스는 생성과 삭제 외에 추가적으로 검색 과 관련된 메소드를 정의한다.

주2. 원격 인터페이스 참조

[J2EE Tutorial]

3.3.25 J2EE 서버(J2EE Server)

J2EE 제품의 실행 부분으로 EJB 및 웹 컨테이너를 제공함 [J2EE Tutorial]

3.3.26 JAR(Java Archive)

여러 파일을 하나로 묶어주는 플랫폼에 독립적인 파일 포맷 [J2EE Tutorial]

3.3.27 JNDI(Java Naming and Directory Interface TM, 자바 명명 및 디렉토리 인터페이스)

자바 응용 프로그램이 일관성 있는 디렉토리 서비스를 구현할 수 있도록 명명 및 디렉토리 기능성을 제공하는 API [J2EE Tutorial]

3.3.28 JMS(JaveTM Message Service, 자바 메시지 서비스)

기업간의 중요한 비즈니스 데이터나 이벤트를 비동기적으로 교환하는데 있어서 보다 신뢰성 있고 유연한 서비스를 제공하기 위한 API [J2EE Tutorial]

3.3.29 JTA(JavaTM Transaction API, 자바 트랜잭션 API )

응용 프로그램과 J2EE 서버 간의 트랜잭션을 처리할 수 있도록 지원하는 API [J2EE Tutorial]

3.3.30 원격 인터페이스(Remote Interface)

엔터프라이즈 빈에 정의되어야 하는 두가지 인터페이스 중의 하나로 클라이언트가 호출할 수 있는 비즈니스 메소드를 정의함

주. 홈 인터페이스 참조

[J2EE Tutorial]

3.3.31 자원 어댑터(Resource Adapter)

전사적 정보 시스템에 연결하기 위해서 EJB 컨테이너 또는 애플리케이션 클라이언트가 사 용하는 시스템 수준의 소프트웨어 드라이버

주. 자원 어댑터는 전형적으로 전사적 정보 시스템에 고유한 것으로 라이브러리 형태로 활 용 가능하고, 이를 이용하는 서버나 클라이언트의 주소 공간 내에서 사용된다, 또한, 컨테이너에 플러그인 됨으로써 컨테이너에 전개된 컴포넌트가 클라이언트 API(어댑터에 의해 노출되어 있음)를 이용하여 전사적 정보시스템에 접근할 수 있도록 한다. 자원 어댑터와 EJB 컨테이너는 협력을 통하여 전사적 정보시스템과 연결에 필요한 기반이 되는 메커니즘, 즉 트랜잭션, 보안 및 커넥션 풀링을 제공한다.

[J2EE Tutorial]

3.3.32 자원 관리자(Resource Manager)

공유 자원에 접근할 수 있는 기능을 제공하는 소프트웨어

주. 자원 관리자는 외부 통제를 받고 트랜잭션 관리자에 의해 조정을 받는 트랜잭션에 관여하며, 전형적으로 클라이언트와 서로 다른 주소 공간 또는 서로 다른 기계에 존재한다. 전 사적 정보 시스템이 자원 및 트랜잭션 관리 문맥에서 언급될 때는 전사적 정보 시스템을 자 원 관리자로 볼 수 있다.

[J2EE Tutorial]

3.3.33 상태 유지 세션 빈(Stateful Session Bean)

클라이언트와 서버 간의 상호작용에서 만들어진 상태를 갖는 빈 [J2EE Tutorial]

3.3.34 무상태 세션 빈(Stateless Session Bean)

클라이언트와 서버간의 상호작용에서 상태를 갖지 않는 빈

주. 무상태 세션 빈의 모든 인스턴스는 동일하다 [J2EE Tutorial]

3.3.35 트랜잭션(Transaction)

데이타를 수정하는 가장 작은 작업 단위

주. 트랜잭션은 하나 이상의 프로그램 문장으로 구성되며, 이들 각각의 문장은 작업을 완료하거나 원래 상태로 되돌려준다. 또한, 트랜잭션은 여러 사용자가 동일한 데이터를 동시에 접속할 수 있도록 한다.

[J2EE Tutorial]

3.3.36 웹 컴포넌트(Web Component)

요청에 응답하여 서비스를 제공하는 서블릿이나 JSP 형태의 컴포넌트 [J2EE Tutorial]

3.3.37 웹 컨테이너(Web Container)

J2EE 아키텍처의 웹 컴포넌트가 만족해야 하는 계약을 구현하는 컨테이너

주. 이 계약에는 보안, 동시성, 생명주기 관리, 트랜잭션 및 전개/배치 등의 서비스를 포함하여 웹 컴포넌트에게 제공해야 하는 실행 환경이 명시되어 있다. 웹 컨테이너는 JSP 컨테이너와 같은 서비스를 제공하며, 웹 또는 J2EE 서버에 의해 제공된다.

[J2EE Tutorial]

3.4 마이크로소프트의 .NET 용어

3.4.1 접근 제어 목록(Access Control List)

자원 접근 권한 규칙을 명시한 목록

주. 윈도우 NT에서는 사용자 별 접근 권한 목록을 통해 개별 사용자마다 접근 가능한 자원을 명시하고 있다.

[MS .NET Library]

3.4.2 ADO.NET

닷넷 프레임워크 클래스 라이브러리 중에서 데이터 접근과 관련된 클래스 라이브러리들을 모아 놓은 집합 [MS .NET Library]

3.4.3 응용 프로그램 영역/애플리케이션 도메인(Application Domain)

동일한 응용 프로그램 범위안에서 생성된 객체들을 다른 응용프로그램의 범위에 있는 객체들과 구별하기 위하여 공용 언어 실행기(CLR)가 설정한 경계

주. 응용 프로그램 영역은 한 응용 프로그램에서 만들어진 객체와 다른 응용 프로그램에서 만들어진 객체를 분리하는 역할을 담당하여, 실행시의 동작을 예측 가능하게 한다. 한 프로세스에 여러 응용 프로그램 영역이 존재할 수 있다.

[MS .NET Library]

3.4.4 ASP.NET

ASP의 진일보된 기술로서 서버기반 웹 응용 프로그램의 구축 시 활용되는 프레임워크

[MS .NET Library]

3.4.5 ASP.NET 서버 컨트롤(ASP.NET Server Control)

사용자 인터페이스 및 관련 기능을 내포하고 있는 서버 측 컴포넌트

주. ASP.NET 서버 컨트롤은 System.Web.UI.Control을 직·간접적으로 상속받아 만들어지며, 웹 서버 컨트롤, HTML 서버 컨트롤, 모바일 컨트롤 등으로 구성되어 있다.

[MS .NET Library]

3.4.6 어셈블리(Assembly)

하나의 구현 단위로서 구축되어, 버전 관리되고, 전개되는 기능성 집합체로, 하나 이상의 파 일로 구성됨

주. 어셈블리는 닷넷 응용 프로그램을 구축하는 기본 단위이다. CLR에 의해 관리되는 모든 형식과 자원들은 이들의 구현 단위 내에서만 접근 가능한 것으로 표시되거나, 구현 단위 외 부 코드에서도 사용할 수 있는 것으로 지정된다. 실행 시, 어셈블리는 요청을 해석하기 위해 이름의 범위를 설정하고, 가시성 경계가 강요된다. 실행 엔진은 모든 수행 중인 객체에 필요한 어셈블리를 파악해서 그 위치를 찾아낼 수 있다. 이는 모든 형식(type)이 어셈블리의 문맥안에서 로드되기 때문이다.

[MS .NET Library]

3.4.7 어셈블리 캐시(Assembly Cache)

어셈블리를 개별적으로 저장하기 위해 사용되는 기계 범용 코드 캐시(시스템 수준에서 활용 되는 캐시)

주. 어셈블리 캐시에는 전역 어셈블리 캐시, 다운로드 어셈블리 캐시의 2 종류가 있다. 전역 어셈블리 캐시는 다수의 응용 프로그램이 공유하도록 설치된 어셈블리를 저장하고, 다운로드 어셈블리 캐시는 인터넷이나 인트라넷에서 다운로드한 코드들을, 다운로드를 유발시킨 응용 프로그램과 분리하여 저장한다. 그렇게 함으로써 한 응용 프로그램을 작동시키기 위해 다운로드된 코드가 다른 응용 프로그램들에 영향을 끼치지 않게 된다.

[MS .NET Library]

3.4.8 어셈블리 명세부(Assembly Manifest)

어셈블리에 들어 있으면서 어셈블리 자신을 설명하고 있는 모든 어셈블리의 총괄편으로 어셈블리에 대한 식별자, 어셈블리를 구현하는 파일들, 어셈블리를 구성하는 형식 및 자원들, 컴파일 시 다른 어셈블리와의 종속성, 그리고 어셈블리 작동 시 요구되는 허가 목록 등을 담고 있음

주. 이들 정보를 활용하여 참조 요청 해석, 버전 관련 바인딩 정책 강화 및 로드된 어셈블 리간 무결성 파악 등을 수행한다. 이런 어셈블리의 자체적으로 설명하는 특성으로 인하여 설치 및 전개가 간명해 진다.

[MS .NET Library]

3.4.9 박싱(Boxing)

값 형식(value type)의 인스턴스를 객체로 변환하는 것

주. 이것은 이 인스턴스가 실행시에 충분한 형식type)정보를 갖게 되고, 힙(heap)에 할당될 것이라는 것을 뜻한다. 마이크로소프트 중간 언어(MSIL) 명령 중 box라는 명령이, 값 형식 (value type)의 복사본을 만들고 이것을 새로 할당된 객체에 내장함으로써 값 형식을 객체 로 변환시킨다.

[MS .NET Library]

3.4.10 코드 접근 보안(Code Access Security)

코드에 대한 권한을 보안 정책에 따라 결정하고, 이에 따라 해당 코드가 수행할 연산을 제한하는 실행 환경의 메커니즘

주. 호출자가 해당 코드를 활용하기 위해서는 필요한 권한을 가지고 있어야 한다.

[MS .NET Library]

3.4.11 코드 배후 클래스(Code-behind Class)

확장자 aspx를 가지는 파일에서 활용되는 클래스로서 독립된 파일(dll, cs 확장자를 가질 수 있음)에 존재함

주. 예를 들어, ASP.NET 커스텀 서버 컨트롤을 만드는 배후 클래스를 작성할 때, 이 배후 클래스는 확장자가 aspx인 파일이 호출하는 코드를 포함하지만, 이 aspx 파일에 존재하지 않는다.

[MS .NET Library]

3.4.12 COM 래퍼(COM Callable Wrapper: CCW)

기존 COM 응용 프로그램이 닷넷 프레임워크 클래스와 같은 관리 클래스(managed classes) 를 사용할 수 있도록 실행환경이 만들어주는 프록시 객체. [MS .NET Library]

3.4.13 공용 언어 실행기(Common Language Runtime: CLR)

관리 코드의 실행 엔진으로 언어간 통합, 코드 접근 보안, 객체 생명주기 관리, 그리고 디버 깅 및 프로필 작성 지원 등의 서비스를 관리 코드에 제공함

주. 공용 언어 실행기(CLR)는 썬마이크로시스템즈의 JVM(Java Virtual Machine)에 비유될 수 있다. 이것에 대해 컴파일된 프로그램은 언어 고유의 실행 환경이 필요하지 않고, 윈도 우 2000 또는 윈도우 XP가 설치된 어떤 시스템에서나 용이하게 실행될 수 있다.

[MS .NET Library]

3.4.14 공용 언어 명세(Common Language Specification: CLS)

상호 호환되는 도구들이 지원하는 언어 특성들의 하부 집합

주. 공용 언어 명세를 따르고 있는 컴포넌트 및 도구들은 다른 공용 언어 명세를 따르는 컴포넌트 및 도구들과 상호 운용성이 보장된다.

[MS .NET Library]

3.4.15 공용 객체 파일 형식(Common Object File Format: COFF)

플랫폼간 이식성이 보장되는 32 비트 프로그래밍에서의 실행 파일 및 오브젝 파일의 형식

주. 마이크로소프트 환경에서는 PE 파일 형식이라 불린다.

[MS .NET Library]

3.4.16 공용 형식 명세(Common Type System)

실행 환경에서 형식을 어떻게 정의하고, 사용하며 및 관리하는가를 결정하고 있는 명세 [MS .NET Library]

3.4.17 문맥 속성(Context Property)

객체 인스턴스의 속성을 대신 보유하고 있는 상태를 관리하는 묵시적 상태 및 코드

주. 예를 들어, 트랜잭션의 문맥 속성은 그 객체가 참여하고 있는 트랜잭션의 트랜잭션 식별자를 보유하고 있다.

[MS .NET Library]

3.4.18 사용자 지정 속성 클래스(Custom Attribute Class)

사용자 지정 메타데이터(custom metadata)를 표현하는 클래스 [MS .NET Library]

3.4.19 사용자 지정 컨트롤(Custom Control)

닷넷 프레임워크 클래스 라이브러리에 속하지 않는 것으로 사용자 또는 제3의 소프트웨어 제 공자가 작성한 컨트롤

주. 이것은 사용자 컨트롤을 포함하는 일반 용어로 사용자 지정 서버 컨트롤은 웹폼에서 활용되며, 사용자 지정 클라이언트 컨트롤은 원폼 응용 프로그램에서 활용된다.

[MS .NET Library]

3.4.20 도메인 중립(Domain-neutral)

어셈블리가 실행될 영역을 지정하지 않고 어셈블리를 로드하는 것

주. 도메인 중립적인 방식으로 어셈블리를 로드할 경우, 해당 어셈블리의 코드 및 연관된 실행 환경 데이터 구조를 한 프로세스내에 있는 모든 용용 프로그램 영역에서 공유할 수 있다.

[MS .NET Library]

3.4.21 닷넷 데이터 제공자(.NET Data Provider)

관계형 데이터베이스의 정보를 접근할 수 있도록 해주는 ADO.NET 컴포넌트 [MS .NET Library]

3.4.22 닷넷 프레임워크(.NET Framework)

웹 서비스 또는 응용 프로그램의 구축, 전개/배치 및 실행에 필요한 플랫폼

주. 인터넷 기반의 응용 프로그램의 전개 및 운영상의 문제점 해결 방안을 제공할 뿐아니라 기존 시스템을 차세대 응용 프로그램 및 서비스로 통합하기 위한 생산성 향상 지원, 표준 지원 및 다중 언어 지원과 같은 환경을 제공해준다. 이것은 크게 세가지 즉, 공용 언어 실 행기(CLR), 통합 클래스 라이브러리의 계층화된 집합 및 ASP의 컴포넌트화된 버전 (ASP.NET)으로 이루어져 있다.

[MS .NET Library]

3.4.23 증거/에비던스(Evidence)

보안관련 접근 허가 판단 기준이 되는 코드의 속성들(디지털 서명 또는 코드가 출처된 사이 트나 지역) [MS .NET Library]

3.4.24 실행 파일(Executable File)

OS 로더에 의해 메모리에 로드되어 실행될 수 있는 PE 파일 형식의 파일로 확장자가 exe 또는 dll임

주. 닷넷에서는 PE파일을 OS에서 실행하기 전에 CLR이 해당 OS용 원시 코드로 바꿔야 한다.

[MS .NET Library]

3.4.25 명력적 보안 검사(Imperative Security Check)

보안이 설정된 코드에서 보안 메소드의 호출시 수행되는 보안 검사

주1. 이 검사 방식은 자료 중심적으로 수행되며, 한 객체 또는 메소드 내의 한 곳에 국한될 수 있다. 예를 들어, 보호해야 할 파일 이름을 실행 시에만 알 수 있을 경우, 보안 메소드에 매개 변수(parameter)로 파일 이름을 넘김으로써 명령적 보안 검사가 실시되게 할 수 있다.

주2. MS에서 “선억적 보안 검사”로 사용되고 있는 Declaritive Security Check와 대비되는 용어이다.

[MS .NET Library]

3.4.26 명명 컨테이너(Naming Container)

페이지 및 사용자 지정 컨트롤을 포함하여 INamingContainer 인터페이스를 구현하는 모든 ASP.NET 컨트롤

주. 상기 컨트롤들은 자신과 하부 컨트롤들을 위해 하나의 고유한 ID 명명 공간을 생성한다. 따라서, 모든 컨트롤은 전체 ASP.NET 응용 프로그램에서 유일한 식별자를 갖게된다.

[MS .NET Library]

3.4.27 권한 클래스(Permission Class)

권한 검사를 지원함으로써 자원 접근 또는 식별자를 정의하고 있는 클래스.

주. 보통 접근 권한(Access Permission)과 함께 “접근 권한 클래스”로 사용되고 있다.

[MS .NET Library]

3.4.28 플랫폼 호출(Platform Invoke)

관리 코드가 비관리 원시 DLL을 호출할 수 있도록 하는 실행 환경의 기능 [MS .NET Library]

3.4.29 PE 파일(Portable Executable File)

실행 프로그램 및 실행 프로그램 내에 링크된 파일의 파일 형식

주. 실행 파일이나 관련 파일들이 따르는 파일 형식으로 이를 준수할 시 상호 운용성이 보장된다.

[MS .NET Library]

3.4.30 대표 사용자(Principal)

사용자의 식별자 및 역할을 대표하여 실제 사용자를 대신하여 행동하는 개체 주. 주로 닷넷 프레임워크 보안에서 사용되는 용어로, 여기에서 보안 시스템이 이것을 인지하 여 자원에 대한 접근 권한을 부여한다. 즉, 실제로 보안 시스템의 인증을 받는 개체로서 인증번호 등이 여기에 해당된다. [MS .NET Library]

3.4.31 전용 어셈블리(Private Assembly)

오직 하나의 응용 프로그램에서만 사용되는 어셈블리

주. 전용 어셈블리는 이를 사용하는 응용 프로그램의 디렉토리 구조속에 전개된다.

[MS .NET Library]

3.4.32 역할(Role)

보안 측면에서 동일한 특권을 가지고 있는 대표 사용자(Principal)의 집합적 명칭 [MS .NET Library]

3.4.33 실행 환경 호스트(Runtime Host)

실행 시기 및 실행 기간이 관리되는, ASP.NET, 인터넷 익스플로러 또는 윈도우 쉘과 같은 운영 환경

주. 실행 환경 호스트는 응용 프로그램 영역을 생성하고, 그 안에서 사용자를 대신하여 관 리 코드를 수행한다.

[MS .NET Library]

3.4.34 안전모드(Safe Mode)

수행할 어셈블리를 컴파일 시 지정한 관련 어셈블리의 해당 버전과만 연동되도록 제한하는 버전 정책의 일종 [MS .NET Library]

3.4.35 보안 허점(Security Hole)

의도적으로 보호를 하면 보안이 되는 컴퓨터, 컴포넌트, 응용 프로그램 또는 기타 온라인 상의 자원 등에 접근할 수 있는, 고의적은 아니나 보호되지 않은 입구, [MS .NET Library]

3.4.36 서비스 공격(Service Attack) 시간, 메모리 또는 기타 자원들을 과도하게 점유하여 고의적으로 정당한 자원의 접근을 방 해하는 공격 [MS .NET Library]

3.4.37 공유 어셈블리(Shared Assembly)

여러 응용 프로그램에서 참조할 수 있는 어셈블리

주. 해당 어셈블리는 공유될 수 있음을 명시해야만 한다. 즉, 어셈블리에 엄격하게 암호화된 이름을 부여함으로써 공유될 수 있도록 어셈블리를 명확하게 구축하여야 한다.

[MS .NET Library]

3.4.38 동시 병행 실행(Side-by-side Execution)

동일한 어셈블리의 여러 버전을 동시에 실행시킬 수 있는 기능

주. 동일한 컴퓨터 또는 동일한 프로세스나 응용 프로그램 영역에서 작동될 수 있다. 여러 버전이 동시에 병행하여 실행될 수 있도록 하는 것은 실행 시에 버전 관련 강건성을 지원하 는데 필수적인 요소이다.

[MS .NET Library]

3.4.39 템플릿(Template)

ASP.NET에서, 템플릿화 된 ASP.NET 서버 컨트롤에게 가시적 인터페이스를 제공하기 위 해 사용되는 선언적 페이지 구성 요소

주. 템플릿은 ASP.NET 서버 컨트롤임을 나타내는 선언적 구문뿐만 아니라, 텍스트, HTML 및 데이터-바인딩 표기 등의 UI 구성 요소들도 포함하고 있다. 템플릿은 확장자가 .ascx인 텍스트 파일로 지속적으로 보관될 수 있다.

[MS .NET Library]

3.4.40 템플릿 컨트롤(Templated Control)

자체적으로 가시적 인터페이스를 제공하지 않지만, 이를 제공하는 템플릿을 사용자가 공급할 수 있도록 해주는 ASP.NET 서버 컨트롤

주. ASP.NET 페이지 파서는 템플릿 컨트롤을 만나면 해당 컨트롤의 템플릿을 문법적으로 분석(parsing)하고, 가시적 인터페이스를 공급하는 자식 컨트롤을 동적으로 생성한다. 예를 들면, ASP.NET 서버 컨트롤인 Repeater 및 DataList가 여기에 해당한다. DataGrid는 엄격 히 말하면 템플릿 컨트롤은 아니지만, 템플릿을 사용하여 사용자 인터페이스를 커스터마이징한다.

[MS .NET Library]

3.4.41 비관리 코드(Unmanaged Code)

실행시에 요구되는 사항들을 고려하지 않고 작성된 코드

주. 비관리 코드는 CLR 환경에서 실행은 가능하지만, CLR 환경에서 제공하는 부가 기능 (예; 가비지 콜렉션, 디버깅 등)을 제공받지 못한다. [MS .NET Library]

3.4.42 값 형식(Value Type)

값을 표현하는 비트들에 대하여 그 순서를 명시함으로써 값을 설명하는 데이터 형식

주. ‘값 형식’의 인스턴스에 대한 형식 관련 정보가 실행시에 그 인스턴스와 함께 저장되지 않지만, 메타데이타에서 유용하다. 인스턴트들은 박싱을 통하여 객체로 취급될 수 있다.

3.4.9 박싱 참조.

[MS .NET Library]

3.4.43 버전 정책(Version Policy)

어떤 버전의 어셈블리들을 활용할지 결정하는 것으로 환경 설정 파일에 기록됨 [MS .NET Library]

3.4.44 뷰 상태(View State)

ASP.NET 페이지 및 페이지 내의 ASP.NET 서버 컨트롤 성질들의 현재 설정 상태

주. ASP.NET은 서식(form)이 전송되는 때(서버로 보내질 때)와 대비해서 서식(form)이 처 음 요청된 때를 식별할 수 있어, 이에 따라 적절한 프로그래밍을 할 수 있도록 하고 있다.

[MS .NET Library]

3.4.45 웹폼(Web Forms)

웹 브라우저에 HTML로 출력되는 서버 컨트롤에 대한 지원 기능을 담당하는 ASP.NET의 페이지 프레임워크

주. ASP.NET의 웹폼은 HTML 스타일 기반의 레이아웃이며 웹 페이지 레이아웃의 모든 특성을 제공하여, 브라우저 상에서 사용자 인터페이스를 구현하는 응용 프로그램을 만드는데 사용된다.

[MS .NET Library]

자료 : 정보통신단체표준 TTAS.KO-11.0029 소프트웨어 컴포넌트 용어 표준