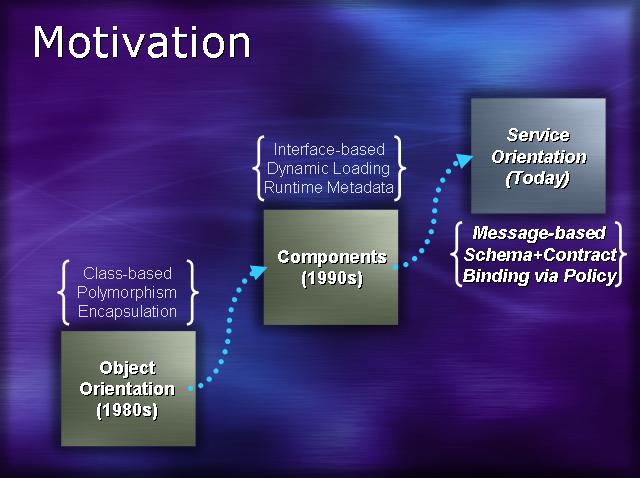
**1. Software Paradigm의 진화**

Service란 무엇인가? 한번 새롭게 정의를 내려 봅시다.

아래 화면의 그림은 재사용이라는 입장에서 본 S/W Paradigm의 진화의 모습입니다.



1980 년대에는 객체라는 개념이 한 시대를 풍미했다는 것은 아무도 부인하지 않으실 겁니다.

Object의 궁극적인 목적은 재사용성 (Reuse)이고,

이러한 것을 보다 잘 지원하기 위해 상속, 조합, 다형성, 캡슐화등 의 개념을 지원했습니다.

재사용성 측면에서 Object를 보도록 합시다. Object는 재사용성의 Domain(범위)이 언어 종속적이라는 단점을 가집니다.

C++ Class로 만든 객체를 다른 언어에서 사용할 수 있나요? 대답은 아니라는 것입니다.

그리하여 언어의 종속성을 탈피하기 위해 나온 개념이 Component 입니다.

Component는 IDL이라는 중간 매개언어를 이용하여 다른 언어로 Proxy 코드를 생성하여 사용합니다.

COM, CORBA, RMI 등이 대표적인 경우이지요.

이 Component라는 개념은 재사용성 측면에서 보았을때는 언어에 독립적이라는 장점을 가집니다.

우리가 COM 형태대로 만들면, VB, VC++ 등 MIcrosoft 플랫폼의 언어 어디에서든지 사용할 수 있는 장점을 가집니다.

하지만 Component는 Platform 종속적이라는 단점을 가집니다. COM은 MIcrosoft 플랫폼에서만 서로를 이해할 수 있는 데이터 포멧과 확장된 IDL 구조를 가지며,

CORBA도 역시 CORBA만 이해할 수 있는 CDR (Common Data Representation) 만을 이용하여 서로 데이터를 주고 받습니다.

Component는 재사용성 측면에 볼때는 언어의 독립성은 얻을수 있었지만, 플랫폼 종속적이라는 단점을 가져왔습니다.

궁극적인 재사용의 기본 단위는 언어에 독립적이고, 플랫폼에 독립적인 단위가 되어야 될 것입니다.

바로 재사용성의 측면에 보았을때, **언어에 독립적이면서, 플랫폼에 독립적인 재사용성을 극대화한 모듈**을 **Service**라고 정의하겠습니다.

(많은 분이 이의를 가하실 수 있는 부분이지만, Don Box라는 분의 의견을 그대로 받아들인 것이니, 어느 정도 신뢰성을 얻을 수 있으리라 봅니다.)

그럼 Service와 Component의 차이점을 자세히 알아보도록 하겠습니다.

기존의 Component는 IDL과 IDL Compiler를 이용하여, 서로 이질적인 환경 (다른 플랫폼, 다른 언어)에서 메세지를 주고 받아 왔습니다.

IDL을 뜯어보면, 단순히 Operation과 Data에 대한 나열로 표시되어져 있으며, 이질적인 환경에 대한 데이터를 변화하는 역할은

ORB (Object Request Broker)의 몫 이었습니다.

하지만 Service라는 것은 메타데이터(공통된 데이터 포멧)를 이용하여 여러 플랫폼과 언어들이 Service를 이해할수 있기 때문에

Mashaling과 Un-Mashaling의 부담을 줄일수 있었습니다.

그리고 IDL보다 좀더 자세한 정보들을 기술하였습니다.

사용되는 Data 형, Operation형 외에도, Binding 정보(Service의 물리적 주소, 사용 되어지는 프로토콜)를 나타태고,

Policy를 이용하여 계약 사항 ( 운영적인 측면 - QoS, Security 등.. )들을 표기함으로써, 좀더 Machine-Readable한 의미 정보를 제공함으로,

언어와 플랫폼에 독립적인 재사용 가능한 모듈로 충분 조건을 제공하는 구조를 가집니다.

추가 설명)

Web Service는 Service가 아니다. Web Service는 Service를 구현하기 가장 쉬운 방법일뿐입니다.

일단 Web Service와 Service를 Mapping 시키면 아래와 같이 되어 집니다.

- 메타 데이터

XML을 사용합니다.

- Web Service는 WSDL과 WS-Policy를 이용하여 Service의 스키마와 Contract를 정의합니다.

위 그림에서 Binding이 바로 프로토콜의 정보가 될 것이며, Message Based Schema가 SOAP 통신지 주고 받는 데이터 타입과 SOAP 메세지의 포멧일 것입니다.

이러한 것이 Policy (WS - Policy)와 메타데이터 (XML 기반의 WSDL)에 의해 정의되어 집니다.

그리고 WS-Policy를 이용하여 QoS (Security, RealTime 등..) 같은 부분에 대해서 정의를 할수 있습니다.

**2. Microsoft SOA Development History (Microsoft사의 SOA 지원 역사)**

****

과거부터 지금까지 Microsoft사의 SOA의 지원 모델의 변천사를 보도록 하겠습니다.

위 그림을 보면 크게 4가지의 제품으로 Microsoft사는 Service를 지원하였습니다.

1. Microsoft SOAP Toolkit

Visual Studio 6.0 에서 Web Service를 지원하기 위해 나온 툴로서

COM과 CORBA간의 데이터를 주고 받는 것이 가능합니다.  하지만 SOAP의 초창기 모델인 SOAP - RPC를 사용하기 때문에.

Primitive (기본 데이터 형 - integer , string....) Type으로 서로 데이터를 주고 받는 것은 가능하지만,

User Defined Type (사용자 자료형)을 사용하여 Web Service를 만들면, 호환성을 제공하지 않는 단점이 있습니다.

이러한 문제가 야기 된것은 SOAP 1.1 Spec시 사용자 정의 자료형을 어떻게 기술한 것이지에 대한 내용이 빠져 있었기 때문입니다.

그래서 많은 밴더들이 각자의 형태로 WSDL의 Type 정보를 생성했기 때문에 호환성이 이루어 지지 않았습니다.

2. ASMX (Micorosft ASP.NET Web Service)

위의 SOAP-RPC의 문제를 해결하기 위해, SOAP - Document라는 방법을 제안한  SOAP 1.2 Spec이 나오게 되었습니다.

SOAP - Document는 Type이라는 Element에 사용자 정의 타입을 직접 기술함으로써,

밴더간의 Interoperability (상호 운영성)을 극대화한 모델을 제공하였습니다.

VS.NET에서는 SOAP-Document 형태의 Communication Style을 지원하는 Asp.NET Web Services라는

Project를 지원함으로써 좀더 높은 상호 운영성을 지원하였습니다.

프로젝트의 기본 Source 파일의 확장자가 .asmx이기 때문에, asmx라고 부르는 것이 관례가 되었습니다.

3. WSE (Web Services Enhancement)

SOAP - Document Model은 상호 운영성을 지원하지만, 진정한 Service로 사용하기에는 많은 부분이 부족합니다.

Security, Transaction, Messaging이 대표적인 예가 될 것입니다.

이러한 문제를 해결하기 위해 Web Services Interoperability Organization (Web Service 조약기구)에서 WS - \* Spec을 제정했습니다.

진정한 Service로 사용하기 위해서는 Security, Transaction, Messagine 또한 모든 밴더가 이해할 수 있는 표준이 필요했기 때문입니다.

이러한 표준을 준수하는 형태로 ASMX에서 Service를 만든다는 것은 아주 많은 양의 코드들이 추가 되어져야 할뿐만 아니라,

개발자들에게 .NET과 WS - \* Spec을 모두 깊게 이해해야 하는 문제들이 생겼습니다.

그래서, 좀더 WS - \* Spec을 준수한 Service를 쉽게 만들기 위해, VS.NET 의 Add-In 형태로 추가 되어지는

Web Services Enhancement라는 툴을 발표하였습니다.

자세한 정보는 <http://msdn.microsoft.com/webservices/building/wse/default.aspx> 를 방문하시면

WSE에 대한 좀더 많은 정보를 얻으실수 있습니다.

4. WCF

진정한 Service로 지원하기 위해서는 다양한 프로토콜, 다양한 기술 끊임없이 발전중인 WS - \* Spec을

기존의 분산 객체들보다 더 생산성이 높은 모델로 제공해야 할 것입니다.

WCF가 그러한 분산 객체입니다.  자세한 이야기는 이제부터 시작하도록 하겠습니다.

**3. WCF**



   WCF는 Windows Platform 상에서 Service Oriented Application을 개발하기 위한 통합된 Framework 입니다.

   1. 단일화

   - 오늘날의 다양한 분산 기술 Stack (P2P, RPC, IPC 등)들을 통합하였습니다.

   - .NET Remoting에서 보았던 Composability를 지원합니다.

   - WCF는 Communication이 필요한 부분에는 어디에서나 적합합니다. (local, remote, internet 등..)

   2. 상호 운영성

   -  Plaform간에 호환성을 가집니다. (WS - \* Spec을 가장 잘 지원하므로)

   -  기존의 Microsoft 분산 제품 (.NET Remoting, System.Messaging, Enterprise Service, ASMX, WSE)들을 통합하여,

      좀더 쉽고 단일화된 프로그래밍 모델을 제공합니다.

   -  오늘날의 분산 기술 Stack들을 통합하였습니다. (Message Queuing, Web Services, RPC 등)

   3. 서비스 지향적

   -  서비스 지향적인 프로그래밍 모델을 지원합니다. WS - \* Spec을 좀더 잘 지원합니다.

   -  WS - \* Spec에서 제안한 SOA의 모든 기능들을 다 제공합니다.

   -  Serivce Oriented Application을 개발하는데, Composabilti를 이용함으로써 최고의 생산성을 보장합니다.

**4. Microsoft  분산 기술을 통합한 WCF의 Feature들**



  미국 MSDN TV Show - Connected System 이라는 것을 다운 받아 보시면, WCF에 대한 Don Box의 애정과 철학을 느낄수 있습니다.

  WCF라는 유래는 WCF를 영문으로 파악하면, 파랗다 또는 남색의 의미를 가집니다.

  다양한 기술들을 모아놓은 모든 것을 다 받아들이고 수용하는 마치 물같은 느낌을 주기 위해 WCF라는 코드명을 사용했다고 합니다.

  또한 Don Box는 WCF가 다양한 기술의 집합체라고 말을 하고 있습니다.

  위의 그림과 같이 Message Queuing을 지원하는 System.Messaging, WS - \* 을 잘 지원하는 WSE, 조합성과 확장성이 뛰어난 .NET Remoting,

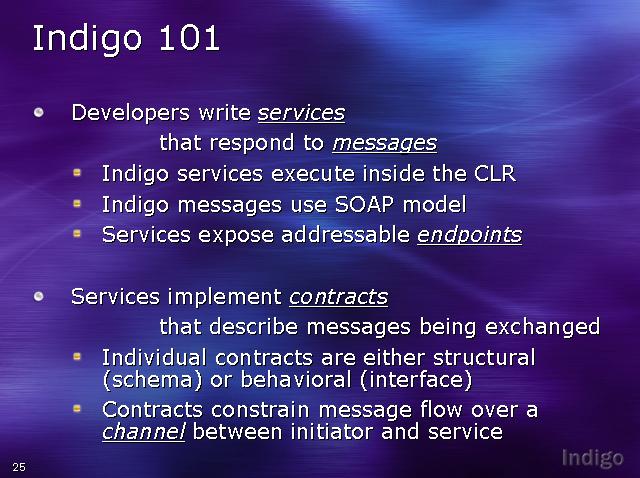
  SOAP - Document 모델을 이용함으로써, 호환성이 높은 ASMX, Attribute를 통해서 세밀한 기능을 제어할 수 있는 Enterprise Service 들의 각 장잠들을

  통합한 것이 WCF라고 할수 있습니다.

  예전에 ASMX에서 Message Queueing 썼을 경우, 두분야의 소스코드가 뒤죽 박죽 섞여서 사용을 했습니다.

  WCF를 이용하면 단지 몇줄로, 예전에 복잡한 기술의 조합시 생기는 복잡함이 감소했습니다.

**5. WCF 101**



 WCF는 ASMX와 달리 CLR 내부에서 실행 되기때문에 더욱더 안정적이며, 높은 성능을 보장할 수 있을뿐만 아니라.

 업게 표준의 SOAP을 Message 로 사용하기 때문에, 호환성을 극대화 하였고,

 기존의 ASMX는 하나의 endpoint만 노출 시킬수 있었지만, WCF는 서비스 하나당 여러개의 endpoint들을 노출 시킬 수 있습니다.

 WCF에서는 Contract(계약)의 개념이 매우 중요합니다.

 크게 세가지의 Contract (Service, Data, Message)가 존재합니다.

 마치 객체의 계약에서 public이 외부로 노출되는 것과 마찬가지로

**서비스로 노출될 Operation을 정의**할 수 있습니다. **(Service Contract)**

**데이터 영역 (Data Contract)** 또한, 역시 서비스에서 **사용될 일부 attribute만 외부로 노출** 시킬 수 있습니다.

 그리고 **Message Contract**라는 것을 이용하여 **주고 받는 SOAP 메세지의 구조를 정의**할 수 있습니다.

 계약에 대한 자세한 애기는 다음 강좌때 더욱 자세히 애기하도록 하고, 이정도의 개념만 잡도록 하겠습니다.

**6. WCF Architecture**



 WCF의 Architecture에 대해서 보도록 하겠습니다. 크게 4가지 Layer로 구분되어 집니다.

 1. Activation & Hosting Layer

 예전에 ASMX는 IIS 위에서 동작했던 것에 반해, WCF는 다양한 Activation & Hosting을 지원합니다.

 먼저 WAS (Windows Activation Service)는 차세대 운영체제 롱혼에 탑재될 서비스 로 IIS 7.0 을 독립된 Application이

 아닌 NT Service 같은 Service로 끌어 내렸기 때문에 WAS 라고 합니다.

 WAS를 이용하는 것이 WCF의 가장 효율성을 유지하는 방법이겠지만, Avalon, .EXE, NT Service, COM+ 에도 Hosting을

 할 수 있어 좀더 프로그래밍의 다양함을 제공합니다.

 2. Messaging Layer

 Service는 다양한 Binding을 제공해야 할  것입니다. WCF는 다양한 Bindiing을 지원하기 위해 Channel이라는 개념을 도입했습니다.

 우리는 단순히 Business Logic에만 신경을 쓰고, 이것을 TCP 로 보낼지, UDP로 보낼지, 보안을 적용하여 HTTP로 보낼지는

 Channel에게 맡기면 됩니다.

 Service를 선언할때, **단순히 어떠한 Channel을 사용하겠다는 것만 선언해 주면** TCP나 UDP의 복잡한 Stack을 우리가 모르더라고

 WCF의 **Transport Componenet들이 알아서 데이터를 변환하여 전송**합니다.

 위 그림의 Messaging Layer를 보면 Channel이 크게 두가지 분류로 나뉘는 것을  볼수가  있습니다.

 HTTP Channel, TCP Channel, Queue Channel 등 이 존재하는데

 HTTP Channel을 사용을 하겠다고 선택을 합니다. 그리고 보안을 적용하고 싶으면 위에 있는 Security Channel을 같이 선언해 주면

 HTTPS 형태로 데이터를 전송할 수 있습니다. 그러면 TCP 상에서 보안을 적용하려면, TCP Channel과 Security Channel을 결합만 하면

 됩니다. 이러한 기능을 Composability라고 합니다.

 3. Service Runtime

 Service Runtime은 Service의 Runtime시 속성들을 정의하는 Attribute들의 집합니다. (ASMX 에서는 WebMethod에 해당하는 것입니다.)

 Instance Behavior 는 Instacne를 Request가 들어올때 마다 만들건지, 아니면 singleton 형태로 만들어 하나의 인스턴스를 공유할 건지  등을 정의하실수 있습니다.

 그리고 Activation은 Request가 들어올때 Service를 Activation 시킬건지, 객체를 생성할때 Activation  할 건 지등을 정의를 하실수

 있습 니다.  성능 계수(Throttling)에 관련된 것들도 제어할 수 있습니다. Cahce의 Service의 결과를 몇초동안 Cache에 남길 건지등

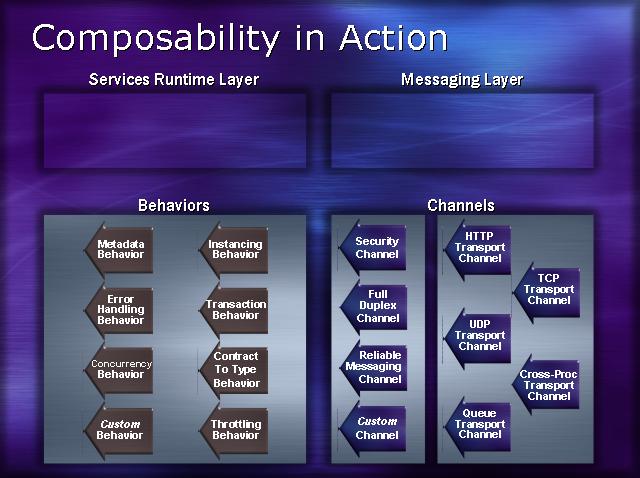
 다양한 성능에 관련된 설정 할 수 있습니다.

 이모든 것들을 Attribute로 제어하는 방법을 제공함으로써, 최고의 개발 생산성을 지원합니다.

 4. Contract

 위 단락 (5. WCF 101) 에서 애기한 것과 동일합니다.

**7. Composable WCF Architecture**



WCF의 Composable 한 Architecture에 대해서 설명 드리겠습니다.

이 그림은 그전 TP의 Message Layer와 Service Runtime Layer만 확대를 한 것입니다.

Transport Layer에는 크게 데이터를 전송하는 Protocol 쪽과 QoS를 담당하는 레이어로 나뉩니다.

그전 섹션에서 설명을 드린것과 같이 Transport Layer는 조합 가능한 레이어 입니다.

위 그림을 보면 왼쪽 상단에 HTTP Channel과 오른쪽에 Security Channel을 같이 조합하면

보안이 적용된 HTTP 상황에서 데이터를 주고 받게 됩니다.

만약 HTTP와 보안 그리고 Request - Response 모델이 아닌 양뱡향 통신으로 Service를 만들고 싶다면

HTTP Channel, Security Channel, Full Duplex Channel 세가지를 조합하면 됩니다.

이와 동일하게 Service Runtime도 역시 Composability (조합 가능성)을 지원합니다.

추가하고자 하는 기능을 단순히 Attribute 형태로 선언만 하면 그러한 기능들이 중첩되어 적용 되어 지는 장점을 가집니다.

**정리**

Windows Communication Foundation 라는 Microsoft 차세대 플랫폼에 대해서 소개를 하는 시간을 가졌습니다.

기술적인 내용 보다는 개념적이면 전체적인 아키텍쳐를 소개하는 시간이었습니다.

Service 를 좀더 잘 지원하기 위핸 Messaing Layer와 Service Runtime Layer의 Composability를 도입하였고,

Contract라는 부분에 초첨을 맞추어 다시 한번 생각하시길 바랍니다.

다음 강좌에 WCF에 사용되는 Contract에 대한 자세한 의미와 프로그래밍 데모를 집중정으로 보여드리도록 하며

부족한 글이지만, 많은 개발자에게 도움이 되었으면 하는 바램으로 이번 강좌를 마칩니다.

**1. WCF Programming**



WCF라는 분산 객체를 사용하기 위해서는 위와 같은 절차를 거쳐야 됩니다.

1. Service 생성하기

   Service라는 것을 만들기 위해 WCF에서 새로 추가된 Contract (계약)에 대해서 파악을 해보도록 합시다.

   Service와 Endpoint에 대한 개념에 대해서 파악해 보도록 하겠습니다.

2. Hosting 선택하기

   IIS 5.0, IIS 6.0에 WCF를 Hosting 할것인지?

   아니면 새로운 운영체제에 탑재될 Windows Activation Service 에 WCF 서비스를 올릴 것인지 선택을 해야 됩니다.

3. WCF Client 만들기

   ASMX (ASP.NET Web Service)에서 Client를 만들기 위해 wsdl.exe를 사용했습니다.

   유사하게 WCF의 Client를 만들기 위해서는 유사한 svcutil.exe (service utility)를 이용하시면 됩니다.

이러한 절차에 의해 WCF라는 서비스를 만드는 것이 이글의 주제입니다.

**2. Contracts (계약)**



WCF는 Contract(계약)이라는 방법을 이용하여 Service의 행위, 형태 등을 설정할 수 있습니다.

Data Contract는 Data를 표현하기 위한 구조적인 Schema를 나타냅니다.

만약 학생이라는 Data가 있다면 학생 이라는 데이터 형을 Xml Schema 형태로 나타내는 것을 의미하며,

WSDL의 Type 정보와 Mapping 되어집니다.

Message Contract는 SOAP 메세지의 구조를 정의합니다.

SOAP의 Header 와 Body의 구조를 Message Contract라는 것을 통해서 정의할 수 있습니다.

Service Contract는 주고받는 메소드를 정의하는 역할을 합니다.

ASMX 모델에서는 Web Service로 외부로 노출시키기 위해 WebMethod라는 Attribute를 선언해 주었습니다.

그것과 유사하게, Service로 노출시킬 Method를 정의하는 역할을 합니다.

WCF는 이러한 세가지 계약을 사용함으로써 Service를 만들수 있습니다.

이제 각각의 계약에 대해서 좀더 자세히 알아보도록 하겠습니다.

**2.1 Service Contracts (서비스 계약)**



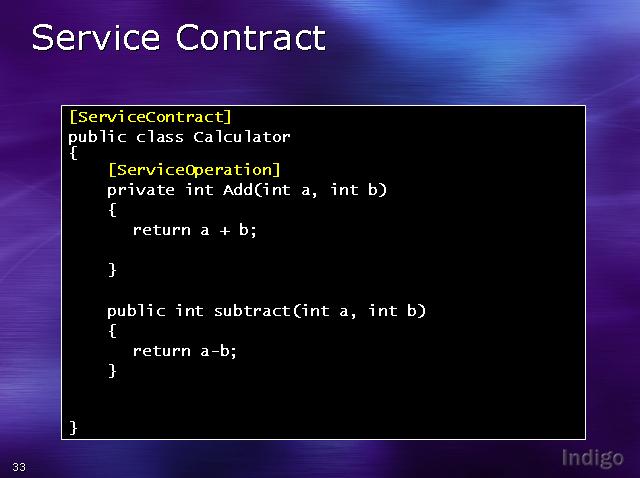
Service Contract는 주고 받는 메세지의 패턴을 정의합니다.

ASMX의 WebMethod와 유사한 기능을 한다고 생각하시면 됩니다.

Service중 외부로 노출 시킬 인터페이스를 정의하는 Contract라고 생각하시면 됩니다.

Contract는 Argument화 된 WSDL Port Type으로 맴핑되어 표현되어 집니다.

(이부분은 다음 강좌때 보여주는 데모를 보시면 이해가 가실 겁니다.)



자 그럼 WCF에서 어떻게 Service Contract를 선언하고 사용하는지 보겠습니다.

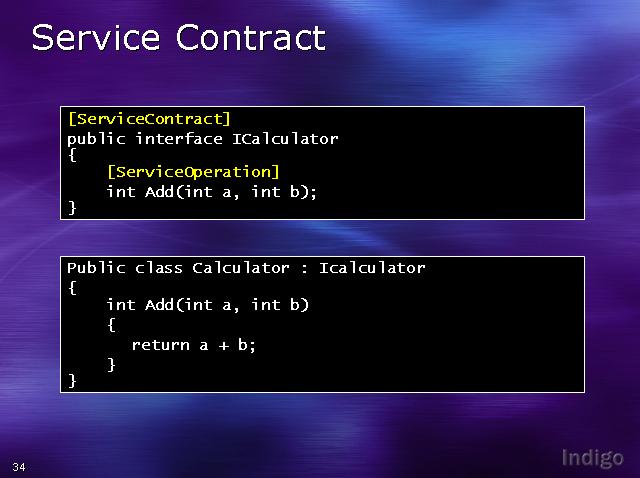
위 그림을 보시면 이전에 ASMX모델과는 별로 큰 차이가 없음을 알수가 있습니다.

WebMethod 대신 ServiceOperation이라는 키워드가 사용되고, WebBinding대신 ServiceContract가 사용되는 것을 보실수 있습니다.

우리가 조심해야 될 부분이 하나 있다면 객체의 Contract와 Service의 Contract는 다르다는 것입니다.

위 그림을 보면 객체의 관점에서 보면 Subtract가 외부에 노출되는 인터페이스가 되고 Add는 접근할 수 없는 Member Method가 됩니다.

하지만 Service의 관점의 계약에서는 Add가 외부로 노출시킬 인터페이스가 되고, Subtract가 Service로 노출되지 않습니다.



WCF는 Interface 상속이라는 개념을 통해서 Service를 만드는데 좀더 유연한 구조를 가집니다.

단순히 Interface만 만들고 상속 받는 클래스에서 Interface를 구현함으로써, Design Pattenr의 Template Method 같이

상황에 따라 Factory를 사용하여 변화에 유연한 형태의 Service를 제공할 수 있습니다.

추가 설명)

Template Mehtod Pattern (Template Method Pattenr에 대해서 잘 안다면 읽지 않으셔도 됩니다.)

Design Pattern에서 가장 기본이 되는 패턴으로, 추상화를 사용하여 확장성 있는 프로그램을 만드는 것이 목적입니다.

간단히 개념을 잡는다면, 부모 클래스와 자식 클래스 간의 관계를 생각해 보도록 합시다.

부모 클래스가 아무런 기능도 구현하지 않고, 인터페이스만 선언해 놓은 것을 C++ 에선 순수 가상 클래스, Java에선 Interface라고 합니다.

이러한 부모 클래스를 상속받는 자식 클래스는 반드시 부모 클래스의 순수 가상 함수(인터페이스 메소드)를 구현해야 되는 책임이 있습니다.

또 이러한 모델은 QoS를 지원하기에도 아주 합리적인 모델입니다.

똑같은 계산기를 만들더라도 저급 권한을 가진 사용자에게는 우선순위가 낮은 Calculator 서비스르 제공하고,

높은 권한을 가진 사용자에게는 우선 순위가 높은 Calculaotr 서비스를 제공할 수 있도록  Factory를 이용하여

Interface 지향적인 서비스를 만들 수 있는 기반 구조를 제공합니다.



마치 IDL 기반의 Component에서 본 것과 같이 in,out의 개념을 지원합니다.

추가 설명)

DOC (Distirbuted Object Computing)에서 데이터를 주고 받는 Communication 방법은 크게 세가지로 이루어 집니다.

1. At-Least-Once

클라이언트가 서버와 통신을 할때, 하나의 통신을 하기 위해서 하나의 메세지만을 보내는 가장 이상적인

방법입니다. 현실적으로 볼때, 네트웍 의 불안전한 상태나, 중간에 패킷에 대한 손실등이 발생할 수 있으므로,

실제적으로 분산객체에서 사용하기에는 비적합한 모델입니다.,

2. At-Most-Once

At-Least-Once는 너무 이상적인 모델로 실제 분산 객체 모델에서 적용하기에는 비적합한 모델입니다.

그래서 Client와 Server가 통신을 할때, 하나의 메세지를 전달하기 위해 네트웍 망의 이상, 패킷의 손실이

발생할 수도 있으므로 중복된 여러개이 메세지를 보냅니다. 가

그러면 서버쪽에서는 클라이언트가 보낸 여러개의 메세지를 받는 상황이 발생합니다.

그럼 어떻게 서버는 들어온 메세지를 보고, 중복된 메세지 인지 아닌지를 알수 있을까요?

그것은 Client가 하나의 메세지를 보내기 위해 여러개의 메세지를 보낼때 동일한 GUID(Asynchronous Completion Token)를 붙여 보냄으로써,

서버는 GUDI를 보고 처음 들어온 메세지 인지 중복 메세지 인지 파악을 할수 있게 됩니다.

가장 현실적인 모델입니다.

3. OneWay

Request만 있고 Response가 없는 서비스일때, OneWay를 사용하시면 됩니다.

그럼 Cient는 더이상 Response가 오는 것을 기다리지 않으며, 단순히 메세지만 뿌림으로써, Client의 과부하를 줄일수 있습니다.

void형 Service를 만들때 oneway를 사용하시길 권고합니다.

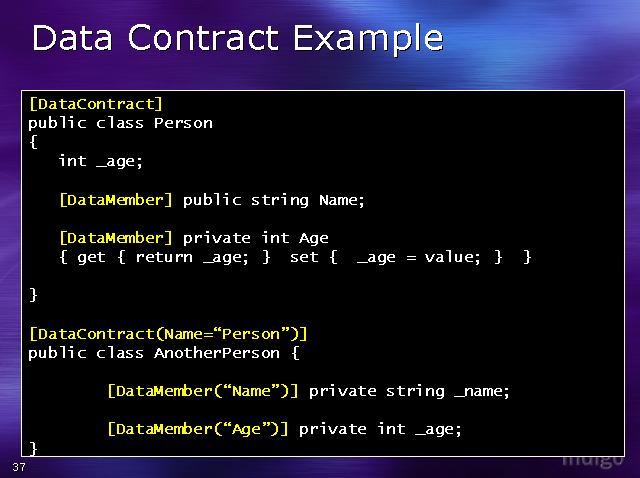
**2.2 Data Contracts (Data 계약)**



 Data Contract는 데이터를 어떻게 Serialization할 것인지를 제어하는 mechanism입니다.

 CLR Type들을 위한 Loosely Coupling을 지원할 뿐만 아니라. 다양한 버젼의 Data Type을 지원합니다.

 쉽게 말하면 Data를 위한 overloading을 지원한다고 생각하시면 됩니다.



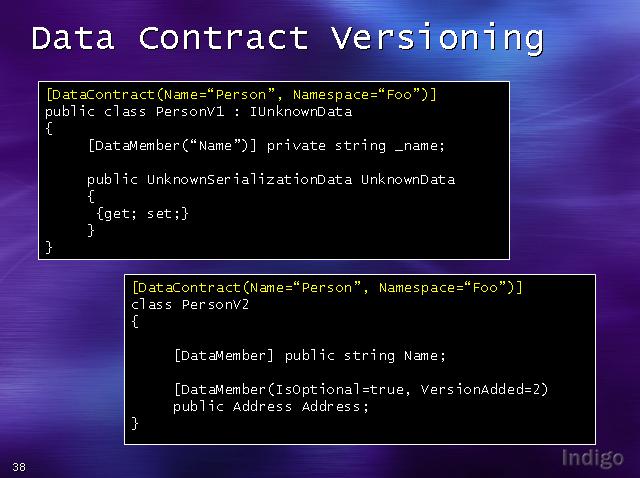
 Data Contract를 사용하기 위해서는 위와 같이 선언하시면 됩니다.

 Class위에 [DataContract]라는 Attribute를 선언하셔서 사용하시면 됩니다.

 DataContract 역시 외부로 노출시킬 맴버데이터들은 [DataMember]라는 Attribute를 줌으로써

 WSDL, Policy 생성시 외부로 Type Element에 맴핑 되어 질 것입니다.

 역시 WebMethod와 유사한 기능으로 외부로 데이터를 노출할때, 다른 이름으로 노출할 수가 있습니다.



또한 다음과 같이 다양한 Version을 지원합니다.  쉽게 말하면 Data를 위한 overloading을 지원한다고 볼수 있습니다.

Person이라는 자료형의 형식으로 Client가 데이터를 요청할대, Person 클래스 내부의 Attribute를 보고 적절한 DataType 형태로

unserialization 됩니다.

**2.3 Message Contracts (Message 계약)**

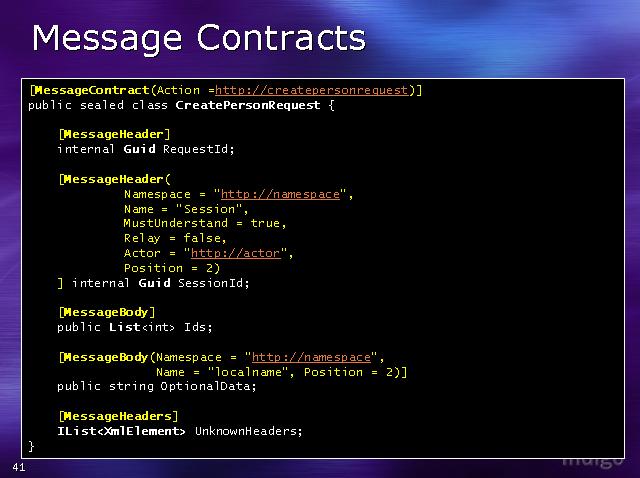


WSE에서는 SOAP의 구조를 정의하기 위해서는 WS-Policy는 이용하여 Custom Filter를 만들거나,

System.Messaging의 SoapSender라는 Class를 이용하여 SOAP릐 메세지 구조를 정의할수 있었습니다.

하지만 WCF에서는 Message Contract라는 것을 이용하여 SOAP Message의 구조를 정의하고 제어합니다.

Header와 Body를 구분하고 MustUnderstand와 Actor등 SOAP Spec에 지원하는 부분들을 직접 제어할 수 있습니다.



 Message Contract의 예입니다.  다른 Contract와 마찬가지로 간단히 Attribute만을 이용하여 SOAP의 구조를 정의할 수 잇으므로

 WSE 보다는 훨씬 쉽게 SOAP의 메세지 구조를 정의할 수 있습니다.

 크게  MessageHeader와 MessageBody라는 Attribute를 이용할 수 있습니다.

 MessageHeader 는 SOAP Header의 구조를 정의할 수 있으며 다양한 속성들을 이용하여 MU, Actor등을 간단히 셋팅할 수 있습니다.

 그리고  MessageBody는 SOAP Body의 구조를 정의할 수 있으며,  Namespace, Name과 Position등을 제어할 수 있습니다.

**3. Service와 Endpoint**

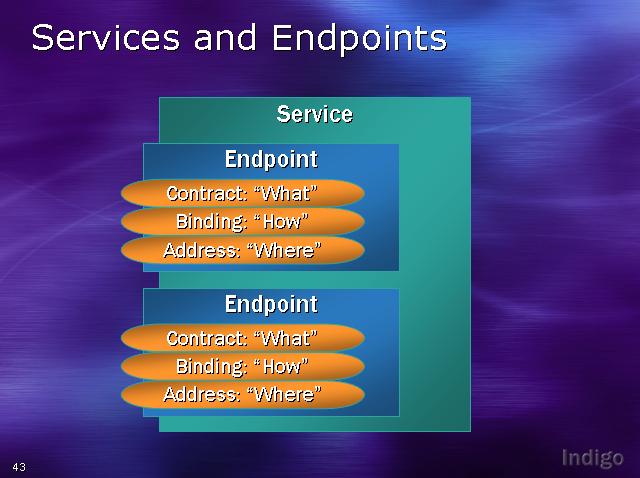


 ASMX 모델의 WSDL에서는 Service의 Endpoint를 여러개를 노출시키는 것을 직접적으로 지원해 주지 않았습니다.

 하지만 WCF에서는 여러개의 Endpoint들을 지원합니다.

 WCF에서는 런타임시에 특정포트에 서비스를 추가하는 것 까지 지원합니다.

 그 뿐만 아니라. Endpoint들에 정보도 ASMX에서는 생성시에 결정되었지만 ASMX는  attribute의 사용만으로 Endpoint를 제어할수 있습니다.



위 그림을 보면 Service는 다양한 Endpoint를 가지고 Endpoint는 Contract, Binding, Address로 정의 되어진 것을 볼수 있습니다.

Contract는 2장에서 본것으로 What의 개념을 지원합니다. 서비스를 사용하기 위해서는 IOPE 들을 어떻게 준수해야 돼는지

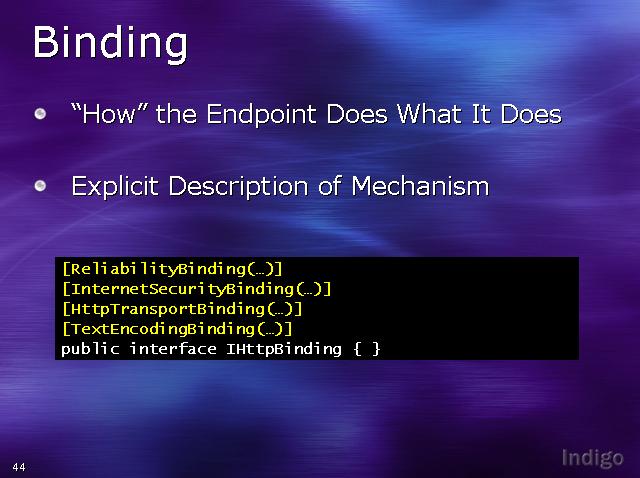
Contract에 기술합니다.

그리고 Binding 정보는 어떠한 프로토콜(또는 Channel)을 이용할지를 결정할 수 있으며,

Address를 통해 서비스에 접근하기 위한 물리적인 주소를 정의합니다.

그림 인디고에서는 어떻게 이러한 것들을 지원하는지 하나 하나 살펴보도록 하겠습니다.

**3.1 Binding**



첫번째  WCF 강좌였던 WCF Architecture에서 Channel을 이용한  Composability 기능에 대해서 기억이 나시는 지요?

바로 그 부분입니다. 위의 TP를 보시면 binding을 하나 선언하고 binding위에 사용하고자 하는 Channel을 Attribute로

설정해 주면 됩니다. WSE에서 보았던 복잡한 Filter 의 개념보다 훨씬 간결한 프로그래밍 모델을 지원한다는 것이

큰 장점일 것입니다.

**3.1.1 Transport Binding**

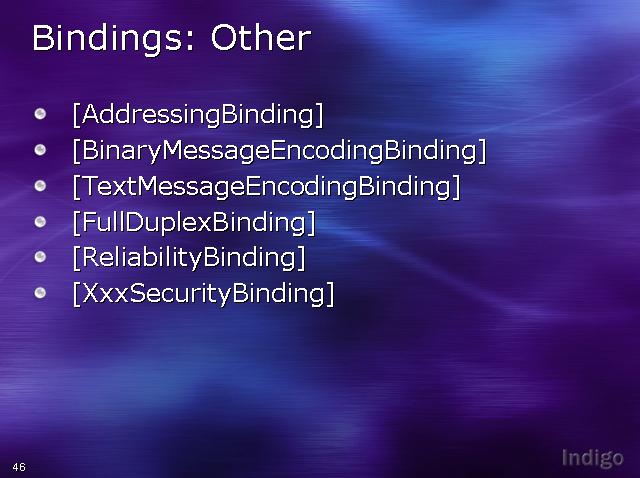


Transport Binding으로 지원되는 것은 위와 같습니다. HTTP를 지원하는 [HttpTransportBinding], HTTPS를 지원하는

[HttpsTransportBinding], MSMQ를 지원하는 [QueuedTransportBinding], TCP를 지원하는 [TcpTransportBinding] 그리고

Udp Multicast를 지원하는 [UdpMulticastTransportBinding]을 지원합니다.

**3.1.1 그외의 다른Binding**



Transport Binding외에도 다른 Binding을 지원합니다.

기전 강좌의 WCF Architecture를 보시면 왼쪽 화면에 TextEncoding, BinaryEncoding이 보이실 겁니다.

바로 [TextMessageEncodingBinding]과 [BinaryMessageEncodingBinding]이 그 역할을 해줍니다.

그 외에 양방향 동신이 가능한 [FullDupleBinding]등을 지원합니다.

신뢰성 있는 멧세지 전송을 위해 지원되는 [ReliabilityBinding]등 이 외에도 아주 많은 Binding을 지원합니다.

**3.1.2 app.config에 설정된 Binding 정보**

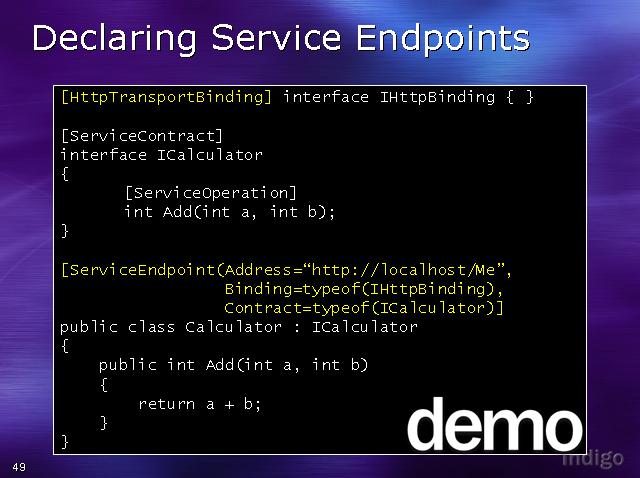


app.config 파일안에는 우리가 설정한 Binding 정보가 위와 같이 endpoint element안에 bindingType Attribute에 선언되어 있습니다.

VS.NET은 Reflection 기반의 Architecture를 지원하므로 metaobject에 해당하는 app.config 파일에 저장되어 있는 bindingType 정보만

바꾸어 주더라도 코드내에서 attribute로 선언한  binding 정보를 셋팅하는 것과 동일한 효과를 얻을수 있습니다.

**4. Service Endpoint 선언하기**



자세한 데모는 추후 동영상으로 업데이트해 드릴 것으로 약속 드리며, 간단한 전체 소스 코드를 보도록 하겠습니다.

첫 줄에 보면 Binding 정보를 셋팅하기 위한 interface를 선언했구요, Attribute로 HTTPTransportBinding을 선언합으로써

HTTP Protocol하에서 데이터를 주고 받을수 있습니다.

추상 클래스의 개념으로 생각하시면 되는 interface 형태로 ServiceContract의 Template을

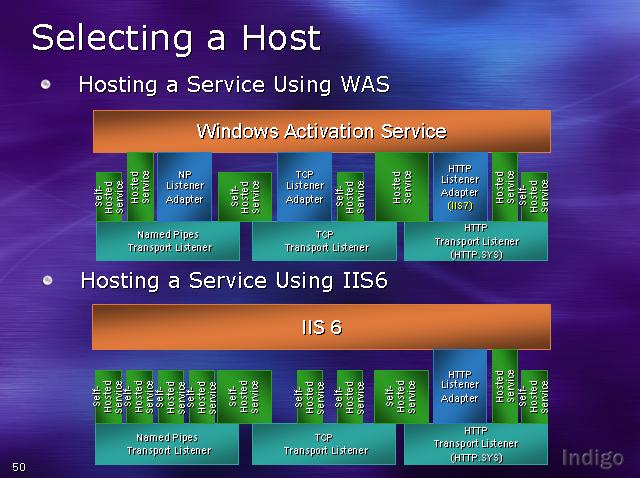
ICalculator를 선언하고 Service의 계약인 ServiceOperation을 설정한 것을 보실 수 있습니다.

Concrete Class에 해당하는 Calculaotr에 로직을 구현해 넣고, 위에 Attribute로 Service가 사용하는

Contract, Binding, 그리고 배포할때 사용하는 URL을 선언하면 됩니다.

이렇게만 선언하시면 간단한 WCF 서비스를 만드실 수 있습니다.

**5. Host 선택하기**



적절한 Host 서버를 선택한다는 것 또한 WCF 서비스의 성능 면에서는 아주 중요한 역할을 합니다.

WCF는 IIS5, IIS6, WAS(IIS7) 어디에서든 돌아갈 수 있습니다.

여기서 눈 여겨 볼것은 Longhorn에 새롭게 탑재될 WAS라는 것입니다.

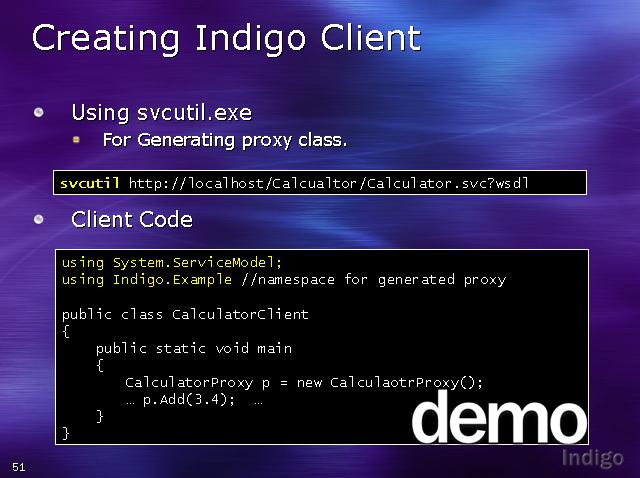
Windows Acitivation Service의 줄임말로 IIS 7.0을 예전 모델인 Application이 아닌

NT Service같은 Service 단으로 끌어내림으로써 여러가지 성능 개선을 가져온 서버를 위한 서비스라고 생각하시면 됩니다.

WAS를 IIS 6.0과 비교해 보면 TCP Trasnport와 Named Pipes에 관한 Adapter를 지원함으로써,

Windows Activation Service가 좀더 Service Oriented Archtiecture를 잘 지원하기 위해 Architecture를 강화했다는 것을 알수 있습니다.

**6. Client 생성하기**



ASMX는 wsdl.exe 라는 파일을 이용하여 Proxy파일을 생성할 수 있습니다.

우리가 많아 사용했던 add web reference 역시 내부적으로 wsdl.exe 파일을 이용한 것입니다.

WCF는 wsdl.exe와 동일한 역할을 하는 service utilitiy의 줄임말인 svcutil.exe 파일을 이용하여 proxy 코드를

생성할 수 있습니다.

사용방법은 console 모드에서 "svcutil  URL주소" 이런식으로 입력하시면 됩니다. 간단한 예제가 위에 있지요.

이렇게 service의 proxy 코드를 생성하구요.

clinet 코드에서 proxy 코드를 이용하여 사용하시면 됩니다.

ASMX와 유사하게 instance를 만드셔서 사용하면 됩니다.

실제 데모는 다음 강좌에서 동영상을 통해서 보여드리도록 하겠습니다.

**7. 맺음**

WCF라는 분산 객체를 사용하기 위한 Contract라는 개념과 실제 사용되는 예를 보았습니다.

WCF는 현재 <http://msnd.microsoft.com> 사이트에 다운 받을수 있습니다.

다음에 오프라인 세미나를 통해 더 멋진 강의를 해드리겠습니다.

조금만 기다려 주시길 바랍니다.

책에서는 WCF의 핵심적이고 기본적인 내용위주로 많이 배울 수 있었지만.. 실제 적용시에는 많은 어려움이 있었습니다.

1. 클라이언트(Proxy)와 서비스 간에 연결 시간 제한

  - 클라이언트 GUI 프로그램을 실행한 상태로 아침에 돌아와보니 서비스와 연결이 끊겨있었습니다. 그 때 서비스연결의 시간이 여기에 적용되는 구나라고 생각했고.. 이것을 해결하기 위해서 지속적으로 연결하는 방법이나 재 연결을 고민하다가 지속적으로 연결하는 방법을 찾아내서 해결을 하였습니다(강좌 & 팁 참고).

물론 이런 방법은 다수의 사용자가 사용하는 서비스인 경우.. 서비스의 리소스를 낭비하는 결과가 있긴 하지만.. 사용하는 사용자수가 많지 않은 경우 재연결 코드보다는 연결 시간 제한을 아주 크게 두는 방법으로 선택을 하였습니다.

2. Config 파일의 위치 문제

 - 보통 C# 애플리케이션은 app.config라는 파일을 기본적으로 자기 exe 파일위치에서 읽게끔되어 있어서 서비스와 관련된 config 내용을 app.config에 넣어두어야만 했습니다. 만약 접속하려는 서비스가 많은 경우 app.config가..지저분해지는 결과가 발생했습니다.

그리고, 여러 사용자 컴퓨터에서 서비스의 주소를 변경이라도 하면 각 컴퓨터마다 config를 수정해야하는 유지보수도 힘들었습니다. 그래서 원격의 공유 폴더에 이러한 서비스의 config들을 각 서비스별로 만들어 두고, 이를 원격에서 읽어 서비스를 런칭하거나 서비스에 접속하는 proxy에 사용하도록 구성을 해보았습니다. (강좌 & 팁 참고)

3. 서비스 종료시 에러 처리와 복구 기능

- 서비스로 접속 중인 상태에서 서비스가 종료되는 경우.. 클라이언트 쪽에서 예외처리를 해주어야만 했는데요.. 서비스가 종료되면 클라이언트에서는 서비스의 호출을 실패하게 됩니다. 그래서 서비스를 호출하는 부분마다 try catch로 처리하니 일단 클라이언트에서 안전한 에러처리가 가능하였습니다.

그리고 Proxy에 해당하는 부분을 Agency라는 클래스로 감싼 후에 자동으로 재접속하는 코드를 구현하였습니다. 이렇게 함으로서 서비스의 재시작시 자동으로 클라이언트에서 재연결을 함으로써 서비스의 기능을 지속적으로 이용할 수 있게 되었습니다.

4. Proxy와 Callback 코드의 재사용성

- 서비스에 대한 proxy와 CallBack 코드는 이 서비스를 이용하는 다른 클라이언트 프로그램마다 동일하게 사용되는 코드입니다. 그래서 하나의 서비스에 대한 proxy와 CallBack을 구현하여 하나의 Dll로 만들어 서비스를 사용하는 곳에서 자유롭게 가져다 사용할 수 있도록 구현하였습니다. CallBack 클래스의 경우 클라이언트 프로그램과의 결합성을 떨어뜨리기 위해 이벤트로 처리하여 외부로 알리는 형식으로 구현하였습니다.

5. 클라이언트 비정상적인 종료시 서비스의 콜백 레퍼런스 삭제

- 배포 구독의 구조로 구성된 상태에서 클라이언트가 비정상적으로 종료되는 경우 서비스의 콜백 레퍼런스 리스트에서 삭제시켜야 하는 문제가 있었습니다. 콜백 레퍼런스 리스트에서 삭제를 하지 않는 경우 연결되어 있지 않은 클라이언트에게 콜백을 전송할 위험도 생기고 부하도 발생하기 때문입니다.

이 비정상적인 연결 종료에 대한 이벤트는 여기 게시판의 질문/답란을 통해 알아냈고. 이 이벤트가 서비스에 발생시 해당 클라이언트에 대한 콜백 채널을 통해 콜백 리스트를 관리할 수 있었습니다.

6. 데이터 계약의 공유 문제

- 아시다시피 SOA개념을 지키려면.. 오직 Proxy에 선언된 데이터 계약으로 사용해야 합니다. 하지만 저의 프로젝트 환경에서는 문제가 있었습니다..

데이터 클래스 정의시 개발과정에서는 빈번하게 수정이 발생함으로써 매번 Proxy 코드를 생성하는 부분이 상당히 부담스러운 작업이었습니다.

그리고, 데이터 클래스를 정의 한 후 서비스1, 서비스2에서 사용하는 경우 서비스1, 서비스2에 대한 Proxy 코드를 생성하게 되면 같은 데이터 클래스가 포함됩니다. 이 두 서비스에 접속하는 클라이언트에서는 중복으로 선언된 코드가 포함되므로 에러가 발생합니다.

이러한 상황을 해결하고자  먼저 데이터 클래스는 별도의 dll로 구성하여 작성하였습니다. 그리고 서비스에 대한 proxy 코드 생성시 proxy 코드내에 정의된 데이터 클래스를 삭제하고 인터페이스 부분만 proxy 코드로 남겨놓았습니다.

클라이언트에서는 데이터 계약에 대해서 데이터클래스 dll을 참조하게 함으로써 데이터 클래스가 변경되더라도 proxy를 매번 생성할 필요도 없게 되었습니다. 그리고, 서비스1, 서비스2에 대한 서비스 요청시에도 동일한 namespace를 갖는 데이터클래스를 사용할 수가 있었습니다.

물론 이렇게 하는 것이 여러 분산화된 서비스의 SOA 개념에는 위배되지만 운영되는 소프트웨어는 저희 개발팀에서만 구현하기 때문에^^

두서없이 줄줄 쓰느라 이해하기 어려운 부분도 있겠지만.. 비슷한 상황에 놓이신 분들을 위해 써보았습니다..