Chapter

10

Service Oriented Architecture

CHAPTER AUTHORS

Goh Chun Lin

Koh Eng Tat Desmond

Naing Tayza Htoon

Nguyen Van Thuat

CONTENTS

1 INTRODUCTION TO ENTERPRISE SYSTEMS

Definition of Enterprise Systems (ES)

"Enterprise systems feature a set of integrated software modules and a central database that enables data to be shared by many different business processes and functional areas throughout the enterprise."

- *Essentials of Management Information (8th edition)*

An example of enterprise systems is an Enterprise Resource Planning (ERP) system used in many large organizations.

Given that enterprise systems are often large-scale, there are many critical requirements in a typical enterprise system:

Availability

The system is supposed to be up and ready at any given point in time. Sometimes, there could be a sudden increase in user demands. If the system cannot handle the load, it will result in a system downtime and can impact the company's business performance. Hence, availability of enterprise system is critical because the system is supposed to be supporting 24x7 services.

Scalability

Globalization was driven by the development of technology and because of globalization, technology marches forward. The way that businesses are handled and operated is changing too. Hence, globalization has actually caused the rapid growth and dramatic changes in many organizations. Enterprise systems thus must have a flexible architecture, which can reply to the fast changes that often happen in the organizations. The systems need to have the scalability feature in order to adopt the new changes happening in the organizations.

Performance

Organizations and enterprises invested heavily in their enterprise systems which are meant to improve their business workflow, data control, client management as well as customer responsiveness. The expenditure in enterprise systems grows and eventually turns into a significant part of the total business cost. This has led to the fact that the shareholders and managers care very much about the performance of enterprise systems used in their organizations.

Security

Security is always one of the most important requirements in enterprise systems. The systems have to be secured in order to ensure the continued system availability and data confidentiality.

Manageability

A large percentage of enterprise systems fail mostly because of their high complexity which leads to the fact that the systems are not easy to be controlled and managed. In this case, the Enterprise Systems Management (ESM) specialists have to monitor the system operations as well as the performance in order to track down the source of problems and then pinpoint and fix the problems in the underlying layers.

Data Integrity

One of the primary design considerations for the enterprise systems is data integrity. Data integrity means that data in the systems should not be lost or corrupted.

Interoperability

Interoperability is the ability of enterprise system (or any general IT system) to use information and functionality of another system. One of the examples will then be data exchange between two systems. Interoperability is a key to building enterprise systems with mixed services. It helps in achieving improved efficiency of the system operations. Hence, enterprise systems are required to be interoperable so that they can collaborate with other systems.

1.1 Current demands of enterprise systems

Enterprise systems are important in today's business world. For example, For example, ERP applications support organizations in their day-to-day operations such as order management, client management, finance, manufacturing, and logistics.

The IT systems used in organizations are required to adapt rapidly to the changing environment in the organizations. As an efficient and effective solution to that, enterprise systems receive high demands from the industry.

In addition, in the job market, there is an emerging engineering discipline known as Enterprise System Engineering (ESE). The main focus of ESE is to apply the engineering principles and methodologies to the design and management of the enterprise systems. Also, enterprise system engineers are required when the entire system has high complexity due to its scale, some uncontrollable interdependencies and other uncertainties. Other tasks that an enterprise system engineer needs to perform will be, for example, infrastructure design, security design, server consolidation as well as the deployment of the system. Hence, the understanding of underlying architecture and platform available for building enterprise systems, which will be introduced in the later part of this chapter, is critical. Thus, IT experts with related knowledge and skill sets are what big organizations are always looking for.

1.2 Technologies

Two of the technologies commonly used for implementing enterprise systems are Java Enterprise Edition (Java EE) and .NET. Hence, in the remaining sections, we will be focusing on Java EE and .NET.

Java EE, currently in version 6, is an extension to Java Standard Edition (Java SE). It provides additional APIs which are not available in Java SE. The APIs are mainly for the development of Java enterprise applications.

In addition, .NET will also be used as the platform to illustrate the ideas of web services implementation. There is an API from .NET which is known as Windows Communication Foundation (WCF) will be the focus.

1.3 Future trends of enterprise systems

The development and maintenance of enterprise systems currently have several issues and challenges. For example, enterprise systems are traditionally deployed on-premise because the enterprise wants to have control over the whole system and data and those corporate data should be stored internally. However, this requires the enterprise to either buy the necessary hardware, such as servers, themselves or to pay the hosting service. This is usually costly and the whole deployment process is not trivial as well.

Hence, in the recent years, more and more businesses, especially those who cannot afford big IT infrastructure or hardware, have chosen virtualization and cloud as their solutions to have cost-effective enterprise systems.

Virtualization is the creation of a virtual version of hardware, platforms, devices, and so on. Because of the virtualization, utility computing is possible. Computer processing power can be seen as a utility which the enterprises can pay only when they use it.

In order to reduce the costs of enterprise system development, startups and small businesses also move their systems to the cloud. This actually allows them to have a shorter time and faster speed on rolling out new services because they do not need to worry problems, such as hardware maintenance, network resources scaling, which can now be easily configured on the cloud. Also, since most of the enterprise data and resources are on the cloud, resource sharing is much easier and more efficient as compared to the past when the data is stored internally.

2 SERVICE ORIENTED ARCHITECTURE

Most ES follows an architectural style called Service Oriented Architecture (SOA). For example, several of the largest IT companies in the world such as IBM, Oracle, HP, SAP and Microsoft use SOA in their ES solutions offered to clients.

2.1 Challenges in enterprise systems

First, let us look at some of the challenges SOA tries to overcome.

2.1.1 Isolating business logic

In building business applications for large organization, it is often complex and involves a huge amount business logic. Business logic simply refers to business rules that are most likely imposed by non-IT folks within the organizations. For instance, in a purchase order system, a business rule might be that purchases that exceed a certain pre-determined amount need to be manually approved by the finance manager before being dispatched down the line. To IT folks building the purchase order system, not only they have to have to incorporate these business rules, which can be thought of as 'human aspect' of how to do things, they also have to be concerned about "computer logic", things like "checking if the database connection is available".

The biggest problem in programming is often it is very difficult to keep the business logic separated from the so-called "computer logic" as mentioned above. What makes matter worse is that these non-IT folks can change the business logic any time, without understanding how a small change could result in possibly disproportionate amount of work required by the IT folks to implement the change. In the next section, we will see that in Service Oriented Architecture, implementing business changes logic is no big deal.

Separation of Concerns

"Separation of concerns and is a software engineering best practice that should be applied in the design of all technology systems intended for business users. Unfortunately, this best practice has been observed more in theory than in practice. If you discuss this issue with software engineers, you may hear many excuses. The separation of concerns is often ignored simply because it takes effort to abide by it, and the costs of ignoring it are all in the future — in other words, too often, "quick and dirty" wins out over "slow and sure." Creating a reusable architecture takes discipline. And discipline inevitably takes more time than you'd ever expect to establish itself. Management may need to be educated. The upfront costs of establishing and requiring discipline pay manifold dividends over time."

-Service Oriented Architecture for Dummies

2.1.2 Interoperability

As we can see in the beginning of the section, just as interoperability is a requirement of an enterprise system, it is also a challenge at the same time. The ideal scenario is that there is homogeneity in the systems used across the organizations. Again that is simply hard or not even possible. Even if it is, in the context of the business world, the company might acquire another company that uses a totally different IT system altogether. The result is that additional work has to be done to allow interoperability. This can yet introduce another problem associated with the reluctance to migrate or upgrade existing system. Having invested effort and resources to allow interoperability, migrating to a new system might pose challenges if it is not compatible with existing system.

2.1.3 Software silos

In the physical world, a silo is a robust structure meant to hold and contain things to prevent what is outside from getting in and what is inside from getting out. The problem is that many applications and IT systems end up becoming such silos. The term usually refers to systems that cannot communicate with other related third party systems. Information flow is usually vertical. As the business expands or business requirements change, the end result is that there are a cluster of siloed systems that cannot cooperate or work with one another.

2.1.4 Redundancies

A problem that is common to many large companies is that there are many similar yet slightly different applications that are used throughout the organization. Each department usually comes out with its own version of software components rather than coordinate with other departments to see if component reuse is possible. The latter turns out to take too much effort as such it usually involves chores such as going through rounds of inter-departmental meetings to determine the common functionalities amongst the different systems and what or what not to be included in the system.

While companies might have policies or guidelines for such scenarios, often when deadlines are tight, or due to budget issues, it is often more convenient to build the application the department needs rather than coordinate across divisions. The problem can surface again when one company acquires another and realise that they too have similar applications with the similar functionality.

Another issue with such redundancies is the increased effort and complexity to maintain such applications. Any change in business policy will probably render these applications obsolete. All updates will then have to be propagated through these instances of the application. Again in the context of enterprise systems, where the problem is magnified, this translates to higher cost of IT costs and inefficiencies, something that is not desirable.

2.2 Service oriented architecture defined

While there is no standard or official definition for SOA, we found that IBM's definition of SOA (given below) an adequate one in the context of this section:

Service Oriented Architecture (SOA) is a business-centric IT architectural approach that supports integrating your business as linked, repeatable business tasks, or services.

It is important to note that SOA is not a product but is an architectural style.

2.2.1 Motivating example

Consider a catalogue application that is able to retrieve product information from a database. This application might be used by a department that need to keep updated about the latest products and pricing offered by the company. Furthermore, as it is a multinational organization, the database might reside in a different country. Hence local currency conversion also needs to be performed based on where the application is run. A typical design of such application is shown in Figure 1.

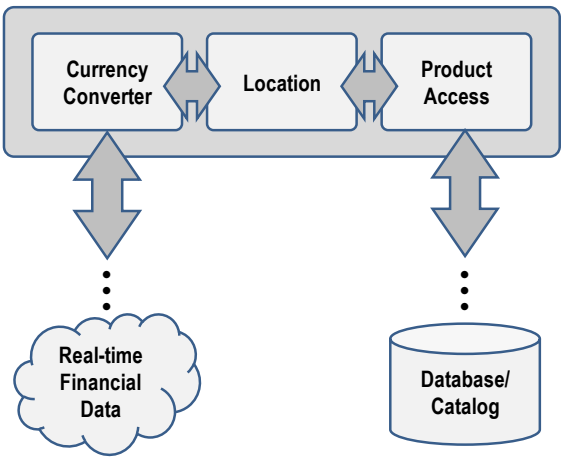


Figure 1: Typical design of a catalogue application

This design has to deal with the following challenges:

Interoperability

The application has to get real-time financial data to perform local currency conversion as well as interface with a database server to retrieve product information. As the two systems are probably different, the developer has to take care of interfacing.

Redundancies

The interfacing component required might have already been developed by other divisions of the company, probably even in other country. However, chances are that it will be unknown to the current developer and the same work is duplicated. Another hurdle in sharing components is that different departments across the organization might not be using similar technology in developing their application and hence increased the difficulty in component reuse.

Isolating Business Logic

In the example of accessing the database, the developer needs to know the database schema, what table to query, how to construct the SQL query etc., the usual chore that distracts the developer from the real work of implementing the application and business logic. Consequently, when the database schema changed, the application breaks and the data access component need to be updated accordingly. The updated component will have to go through unit testing, integration testing etc. before it is deployed. Again these chores are duplicated across the organization. In general, the typical architecture of a non-SOA system will similar to Figure 2.

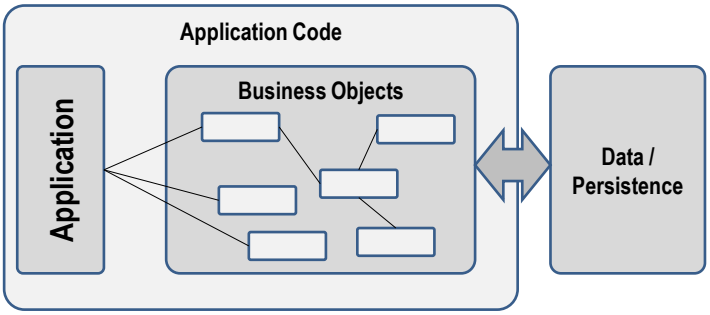


Figure 2: Typical architecture of a non-SOA system

2.2.2 SOA to the rescue

With SOA, the business logic is decomposed into well-defined and reusable services which will be exposed for everyone to use. Now all the application has to do is to consume them. As such, the architecture will be transformed into something similar to Figure 3. Now the application code is reduced greatly. Furthermore, it is no longer needs to traverse complex objects hierarchy and the developer no longer needs to understand details of domain-specific logic.

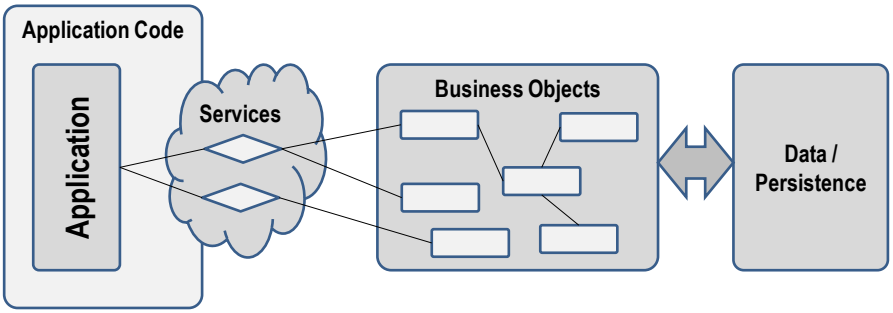


Figure 3: Improved architecture of system with SOA

SOA exposes business functionalities as services to be consumed by applications so that developers have fewer things to worry about. The services are in fact a form of abstraction, and we are familiar with the benefits that come with abstraction. For comparison, our application introduced earlier might now have the following design shown in Figure 4 Figure 4if SOA is adopted.

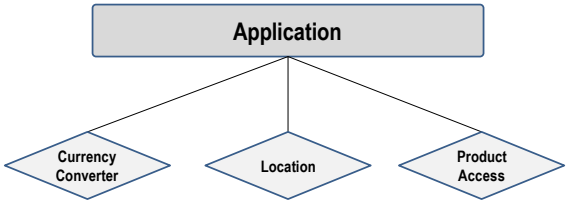


Figure 4: Improved design with SOA

Having introduced the notion of services, let us look at different types of services.

2.2.3 Types of services

There are several types of services used in SOA systems.

- Business services
- Entity services
- Functional services
- Utility services

Business services

Business service can be defined as the logical encapsulation of business functions. It has to be relevant to the business of the organization is running. An easy way to determine whether a service is a business service is to ask whether the service can be created without the consultation of business managers. If not, the service isn't probably a business service.

Another desirable feature of a business service is that it should have as little dependencies as possible so that it can be reused easily throughout the organization. This reusability means that there is consistency. In addition, any change in business policy can be propagated throughout the organization much more easily.

While the concept of reusability might already be familiar in the world of software engineering, in SOA the level of reuse is different. The concept of reusability in SOA refers to reusable high-level business services rather than reusable low-level components.

In view of the above discussion, it is indeed by no means easy to identify appropriate business services in a SOA. It involves both the IT and business departments to do that. Nevertheless, it is an important step as defining business services is important to building a strategic SOA.

Business services are not the only services in SOA. A typical service model might include Entity Services, Task/Functional Services and Utility/Process Services.

Entity services

An entity service usually represents business entities (e.g. Employee, Customer, Product, Invoice etc.). Such entity service usually expose CRUD (create, read, update, delete) operations.

Functional services

Functional services do not represent business-related tasks or functions. Rather it usually can be represented in a sequence diagram. In other words, it is usually a technology-oriented service and not a business oriented one. Task services can be thought of as controller of composition of services and hence its reusability is usually lower.

Utility services

Utility services offers common and reusable services that are usually not business centric. They might include event logging, notifications exception handling etc.

2.2.4 Service composition

A key concept in SOA is service composition. This refers to putting together several different services to create a new service. In that sense, services in an SOA environment can be thought of as building blocks. Service composition can only be achieved if services have a narrowly defined scope i.e. they do just 'one thing'. Again this is related to the idea of reusability.

2.3 Web Services

Web service is a realization of SOA. It is important to note that the SOA is an architectural model that is independent of any technology platform and Web Services the most popular SOA implementation. As the name implies, web services offers services over the web. This is not surprising as the choice of the Internet it already connects many different systems from all over the world. In this section, we will give a brief description to web services and introduce various web service terminologies.

2.3.1 Hypertext transfer protocol [HTTP]

As mentioned earlier, HTTP is a widely accepted standard that is implemented in many systems and operating systems. Hence it is able to address the issue of interoperability. By building web services on HTTP, all the computers that are able connect to the internet can become potential consumers of web services. Furthermore, by using the HTTPS protocol, web service communication can be secured.

2.3.2 Extensible Markup Language [XML]

Having decided on the protocol to communicate over, we need to decide on the language used to communicate. XML was chosen as it is a platform-independent language that can be understood by different systems.

2.3.3 Web Services Description Language [WSDL]

In programming languages, in order to call a function, one needs to know the method signature. In web services, WSDL is analogous to such method signatures. A WSDL document is written in XML, so any web service consumer can understand it and invoke the service. WSDL typically includes where the service is located, the functions/methods available for invocation, parameters and its data type as well as the format of the response.

```
<s:element name="ConversionRate">...
    <s:element name="FromCurrency" type="tns:Currency" />
    <s:element name="ToCurrency" type="tns:Currency" />
</s:element>
```

The XML above shows a possible excerpt of a WSDL document which defines a *ConversionRate* method that requires two parameters, namely *FromCurrency* and *ToCurrency* of type *Currency*.

2.3.4 SOAP

The next step after knowing the method signature is to invoke it. In web services, it is accomplished through SOAP messages. SOAP traditionally stands for Simple Object Access Protocol. However, from SOAP 1.2 onwards, the acronym is dropped as the name does not really represent what SOAP is. A SOAP message again is written in XML and sent to the web service over HTTP for web service consumption. The web service consumer or client will be able to construct the correct SOAP messages solely based on the WSDL document. Similarly, the response will also be in SOAP format. An excerpt of a SOAP message is given below.

```
<soap:Body>
  <ConversionRate xmlns="http://www.webserviceX.NET/">
    <FromCurrency>USD</FromCurrency>
    <ToCurrency>SGD</ToCurrency>
  </ConversionRate>
</soap:Body>
```

2.4 SOA benefits

Having a brief understanding on the basics of an SOA approach of building enterprise system, we will now list out some of the advantages of SOA.

2.4.1 Ability to build business applications faster and more easily

This is based on the assumption that the business services have been identified correctly. As such all business applications have to do is just to consume the correct services. The application code will be lesser, and the developer has fewer things to know and worry about as many of the plumbing now happens behind the scenes. Lesser code also means easier testing, and the application development process gets shortened.

2.4.2 Easier maintenance / update

This benefit follows easily from the previous one. Less code is easier to maintain. Moreover, as consumers of web services, it will not be affected by changes in implementation of web services. Say for example if a new database is added to the data, the web service will just include information from the new database in its response without the developer having to do a single thing. At a higher level, if a business process is modified, the equivalent business service can be recomposed to adapt to the changes. In addition, the change will be consistent throughout the organization.

2.4.3 Business agility and extensibility

Bearing in mind in an enterprise context, business environment is rapidly changing. How fast an enterprise system is able to react to these changes has important consequences to an organization. Service composition plays an important part in this aspect. The agility of enterprise system is demonstrated say when the requirements of a composite service changes, all that needs to be done is to replace relevant constituent services in order to update the composite service. Extensibility comes in when a totally new business service needs to be implemented, all that needs to be done is to assemble relevant services that already exists.

2.4.4 Lower total cost of ownership

All the benefits above translate to lower cost of ownership of IT infrastructure. This logically follows from reusability of services. Not only the service is reused, the IT infrastructure supporting these services is also being reused. Another cost savings comes from the fact that the shorter time-to-market of business applications also translates to better returns on the investment of IT infrastructure.

3 SOA IMPLEMENTATION IN JAVA EE 6

In section 2, we have discussed the principle of Service Oriented Architecture and the benefits it brings forward to the enterprise environment as well as the role web services play in the architecture. In this section, we will cover how web services can be realized using Java, one of the most widely-used enterprise technologies. There are several web services implementation in Java technology such as Axis2 and CFX from Apache, Spring Web Services, JBossWS and Glassfish Metro. However, we will only discuss Metro, a reference implementation of Java EE web services technologies.

Metro web services stack is fully supported in Glassfish server which is also a reference implementation of Java EE specifications. It mainly consists of two components: Java API for XML-based Web Services (JAX-WS) and Java API for RESTful Web Services (JAX-RS). Our emphasis will be on the former rather than the latter whose data exchange could be JSON, XML or any other data exchange protocol and whose operations are mainly in the form of HTTP methods such as GET, PUT, POST, or DELETE.

3.1 JAX-WS and related specifications

JAX-WS is designed specifically for SOAP-based web services in which data exchange is in the form of XML. Partial implementation of the stack can also be found in Java Standard Edition (Java SE) with the utilities to consume and deploy web services. Other important specifications for web services are also described below. All of these APIs provide necessary tools for the developers to consume and publish web services without worrying about the technologies underneath – SOAP, XML, HTTP and WSDL.

Java API for XML-based Web Services [JAX-WS] JAX-WS is a set of APIs that enables us to consume and create web services using Java programming language. It deals with all the processing required to send and receive SOAP messages either at the consumer or at the provider end by hiding the complexities of the underlying protocols. It also handles generating WSDL and parsing SOAP messages at the lower level leaving consumers and providers focus more on business logic as Java classes at the higher level.

Java API for XML bindings [JAXB] This API is responsible for mapping between XML and Java objects/classes. It is closely coupled with JAX-WS though it can be used independently to handle XML. As web services exchange XML messages between service providers and consumers via SOAP requests and responses, the API ensures that these XML documents are correctly mapped to Java objects/classes so that developers can make use of them without having to worry about the underlying XML.

Web Services meta-data These are the specifications that simplify the development of web services in Java. Through the annotations, they handle mapping between WSDL and Java classes in the definition and development of web services.

SOAP with attachments API for Java [SAAJ] SAAJ provides a standard way to send XML documents using Java. As the name suggests, it enables us to send attachments as part of SOAP messages according to the specifications. Besides, it is an API for handling these messages at a much lower level. Typically, we can use the API to create the messages without having to deal with the underlying XML directly and also extract information from response messages even though it requires more effort from the part of the developer.

3.2 XML binding in Java

Web services are described by documents in WSDL which is written in XML and data is exchanged between service consumers and providers via SOAP messages which are also in XML. However, we deal with Java classes and objects at higher level and thus web services require

binding or mapping between these two different models and this is what JAXB provides for us. Its operations can be divided into two different types.

First, JAXB facilitates conversion between XML schemas, described by the standard XML Schema Definitions (XSD), and Java classes. For example, web services are written in Java classes that need to be described in WSDL documents. In this case, the XML binding API is responsible for generating XSD schema from the Java classes. On the other hand, service consumers need to consume to web services by subscribing to WSDL documents. If service consumers are Java clients, it produces Java interfaces and classes from WSDL documents that can readily be used in our applications. The figure below sums up this process that often happens during development.

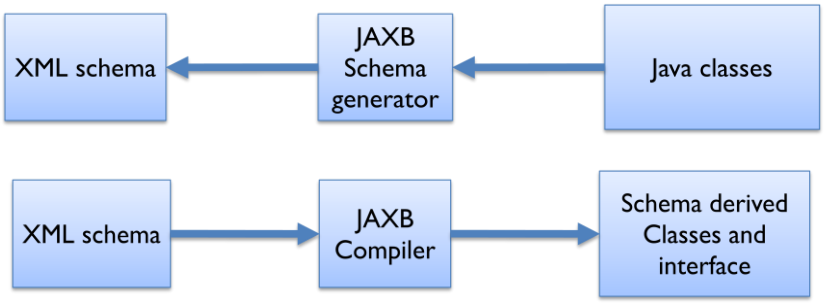


Figure 5: XML binding in Java

JAXB is also involved in conversion between XML documents and Java objects and vice versa during runtime. At some point in time, either at service consumer or service provider end, it is sometimes necessary to pass Java objects as we often do in normal Java applications. In this case, the Java object at one end is transformed into the corresponding XML document by the API to be carried over via SOAP messages and when it reaches the other end, it is converted back to the Java object which can be used as a normal Java object. This whole process is called marshalling and un-marshalling XML documents.

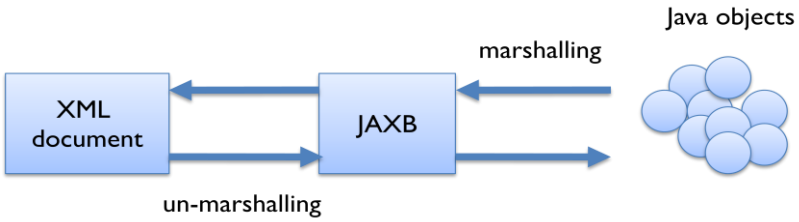


Figure 6: Marshalling and un-marshalling XML document from/to Java objects

Following is the table with some of the default bindings. You can override the default mappings by a custom binding declaration which is beyond the scope of this section.

XML Schema Type	Java Data Type
xsd:string	java.lang.String
xsd:positiveInteger	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte

For example, the following class in Java is converted into a corresponding XML schema.

```
public class Currency {
    private String currencyCode;
    private String country;
    private double conversionRateWithSGD;
    public Currency() {}
    public Currency(StringcurrencyCode, String country, double
conversionRateWithSGD) {
        setCurrencyCode(currencyCode);
        setCountry(country);
        setConversionRateWithSGD(conversionRateWithSGD);
    }

    ... getter and setter methods here ...
}
```

3.3 Consuming Web Services using JAX-WS

Consuming web services in Java is relatively straightforward. If you are using IDEs like NetBeans and Eclipse, you just need to add the web service you want to subscribe to your project. This is as easy as entering the URL of the WSDL document in appropriate settings. The IDE will automatically generate the proxy classes – called artifacts which are client-support

code – in a package or folder and all you have to do is to make use of them in your application logic. NetBeans, in particular, generates code templates also via simple drag-and-drop operation by the user.

Yet behind the scenes, these IDEs are using utility called "wsimport" which is shipped together with Java SE. Following is the usage of *wsimport* utility.

C:\>wsimport –p somepackage http://www.somewebservice.com/service?wsdl

Running *wsimport* in command-line (Windows) assuming that your machine already has a JDK, you will see that in your current directory a folder by the name of *somepackage* is created that contains the artifacts or proxy classes with the package structure *somepackage*. We suggest you use the IDE approach to accelerate your development.

3.4 Creating Web Services using JAX-WS

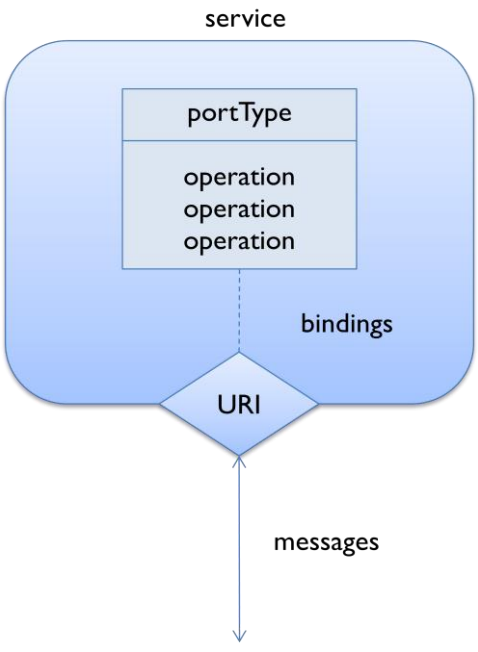
Before going to the discussion of creating and publishing web services, let's look at structure of

```
<xs:complexType name="currency">
    <xs:sequence>
        <xs:element name="conversionRateWithSGD" type="xs:double"></xs:element>
        <xs:element name="country" type="xs:string" minOccurs="0"></xs:element>
        <xs:element name="currencyCode" type="xs:string" minOccurs="0"></xs:element>
    </xs:sequence>
</xs:complexType>
```

a typical WSDL document.

Element	Function
types	Type defined using XML schema
message	Messages used by web service
portType / interface	Operations supported by web service
binding	Protocols used by web service

```
<definitions>
    <types> … </types>
    <message> …      </message>
    <message> …      </message>
        .
    <portType> … </portType>
    <binding> … </binding>
    <service> … </service>
</definitions>
```



service	Collection of endpoints (combination of binding and URI)

As you can see, a WSDL document is a representation of a web service that has a set of operations governed by the bindings (in our case, SOAP) and that is accessible at an endpoint represented by an URI. JAX-WS uses annotations to turn a normal Java class into a web service which is to be published with the help of an endpoint class.

Service endpoints

To turn a regular Java class into a web service, all we need is the annotation: **@webservice**. Yet there are a few other requirements. For a class to be a web service, it

- Must not be final or abstract.
- Must be defined as public.
- Must have a default public constructor.
- Must not define the finalize() method.
- Must be a stateless object and should not have client-specific states across method calls.

Once your service is compiled, it cannot publish itself at an endpoint. It needs another class called endpoint publisher class as shown below.

```
import javax.xml.ws.Endpoint;


public class SomePublisher {
    public static void main(String [] args) {
        Endpoint.publish("http://127.0.0.1:9999/someservice", new SomeService());
    }
}
```

The endpoint publisher class needs two parameters. The first is the publication URL at which your web service can be accessed and the second is an instance of your service class. Here, when you run the endpoint class, it publishes an instance of *SomeService* at your local machine. You can test your deployed service by checking its WSDL document at *http://127.0.0.1:9999/someservice?wsdl*.

Annotations

You can use more annotations to customize your web service. For example, web service operations follow Java naming conventions by default but you can change them as shown below.

```
@WebService (name="AnnotatedCCServiceDocument",
serviceName="RevisedCCServiceDocument")
public class CCServiceDocumentAnnotated {
    private HashMap<String, Currency> currencies
                    = new HashMap<String, Currency>();
    public CCServiceDocumentAnnotated(){
        currencies.put("SGD", new Currency("SGD", "Singapore", 1));
        currencies.put("USD", new Currency("USD", "United States", 1.28));
    }
```

```
    @WebMethod(operationName="conversion_rate")
    public double getConversionRate(@WebParam(name="from")
                    Currency from, @WebParam(name="to") Currency to){
            return from.getConversionRateWithSGD()/to.getConversionRateWithSGD();
    }
}
```

The annotation **@WebMethod** mainly used to rename the web service operations. It has another use as well. All the public methods of a service class are described in the WSDL document by default. You can use **@WebMethod(exclude = true)** annotation to prevent this from happening. Other useful annotations are **@WebResult** that is used to change the return result of an operation, **@WebParam** that is used for changing the names of parameters, **@OneWay** that is used for indicating that the operation has no return values and hence the invocation can be optimized.

Deployment in servers

The endpoint publisher approach described earlier is mainly useful for development and light production. In real-world scenarios, web services are deployed in Java EE servers such as Glassfish, WebLogic and WebSphere. There are two modes of deployment. The first is to package the web service in a web archive (.war) file and deploy it in a servlet container and let a servlet run as an endpoint. The second is to package the web service inside an enterprise archive file (.ear) and deployed it an EJB container (EJB endpoint). You can use IDEs such as NetBeans to directly deploy it in GlassFish server. Assuming your Glassfish server is bundled with NetBeans, the approach requires just a few clicks. You can also use tools like **ant** and **maven** for deployment but discussion of these tools together with EJB way is beyond the scope of this section. In general, developing and experimenting with web services using servers have the following benefits:

- Production-level experience – you may rather want to test your web services in servers that will give you some measures.
- Inspection of WSDL plus testing WebMethods – it is as simple as clicking on the options in your server's console.
- Interface with other enterprise components such as EJBs – EJBs are mainly for business logic and you can develop a servlet-endpoint web service that makes use of EJBs.
- Administrative support such as logging – many Java EE servers provides features such as dumping HTTP traffic which you can inspect and look for SOAP messages exchanged between your web service and consumers.
- Publishing web services as part of web applications

4 SOA Implementation in .NET 4 Windows Communication Foundation (WCF)

Microsoft introduced WCF as a part of .NET framework to help developers to create, consume services for .NET applications.

4.1 Why WCF?

Before discussing about WCF, it might be useful to discuss about pre-existing technologies and understand how are not able to meet the requirement of interoperability.

In the early time, remote communication amongst systems was possible through communication technologies such as Distributed Component Object Model (DCOM) or Common Object Request Broker Architecture (CORBA). DCOM is a set of concepts and program interfaces in which client program objects can request services from server program objects on other computer in a network. DCOM is an extension which is added to COM to solve COM's interoperability problem by introducing:

Marshalling: serialize and de-serialize the arguments and return value of methods calls.

Distributed garbage collection: Ensuring that references held by clients interfaces are released in case of termination (program closed or internet connection is lost).

With DCOM interface, functions of remote objects could be invoked remotely although the job is quite tedious because developers still need to handle manual work. For example, they must manually release reference of the remote object in the case of disconnection or application termination.

Although DCOM partially solves the interoperability problem, since DCOM is heavily dependent on the COM, only technologies that support COM could communicate with each other. This raised a big issue in intercommunication amongst systems which lie on different software technologies. Another drawback of DCOM is that it could not support communication over internet. DCOM relies on a proprietary binary protocol that not all object models support, which hinders interoperability across platforms. In addition, DCOM communicates over range of ports that are typically blocked by firewalls. In terms of design, development and deployment, DCOM is not suitable for a huge enterprise system.

.NET Remoting which was released in version 1.0 of .NET Framework eliminates the difficulties of DCOM by supporting different transport protocol formats and communication protocols. This allows .NET Remoting to be adaptable to the network environment. However, .NET only facilitates communication amongst systems lying on .NET framework and client still needs to make RPC for communication. This high coupling does not allow two systems running on different platform to interoperate. A remote object extends *System.MarshalByRefObject* and a client can make a call through a proxy object to invoke methods of this object remotely. This object is only known to systems running on the same environment (.NET environment). Hence, this technology was superseded by WCF, which is a part of .NET Framework 3.0

WCF is created with the mission to allow seamless communication between systems running on different platforms. WCF is much more flexible than its counterpart .NET Remoting because it allows different system running on different platforms to interoperate. WCF defines a set of industrial standards about service interactions, type conversions, marshalling and different method of service bindings for interoperability purpose. Not only allow communication through traditional SOAP based messages, WCF also defines a very platform specific binary protocol that allows communication with much better performance amongst .NET applications. Besides fully compliant with WS-* standards, WCF also offers features that are required for any enterprise systems such as security, reliability and transactions.

Some of the readers will wonder what happen to the systems that were built on pre-existing .NET technologies such as ASP.NET Web Services (ASMX), .NET Remoting, Enterprise Services, Web Service Enhancement or Microsoft Message Queuing. Are these systems able to communicate and interact with systems built on WCF? The information below gives readers a general idea of how well WCF supports backward compatibility and how much effort developers need to achieve compatibility.

ASMX application will be able to interact directly with WCF applications because the basic structure of the two technologies is similar. Both supports SOAP standard for sending and receiving messages. However, some advanced features of ASMX will not be able to port over WCF. We will not discuss this issue in this book chapter.

.NET Remoting: The mechanism of .NET Remoting and WCF is totally different. One is based on remote call (invoke function of remote objects) and the other based on SOAP. Therefore, it would not be surprising that these 2 technologies are not compatible.

Enterprise services: Developers need to make minimal efforts so that system using enterprise services can interact with systems using WCF. What they need to do is defining the interface that the services expose and the rest will be taken care by the framework. The effort put is as little as effort required making ASMX compatible with WCF applications.

Web service enhancements: Applications using WSE 1.0 and WSE 2.0 are not compatible with WCF. However, WSE 3.0 onwards can have direct communication with WCF. The effort required is also very minimal.

Microsoft Message Queue: Applications developed with WCF can interact with applications using message queue because WCF's queuing functions are based on MSMQ.

The above is not the only advantage and benefit WCF provide. Last but not least, WCF also explicitly supports for RESTful communication using POX, RSS and ATOM. However, discussion on these formats is beyond the scope of this book chapter. That's enough for the introduction. Now, let's start the journey of exploring WCF.

4.2 WCF concepts

For readers to have better understanding, we will go through some important concepts and terminologies used in WCF.

4.2.1 Endpoint

Service is exposed through endpoints. Endpoint is the combination of 3 aspects that lie between service providers and service consumers: address tells the client ***where*** the service is, contract is ***what*** the client can do with the service or in another words, what the service offers to the client and binding is ***how*** the client can consume the service. We will go into details of these 3 aspects.

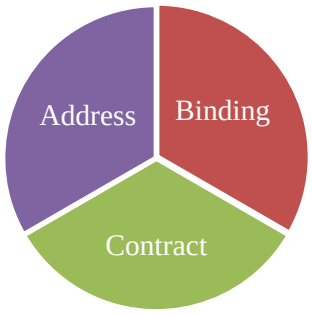


Figure 7: Three important components in WCF

4.2.2 Addresses

In WCF, every service is associated with a unique address that tells clients where the service is hosted. The address provides two important elements: the location of the service and the transport protocol (transport schema) used to communicate with the service. Addresses always have the following format:

[Transport]:// [machine or domain][:optional port]/path

Here are a few sample addresses.

https://localhost:8080/Secureservice

net.p2p://localhost:8080/service

Protocols supported in WCF:

TCP addresses use net.tcp for the transport, and typically include a port number. When a port number is not specified, the TCP address defaults to port 808. It is possible for two TCP addresses to share a port. TCP enables messages to be exchanged in binary format, hence enhances the performance.

HTTP addresses use HTTP for transport, and can also use HTTPS for secure transport. When a port number is unspecified, it defaults to 80. Similar to TCP addresses, two HTTP addresses from the same host can share a port, even on the same machine.

IPC (Inter-Process Communication) addresses use *net.pipe* for transport, to indicate the use of the Windows named piped mechanism. In WCF, services that use named pipes can only accept calls from the same machine. Consequently, you must specify either the explicit local machine name or local host for the machine name, followed by a unique string for the pipe name. Only one named pipe could be open on a machine, and therefore it is not possible for two named pipe addresses to share a pipe name on the same machine. Name piped protocol is optimized for inter process communication; therefore, if client and service provider are running on the same machine, IPC protocol would be preferred over others.

MSMQ (Microsoft Message Queue) addresses use *net.msmq* for transport, to indicate the use of Microsoft Message Queue. MSMQ is a protocol that often used in distributed communication environment. Clients do not need to be online all the time but still can queue their service request for consumption later when they go online.

Peer network address use net.p2p for transport, to indicate the use of the Windows peer network transport. Using peer networks is beyond the scope of this book chapter.

Although protocols are in wide range of choice, suitable protocol must be carefully chosen to fit service consumer's need.

4.2.3 Contracts

In WCF, all services expose contracts. The contract is a platform-neutral and standard way of describing what the service does. WCF defines four types of contracts

Data contracts define which data types are passed to and from the service. WCF defines implicitly contracts for built-in types such as *int* and *string*, but you can easily define explicitly opt-in data contracts for custom types. For example

```
using System.Runtime.Serialization;
namespace ServiceDemo
{
	[DataContract]
	public class Name
	{
		[DataMember]
		public string Id;
		[DataMember]
		public string FirstName;
		[DataMember]
		public string LastName;
	}
}
```

This code fragment shows a simple C# class with special annotations *[DataContract]* and *[DataMember]* above each attribute of the class. These annotations actually tell WCF that these attributes are exchangeable though web services. Note that WCF allows flexibility in annotating class members. Members without *[DataMember]* annotations will not be exchangeable through the service.

Service contracts describe which operations the client can perform on the service. For example

```
using System.ServiceModel;
namespace ServiceDemo
{
	[ServiceContract]
	public interface INameService
	{
		[OperationContract]
		void AddName(Name name);
		[OperationContract]
		List<Name> GetNames();
		[OperationContract]
		void RemoveName(string id);
	}
}
```

The above code fragment is nothing else but a very simple interface with some declared operations. But please notice the *[ServiceContract]* and *[OperationContract]* annotations. The operations with *[OperationContract]* annotation are "invokable" remotely by service clients. Similarly to data contract, method without [*OperationContract*] will not be invoke-able by service consumers.

Fault contracts define which errors are raised by the service, and how the service handles and propagates error to its clients.

Message contracts allow the service to interact directly with message. Message contracts can be typed or un-typed, and are useful in interoperability cases and when there is an existing message format you have to comply with. As a WCF developer, you should use message contracts only rarely.

4.2.4 Binding

Binding simply tells clients how to consume service. In previous section, we discussed about the contracts. Readers may ask how these data contracts can be exchanged between clients and the service provider. That's how the binding fits in the scenario. Binding is actually a collection of binding elements that define how message could be exchange over the service. Some but not limited to binding elements are message encoding, transport protocol, or security options. Knowing what binding method service provider used, clients would be able to communicate with it.

WCF provides a wide range of binding methods. Each binding method is used with different purpose. The binding methods starting with WS-* are web service compliant. It means that they are used for open system interoperability. The binding started with Net, as you can guess, are binding methods that allow WCF applications able to talk to existing applications running on pre-existing .NET technologies.

Binding name	Transport	Message Encoding	Security Mode	Reliability
BasicHttpBinding	HTTP	Text	None	Not Supported
WSHttpBinding	HTTP	Text	Message	Disabled
WSDualHttpBinding	HTTP	Text	Message	Enabled
WSFederationHttpBinding	HTTP	Text	Message	Disabled
NetTcpBinding	TCP	Binary	Transport	Disabled
NetPeerTcpBinding	P2P	Binary	Transport	Not Suppported
NetNamedPipesBinding	Named pipes	Binary	Transport	Not Supported
NetMsmqBinding	MSMQ	Binary	Message	Not Supported
MsmqIntegrationBinding	MSMQ	Use pre-WCF serialization format	Transport	Not Supported
CustomBinding	Customized	Customized	Customized	Customized

4.3 What else WCF supports?

4.3.1 Dynamic discovery

In the case of disaster, a machine that hosts a particular service goes down that may cause the service interruption. A business would be drastically affected if the supporting system could not

ensure 100% uptime. Therefore, a concept of dynamic discovery is introduced in WCF 4.0. This is a set of APIs that allows clients to dynamically discover and bind to a particular service. Figure 8: Dynamic discovery feature in WCF visualizes the concept of dynamic discovery.

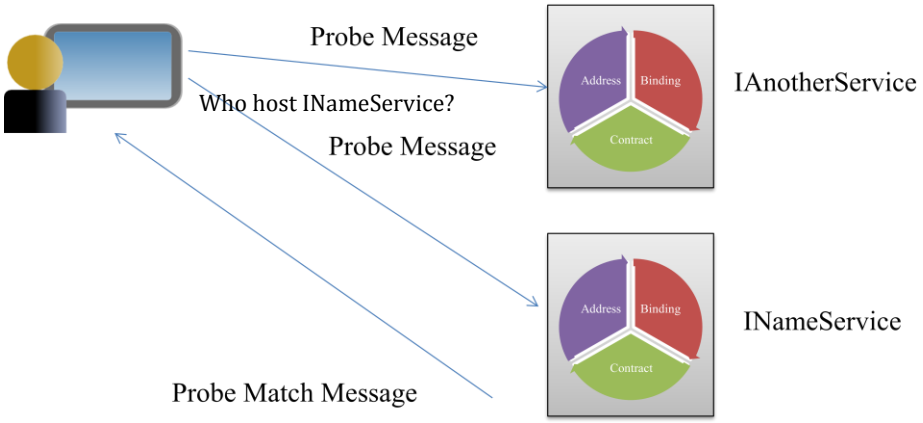


Figure 8: Dynamic discovery feature in WCF

The client sends a broadcast probe message to ask who is hosting the INameService. If a particular endpoint exposes INameService, it will send client a probe match message. The client will choose from the list of replies an endpoint and binds to it. With this feature, client does not necessarily bind to a fixed address defined by the developer. The code below actually illustrates the idea of dynamic discovery.

```
DiscoveryClient discoveryClient = new DiscoveryClient("udpDiscoveryEndpoint");
FindCriteria findCriteria = new FindCriteria(typeof(INameService));
FindResponse findResponse = discoveryClient.Find(findCriteria);

if (findResponse.Endpoints.Count > 0)
{
    EndpointAddress address = findResponse.Endpoints[0].Address;
    ChannelFactory<INameServiceChannel> factory
                    = new ChannelFactory<INameServiceChannel>(
        new BasicHttpBinding(), address);
    INameServiceChannel client = new factory.CreateChannel();

    //<Client code here>

    client.Close();
    factory.Close();
}
```

Notice that BasicHttpBinding is used. The above code will be meaningless if service clients do not know what binding method a particular endpoint uses. Therefore, in the discovery API also allows clients to automatically resolve the binding method used by the endpoint. The sample code below shows how binding method can be resolved:

```
DiscoveryClient discoveryClient = new DiscoveryClient(new
UdpDiscoveryEndpoint());
FindCriteria findCriteria =
FindCriteria.CreateMetadataExchangeEndpointCriteria();
FindResponse findResponse = discoveryClient.Find(findCriteria);


if (findResponse.Endpoints.Count > 0)
{
       var endpoints = MetadataResolver.Resolve (typeof(INameService),
                                          findResponse.Endpoints[0].Address);


       foreach(ServiceEndpoint endpoint in endpoints)
       {
            Console.Writeline ("Endpoint address : " + endpoint.Address +
                           "\tBinding method: " + endpoint.Binding.Name);
       }
}
```

4.3.2 Multiple endpoints

With the mission to interoperate as many system running on different platforms as possible, WCF allows a service could be exposed through different endpoints. Each endpoint is configured to serve a particular type of client.

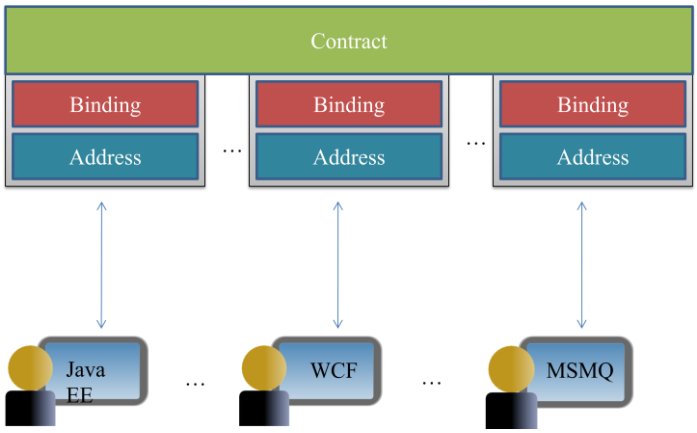


Figure 9: Multiple endpoints features in WCF

Configuring a service to be exposed through different endpoints is extremely easy. Firstly, you as a service provider need to understand which platform your system needs to support. Are clients running on Java platform or using message queue or ASP.NET web services? Knowing this is critically important because different technologies will use different binding methods. The service configuration with GUI is a very handy tool to achieve this task.

4.3.3 Deployment options

WCF service class cannot exist in a void. Every WCF service must be hosted in Windows process called host process. A single host process can host multiple services, and the same service type can be hosted in multiple service type can be hosted in multiple host processes. Hosting a WCF

is never that simple. It can be hosted on IIS, hosted as a windows service or even self hosting (hosting in an application).

Hosting on IIS

The main advantage of hosting a service in IIS web server is that the host process is launched automatically upon the first client request, and you rely on IIS to manage the life cycle of the host process. Of course hosting services on IIS will reduce the workload for developers because developers can rely on IIS to manage process, idle shutdown, and health monitoring. The main disadvantage of IIS hosting is that you can use only use HTTP. With IIS5, you are further restricted to having all services use the same port number.

Hosting as a Windows service

The advantage of this hosting option is that the developers do not need to care about life cycle and states of the service. Everything will be handled by the Service Control Manager (SCM). SCM provides a GUI for service administrator to start, stop and even set dependency for the service. This hosting is supported by all windows version and does not require any extra software installation.

Hosting using Windows Activation Service (WAS)

WAS is part of IIS7, but can be installed and configured separately. To use the WAS for hosting your WCF service, you need to supply a .svc file, just as with IIS. The main difference between IIS and WAS is that WAS is not limited to HTTP and can be used with any of the available WCF transport, ports, and queues. WAS offers many advantages over self-hosting, including application pooling, recycling, idle time management, identity management.

Self hosting

Self-hosting is the name for the technique used when the developer is responsible for providing and managing the life cycle of the host process. Service is hosted inside a managed application. Since no endpoint is explicitly assigned, the WCF runtime will create one endpoint per base address for each service contract implemented by the service.

5 DATA SERVICES

In the earlier sections, we have been focusing on web services. Consuming web services can be thought invoking methods or functions over the web. These methods usually encapsulate logic and therefore provide abstraction from its implementation. In this section, we will be looking at data as a service. In a nutshell, data service provides data as 'it is', without any sophisticated logic that will be performed on the data. Data service provides abstraction at the data access layer. It is to be noted that one can also build a web service that consumes data service. The data service will provide relevant data, and the web service will contain logic that acts on the data.

5.1 Benefits of Data as a Service

Compared to web service, a data service consumer has full control and therefore more flexibility on how to operate on data in data-intensive applications. The developer no longer needs to be restricted by logic provided by available web services.

Other benefits are similar to web services such as independence from data storage implementation and interoperability.

5.2 Open Data Protocol

Just as web services are based on a set of standards (for interoperability), there are standards for data services as well. In the rest of the section, we will be focusing on Open Data Protocol (OData). OData is a set of standards that allow data consumers and get data in a standardized way (interoperability on consumer end). Publisher of data services will have to conform to OData to expose their data as a service regardless of how their data store or databases are implemented (interoperability on provider end). OData is based on an open standard and more information regarding the protocol is available at http://www.odata.org.

OData is also relevant to enterprise systems. Examples of enterprise systems that are beginning to use or have adopted OData include Microsoft Sharepoint, IBM WebSphere and SAP.

5.2.1 OData Data Format

There is some similarity between OData and Web Service. Namely, both of them use HTTP protocol and exchange data in XML format (it is to be noted that OData provides JSON communication as well). Figure 10 shows the XML returned by eBay data services at http://ebayodata.cloudapp.net/.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<service xml:base="http://ebayodata.cloudapp.net/" ...>
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="Bidders">
      <atom:title>Bidders</atom:title>
    </collection>
    <collection href="Categories">
      <atom:title>Categories</atom:title>
    </collection>
    <collection href="Items">
      <atom:title>Items</atom:title>
    </collection>
    <collection href="Users">
      <atom:title>Users</atom:title>
    </collection>
    ...
  </workspace>
</service>
```

Figure 10: OData feed (partial)

One can see the *collections* listed in the XML in Figure X. A collection is analogous to tables in a database. In this case we have collections with names Bidders, Categories, Items etc. To browse the collections, we simply follow the *href* links. For example, we can browse to http://ebayodata.cloudapp.net/Users to browse to the *Users* collection.

If one browse to previous link to Users collection, a HTTP 403 Forbidden status will be returned. This brings us to another aspect of exposing data service, namely access control and rights. Depending on user credentials, selected subset of data can be exposed to different users.

5.2.2 Data querying

Another powerful feature of OData is data querying. This makes sense because we would want to be selective of the data that is returned. OData query supports predicates, projections as well as aggregation in querying data which is similar to SQL. For example the following URL:

http://ebayodata.cloudapp.net/Items?$filter=CurrentPrice lt 200&country=SG&search='monitor'

returns a XML feed of listings matching the search term 'monitor' with price lesser than $200.

Below are some of the more common Uri operations supported by OData. An extensive documentation of the URI query operations supported is available at http://www.odata.org/developers/protocols/uri-conventions.

$orderby	For sorting
$top	For retrieving first N data
$skip	For excluding first N entries in data
$filter	For selecting entries satisfying certain predicate expressions
$expand	Eagerly load details of sub-entries in data
$format	For specifying return format (Atom, XML, JSON)
$select	For projecting specified fields
$inlinecount	For counting entries

5.3 WCF data service

WCF Data Service is Microsoft's implementation of OData. It is previously called ADO.NET Data Services. It allows developer to expose data from sources such as databases (or objects) as OData.

5.3.1 WCF metadata

The WSDL equivalent of web service in OData is called metadata. For instance, one can browse to http://ebayodata.cloudapp.net/$metadata to view the metadata of the eBay data service. It is to be noted that references between collections (similar to references in relational database) can be captured in the metadata as well.

5.3.2 WCF architecture

The architecture of WCF Data Service is listed in Figure 11. It can be seen the flexibility in the kind of data source that can be exposed as a data service. This includes .NET Common Language Runtime (CLR) objects, Microsoft SQL Server or even 3rd party data source. Example of 3rd party provider includes MySQL, where an official support for Entity Framework was introduced in MySQL Connector for .NET from 6.0 onwards.

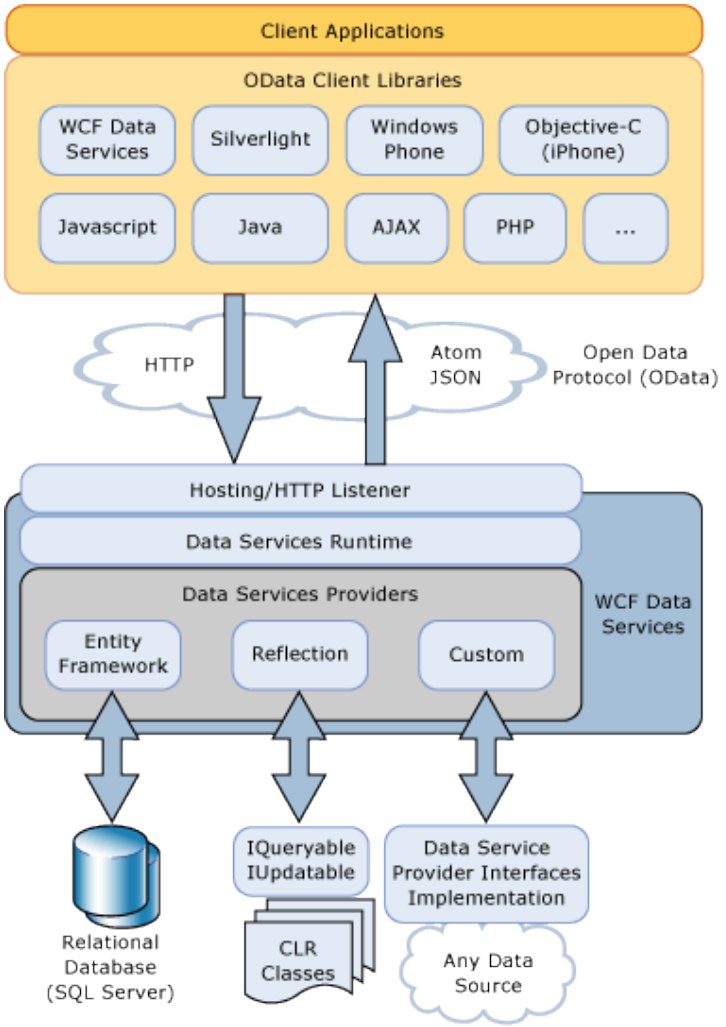


Figure 11: WCF Architecture

For a tutorial on how to expose data as a data service, you can visit http://sites.google.com/site/cs4217jan2011team1/tutorials.

6 CONCLUSION

As enterprise systems tend to become more and more complex, system designers must ensure the scalability of the systems and how they communicate with each other. This task requires more and more effort both in terms of business functionalities and regarding the technologies chosen. Using SOA approach will reduce the difficulty to a certain level. Generally in SOA, systems expose functionalities for other systems to consume without knowing the complex logic behind. As a matter of fact, those consumers, in turn, could be service providers as well. Hence, service oriented architecture is an excellent option to scale the systems. Since more and more systems such as SAP are moving towards SOA, we believe that SOA is creating a new trend today. Being conceived and developed a long time ago, however, only until now, SOA proves its important role as the backbone of enterprise systems and an essential building block in cloud computing, which is now deemed as a promising technology in the future.

7 BIBLIOGRAPHY

Amit Bahree, Dennis Mulder, Shawn Cicoria, Chris Peiris, Nishith Pathak. *Pro WCF: Practical Microsoft SOA Implementation [Paperback].* Apress, 2007.

Chappell, David. *Introducing Windows Communication Foundation in .NET Framework 4.* March, 2010. http://msdn.microsoft.com/library/ee958158.aspx (accessed 9 March, 2011).

Corporation, Oracle. *The Java EE 5 Tutorial.* n.d. http://download.oracle.com/javaee/5/tutorial/doc/bnazq.html (accessed 10 March, 2011).

Eichengreen, Barry. *One Economy, Ready or Not: Thomas Friedman's Jaunt Through Globalization.* May/June, 1999. http://www.foreignaffairs.com/articles/55017/barry-eichengreen/one-economy-ready-or-not-thomas-friedman-s-jaunt-through-globaliz (accessed 6 March, 2011).

Erl, Thomas. *SOA Principles of Service Design.* Prentice Hall, 2007.

Goncalves, Antonio. *Beginning Java(TM) EE 6 with GlassFish(TM) 3: From Novice to Professional.* Apress, Inc., 2009.

Hewitt, Eben. *Java SOA Cookbook.* O'Reilly Media, 2009.

Jane Laudon, Kenneth Laudon. *Essentials of Management Information Systems.* Prentice Hall, 2007.

Judith Hurwitz, Robin Bloor, Carol Baroudi, Marcia Kaufman. *Service Oriented Architecture for Dummies.* Wiley Publishing, Inc., 2007.

Kalin, Martin. *Java Web Services: Up and Running.* O'Reilly Media, Inc., 2009.

Klein, Scott. *Professional WCF Programming: .NET Development with the Windows Communication Foundation.* John Wiley & Sons, 2007.