# Windows Communication Foundation Overview

One of the biggest IT topics today has to be the concept of Service-Oriented Architecture (SOA). Service-Oriented Architecture isn't new. You'd think that with the coverage it has received over the past few years that developers and "techy" individuals would understand it better, yet it ranks fairly high on the misunderstood-o-meter because its interpretation, implementation, and use is pretty loose due to the fairly vague definition.

When you want to understand the meaning of something, you usually go to a place that defines it, such as a dictionary. In this case, we turn to the W3C to understand the definition of SOA. The W3C defines Service-Oriented Architecture as "A set of components which can be invoked and whose interface descriptions can be discovered and published" (`http://www.w3.org/TR/ws-gloss/`). As you sit and ponder this definition, it becomes quite apparent that this definition is fairly broad. It also becomes apparent why the Service-Oriented Architecture picture is somewhat fuzzy, because the definition leaves a lot of room for interpretation.

With this in mind, the purpose of this chapter is twofold. First, to better explain what SOA is and the need for it; and second, to introduce Windows Communication Foundation (WCF) and explain how it answers some of the SOA needs. This chapter covers the following:

❑ The need for SOA

❑ How Windows Communication Foundation addresses the SOA needs

## The Need for SOA

To understand Service-Oriented Architecture, take a look at the scenario shown in Figure 1-1.

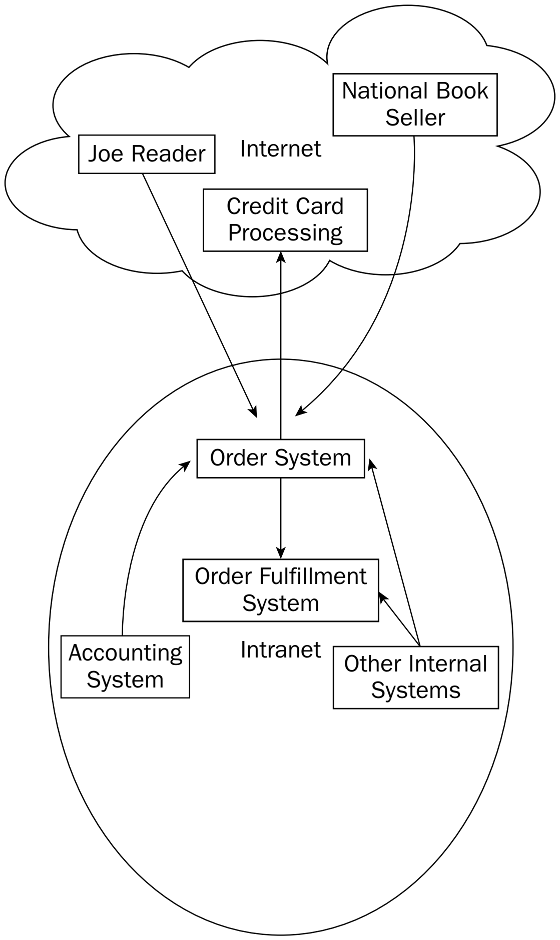

**Figure 1-1**

The preceding illustration shows the process of a very simplified book publisher order solution. In this example, a book publisher can receive book orders from both a single, individual reader as well as large-quantity book orders from large national book resellers. Orders are received by the Order System application, which collects and processes the orders.

Internally, the Order System application collects and processes the orders, such as validating credit cards and forwarding the order to the Order Fulfillment system. Both the Order System application and the Order Fulfillment application communicate with other internal applications and systems for various reasons.

Over time this publishing company becomes popular because it is hiring great authors and putting out high-quality books, and it becomes apparent that this simple solution is not keeping up with the demand and volume of orders. For example, maybe the current Order System can't keep up with the volume of orders coming in, and thus the Order Fulfillment system is having difficulty handling the amount of orders being handed to it from the Order System.

Service-Oriented Architecture says that the current system can, and should, be flexible enough to allow changes to the existing architecture without disrupting the current architecture and infrastructure currently in place. That is, each piece should be isolated enough that it can be replaced without disturbing the flow and process of the rest of the system. It is the concept of designing the technology processes within a business.

If the developers of the original systems in place were smart, they would have designed the architecture to allow for such a "plug-and-play" type of environment, but at that time their decision was somewhat difficult because of the technologies they had to choose from.

## A Look Back

Through the years, developers have had a plethora of technology choices to choose from when it has come to building distributed applications. Lately it has been ASP.NET and WSE (Web Service Enhancements), used to build web services. These allow for the communication of information across different platforms regardless of the client. In addition to these two technologies, the Framework provides .NET Remoting, which provides object communication no matter where the application components reside. Yet even with Remoting you have pros and cons. Many times objects are created remotely, whereas WCF deals with message transmissions natively. .NET Remoting works well in some distributed scenarios yet lacks the ability to be a primary application interface. For example, web services can call Remoting components, but a remote object created outside of your network won't work with Remoting. Prior to that you had COM+ and Microsoft Message Queuing (MSMQ). MSMQ provided a fast and reliable method of application communication through the process of sending and receiving messages.

I remember not many years ago using MSMQ in a fairly large project because the guaranteed delivery of information was critical, yet today I use ASP.NET web services almost religiously because of the many benefits they offer (simplicity, diversity of environments, and so on).

These were, and still are, great technologies that provided great solutions to countless development problems. Yet every time you as a developer had a distributed system problem to solve, you had to ask yourself "which one of these technologies more easily and efficiently solves my problem?" The downside to each one of these is that your method of implementation will vary greatly based on your choice of solution, especially if your solution uses more than one of these technologies.

Given this information, how would you as a developer tackle the responsibility of designing and architecting the book publisher systems given the definition of SOA as described earlier? ASP.NET web services, WSE, and .NET Remoting have been out roughly five years; COM+ and MSMQ have been out a lot longer.

## Understanding Service Orientation

Given the book publisher example, this section now looks at it from a Service-Oriented Architecture perspective. Again, SOA says that the current system can, and should, be flexible enough to allow changes to the existing architecture without disrupting the current architecture and infrastructure currently in place.

Building this with service orientation in mind, several changes are made to the architecture, which fulfills the needs of the system without disrupting the process, as shown in Figure 1-2.

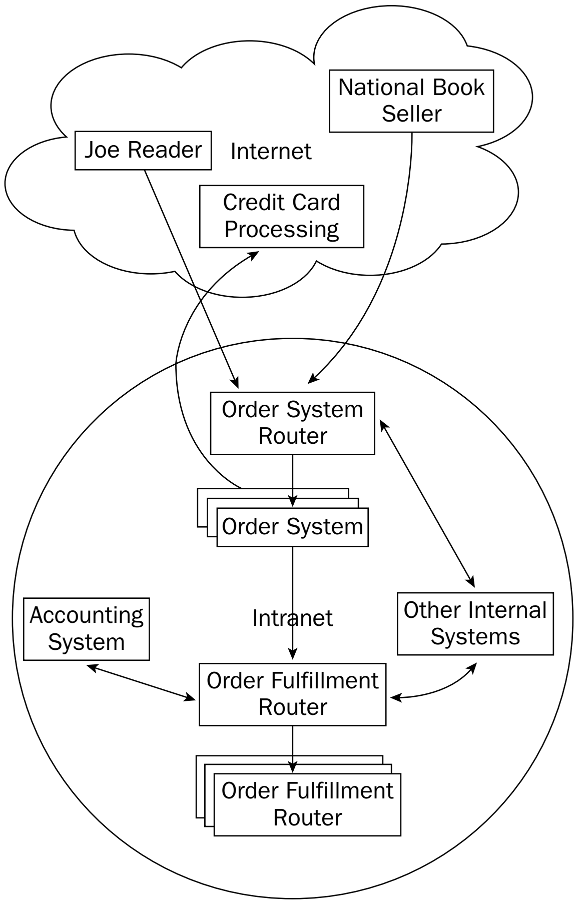

**Figure 1-2**

To handle the amount of orders coming in, a router was put in front of the Order Process service that then distributes the orders to one of many Order Process services. An Order Fulfillment router was also placed in front of the Order Fulfillment service, which accomplishes the same thing as the Order Process router; that is, it takes the incoming orders from the Order Process service and distributes the orders to one of many Order Fulfillment services.

The other internal systems can still communicate and exchange information with these services without any changes. Externally, Joe Reader and the National Book Seller have no idea that changes were made at the publisher's end — it is all transparent to them. In fact, they might even see a better responding system when placing orders. The key here is that with SOA, major changes can take place behind the scenes completely transparent to the end user and without any interruption of the system.

Software as a service is not a new concept, and in fact if you have been using web services you can start to see where this is going. One could argue that web services was the first step on the road to SOA. Is that statement true? Yes and no. No, in that you can have an SOA solution without using ASP.NET web services. Yes, in that it is a great beginning to something much larger.

It is possible to have a web service that follows no SOA principles and a web service that exhibits wonderful SOA traits. For example, a good web service is almost self-describing, providing useful information. I want to hit a web service that gives me stock quotes, or lets me buy or sell stocks, or tells me how my portfolio is doing. In the case of the book publisher, I want a web service that tells me information about my order. These are web services that exhibit SOA traits.

In contrast, a web service that provides reads as well as writes data to my database shows no SOA. Now, don't get me wrong. Those types of web services have their place and provide great benefits (I have written a few myself). But they don't fit in the SOA realm and don't conform to the SOA principles.

So what are these principles?

## *Service-Oriented Architecture Principles*

Streams of information have been flowing from Microsoft in the forms of articles and white papers regarding its commitment to SOA, and in all of this information one of the big areas constantly stressed are the principles behind service orientation:

- Explicit boundaries
- Autonomous services
- Policy-based compatibility
- Shared schemas and contracts

### *Explicit Boundaries*

As you will learn in the next section, SOA is all about messaging — sending messages from point A to point B. These messages must be able to cross explicit and formal boundaries regardless of what is behind those boundaries. This allows developers to keep the flexibility of how services are implemented and deployed. Explicit boundaries mean that a service can be deployed anywhere and be easily and freely accessed by other services, regardless of the environment or development language of the other service.

The thing to keep in mind is that there is a cost associated with crossing boundaries. These costs come in a number of forms, such as communication, performance, and processing overhead costs. Services should be called quickly and efficiently.

### Autonomous Services

Services are built and deployed independently of other services. Systems, especially distributed systems, must evolve over time and should be built to handle change easily. This SOA principle states that each service must be managed and versioned differently so as to not affect other services in the process.

In the book publisher example, the Order Process service and Order Fulfillment service are completely independent of each other; each is versioned and managed completely independent of the other. In this way, when one changes it should not affect the other. It has been said that services should be built *not* to fail. In following this concept, if a service is unavailable for whatever reason or should a service depend on another service that is not available, every precaution should be taken to allow for such services to survive, such as redundancy or failover.

### Policy-Based Compatibility

When services call each other, it isn't like two friends meeting in the street, exchanging pleasantries, and then talking. Services need to know a little more about each other. Each service may or may not have certain requirements before it will start communicating and handing out information. Each service has its own compatibility level and knows how it will interact with other services. These two friends in fact aren't friends at all. They are complete and total strangers. When these two strangers meet in the street, an interrogation takes place, with each person providing the other with a policy. This policy is an information sheet containing explicit information about the person. Each stranger scours the policy of the other looking for similar interests. If the two services were to talk again, it would be as if they had never met before in their life. The whole interrogation process would start over.

This is how services interact. Services look at each others' policy, looking for similarities so that they can start communicating. If two services can't satisfy each others' policy requirements, all bets are off. These policies exist in the form of machine-readable expressions.

Policies also allow you to move a service from one environment to another without changing the behavior of the service.

### Shared Schemas and Contracts

Think "schemas = data" and "contracts = behavior." The contract contains information regarding the structure of the message. Services do not pass classes and types; they pass schemas and contracts. This allows for a loosely coupled system where the service does not care what type of environment the other service is executing on. The information being passed is 100 percent platform independent. As described previously, a service describes it capabilities.

## Microsoft's Commitment to SOA

In 2005 Microsoft showed it was serious about SOA by releasing SQL Server 2005 and Visual Studio 2005. VS 2005 introduced several new components that aid in the architecting of service-oriented applications. The first of these tools is the Application Designer. This tool is to help application developers and designers visually build applications that use or make available services.

The second tool is the Logical Datacenter Designer, which provides the ability to create a logical representation of your datacenter. When I first read about this I thought "why would I want this?," but think about it. Nine times out of 10 the environments in which you develop are not the same environments in which your production application will go. Thus, the Logical Datacenter Designer is a tool that allows developers to see important information about the environment in which their application is targeted, providing such information as current software configurations and versions (SQL Server, IIS, and so on).

Not to be left out, SQL Server 2005 hit the street with a ton of new features and enhancements, many of which are SOA-focused such as a new XML data type and the ability to return XML in a Dataset, support for XQuery, and endpoints exposed as web services.

Where is Microsoft going next? Good question. In the next 12 to 18 months, Microsoft will be introducing two new products. This book is the focus of one of them, Windows Communication Foundation. The other is Windows Vista and WinFX. Windows Vista introduces a new XML markup language called XAML (pronounced "zamel"), which uses a declarative syntax for describing user-interface elements. It also includes a brand-new programming interface, further building on the SOA model by unifying all of the messaging models as well as exposing all of the Windows Communication Foundation capabilities.

### SOA Wrap-up

Entire books have been written solely for the purpose of describing in great detail SOA and explaining the reasoning behind it. I have tried to sum it up in seven or eight pages. Consider it the *Readers Digest* version, but hopefully in these few pages you have come to understand SOA a little bit and see some of the benefits of it. But for your enjoyment, I will briefly list them here:

- ❑ Services are platform and location independent. A service does not care where the service is located, and it does not care about the environment of another service to be able to communicate with it.
- ❑ Services are isolated. A change in one service does not necessitate a change in other services.
- ❑ Services are protocol, format, and transport neutral. Service communication information is flexible.
- ❑ Services are scalable. Remember the book publisher example?
- ❑ Service behavior is not constrained. If you want to move the location of the service, you only need to change the policy, not the service itself.

With that, you can move on to the real reason for this chapter, and realistically, this book. Earlier in this chapter, several other technologies were mentioned when SOA was being introduced and when distributed architecture was being discussed. This begs the question: Wouldn't it be great if all of these were combined into a single SOA solution?

## Why Windows Communication Foundation

Windows Communication Foundation (WCF) is a platform, a framework if you will, for creating and distributing connected applications. It is a fusion of current distributed system technologies designed and developed from day one with the goal of achieving SOA nirvana, or coming as close to it as possible.

WCF is a programming model that enables developers to build service solutions that are reliable and secure, and even transacted. It simplifies development of connected applications and offers something to developers that they have not seen in quite a while — a unified, simplified, and manageable distributed system development approach.

Built on top of the 2.0 .NET Framework CLR (Common Language Runtime), the Windows Communication Foundation is a set of classes that allow developers to build service-oriented applications in their favorite .NET environment (VB.NET or C#).

This section begins by taking a detailed look at the architecture of WCF and the components that make WCF what it is.

## WCF Architecture

At the heart of WCF is a layered architecture that supports a lot of the distributed application development styles. Figure 1-3 illustrates the layered architecture of Windows Communication Foundation.

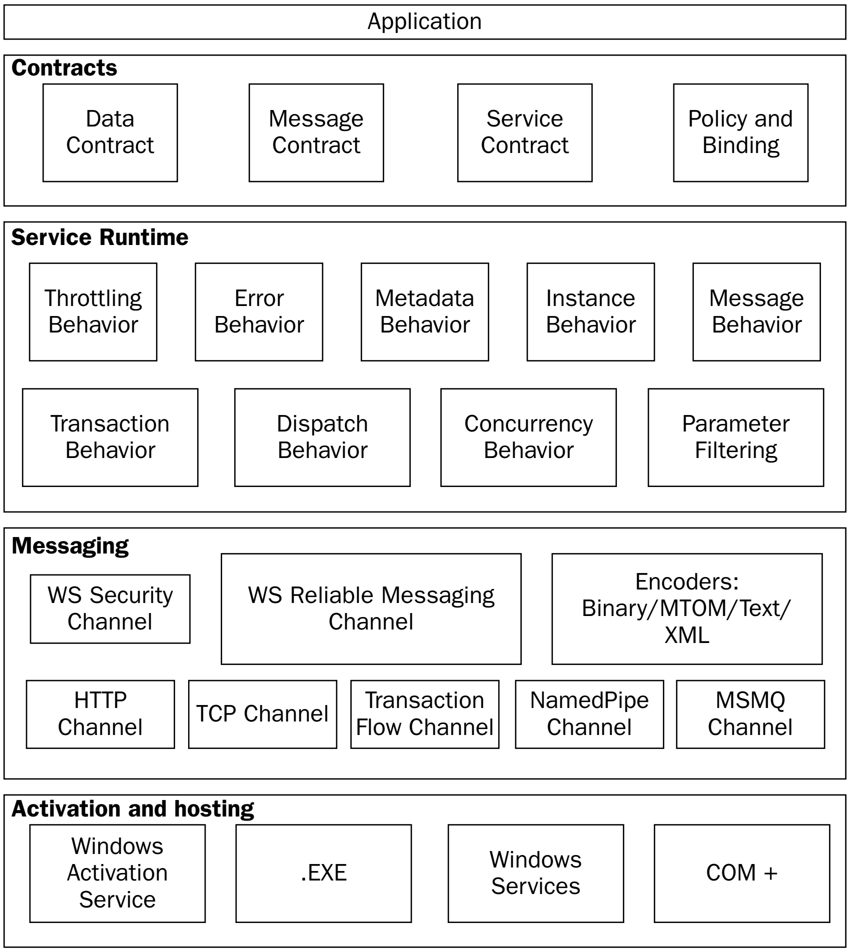

Figure 1-3

This layered architecture, which provides developers a new service-oriented programming model, is discussed in detail in the following sections.

## *Contracts*

WCF contracts are much like a contract that you and I would sign in real life. A contract I may sign could contain information such as the type of work I will perform and what information I might make available to the other party. A WCF contract contains very similar information. It contains information that stipulates what a service does and the type of information it will make available.

Given this information, there are three types of contracts: data, message, and service. More detailed information about contracts is given in Chapter 6, so consider this a primer.

### Data

A data contract explicitly stipulates the data that will be exchanged by the service. The service and the client do not need to agree on the types, but they do need to agree on the data contract. This includes parameters and return types.

### Message

A message contract provides additional control over that of a data contract, in that it controls the SOAP messages sent and received by the service. In other words, a message contract lets you customize the type formatting of parameters in SOAP messages.

Most of the time a data contract is good enough, but there might be occasions when a little extra control is necessary.

### Service

A service contract is what informs the clients and the rest of the outside world what the endpoint has to offer and communicate. Think of it as a single declaration that basically states "here are the data types of my messages, here is where I am located, and here are the protocols that I communicate with."

There is a bit more to it than this, and Chapter 6 covers this in greater detail. But for now, suffice it to say that a service contract is one or more related message interactions.

### Policy and Binding

Remember the SOA principles discussed earlier? Remember the one that discusses policies? Here is where they come into play. Policy and binding contracts specify important information such as security, protocol, and other information, and these policies are interrogated looking for the things that need to be satisfied before the two services start communicating.

## *Service Runtime*

The Service Runtime layer is the layer that specifies and manages the behaviors of the service that occur during service operation, or service runtime (thus "service runtime behaviors"). Service behaviors control service type behaviors. They have no control over endpoint or message behaviors. Likewise, endpoint and message behaviors have no control over service behaviors.

The following lists the various behaviors managed by the Service Runtime layer:

- **Throttling Behavior:** The Throttling behavior determines the number of processed messages.
- **Error Behavior:** The Error behavior specifies what action will be taken if an error occurs during service runtime.
- **Metadata Behavior:** The Metadata behavior controls whether or not metadata is exposed to the outside world.
- **Instance Behavior:** The Instance behavior drives how many instances of the service will be available to process messages.
- **Message Inspection:** Message Inspection gives the service the ability to inspect all or parts of a message.
- **Transaction Behavior:** The Transaction behavior enables transacted operations. That is, if a process fails during the service runtime it has the ability to rollback the transaction.
- **Dispatch Behavior:** When a message is processed by the WCF infrastructure, the Dispatch Behavior service determines how the message is to be handled and processed.
- **Concurrency Behavior:** The Concurrency behavior determines how each service, or instance of the service, handles threading. This behavior helps control how many threads can access a given instance of a service.
- **Parameter Filtering:** When a message is acted upon by the service, certain actions can be taken based on what is in the message headers. Parameter Filtering filters the message headers and executes preset actions based on the filter of the message headers.

### Messaging

The Messaging layer defines what formats and data exchange patterns can be used during service communication. Client applications can be developed to access this layer and control messaging details and work directly with messages and channels.

The following lists the channels and components that the Messaging layer is composed of:

- **WS Security Channel:** The WS Security channel implements the WS-Security specification, which enables message security.
- **WS Reliable Messaging Channel:** Guaranteed message delivery is provided by the WS Reliable Messaging channel.
- **Encoders:** Encoders let you pick from a number of encodings for the message.
- **HTTP Channel:** The HTTP channel tells the service that message delivery will take place via the HTTP protocol.
- **TCP Channel:** The TCP channel tells the service that message delivery will take place via the TCP protocol.
- **Transaction Flow Channel:** The Transaction Flow channel governs transacted message patterns.
- **NamedPipe Channel:** The NamedPipe channel enables inter-process communication.
- **MSMQ Channel:** If your service needs to interoperate with MSMQ, this is the channel that enables that.

### Activation and Hosting

The Activation and Hosting layer provides different options in which a service can be started as well as hosted. Services can be hosted within the context of another application, or they can be self-hosted. This layer provides those options.

The following list details the hosting and activation options provided by this layer:

- **Windows Activation Service:** The Windows Activation Service enables WCF applications to be automatically started when running on a computer that is running the Windows Activation Service.
- **.EXE:** WCF allows services to be run as executables (.EXE files).
- **Windows Services:** WCF allows services to be run as a Windows service.
- **COM+:** WCF allows services to be run as a COM+ application.

## The Makeup of WCF

Now that you have an idea of the architecture behind Windows Communication Foundation, a few pages need to be spent covering those things that make WCF what it is. That is, why all the hype surrounding WCF and what makes it so unique?

If you have been paying attention during this chapter, you can more than likely pick out a small handful of things that really make WCF stand out. On the surface, you might make the incorrect assumption that WCF is just a bunch of "add-ons" to the already existing framework. Not so. As you dig into WCF you will really start to see that a lot of thought and time went into developing what you see in front of you. To help you out, this section lists a number of the great focus points that WCF has to offer. Think of it as the personality of WCF:

- Programming model
- Scalability
- Interoperability
- Enhanced communication
- Enterprise enabled

### Programming Model

The great thing about WCF is that there is no "right way" to get from point A to point B. If fact, WCF lets users start at point A and go to point B any way they see fit. This is because the programming model in WCF lets developers control how and when they want to code things and yet gives them the ability to do that with a minimum amount of code.

As you have seen from the architecture, there are only a small handful of major components that a developer will need to work with to build high-class services. However, WCF also lets developers drill down to lower-level components if they desire to get more granular with their options. WCF makes this very simple. The WCF programming model lets a developer take whichever approach he or she desires. There is no single "right" way.

The programming model also combines many of the earlier technologies, such as the ones mentioned earlier in the chapter (MSMQ, COM+, WSE, and so on), into a single model.

### Scalability

WCF services scale, and they scale in all directions. Not just up or out, but in all directions. They scale out via routing mechanisms and farms. Remember the book publisher example? The Order Process service was scaled out by providing an Order Process router, which routed orders to multiple Order Process services.

Services scale up by not being tied to a single OS or processor. Services scale up by the pure ability to deploy them on bigger and better servers and taking advantage of the new processor technologies that are starting to appear.

Services scale in by way of cross-process transports, meaning on-machine and cross-machine messaging and Object-RPC.

Services scale down by interfacing and communicating with devices such as printers, scanners, faxes, and so on.

### Interoperability

How sweet is it to be able to build high-class services using a single programming model and at the same time take advantage of earlier technologies (see "Programming Model"), irrespective of the OS, environment, or platform? WCF services operate independent of all of these.

WCF services also take advantage of the WS architecture utilizing the already established standards as far as communication and protocols are concerned.

### Enhanced Communication

Services aren't picky as far as transports, formats, or much else. You as a developer can choose from a handful of transports, different message formats, and surplus of message patterns.

Along these same lines, WCF is like the country of Switzerland (nice segue, heh?), in that services are neutral as far as transports and protocols are concerned. A service can use TCP, HTTP, Named Pipes, or any other protocol to communicate. The same goes for transports. In fact, if you want to build and use your own, feel free to do so.

The reason it is this way is because, as you hopefully have figured out by now, communication is completely separate from the service. They are completely independent from one another.

### Enterprise Enabled

A lot of times there is a give-and-take relationship when dealing with web services, interoperability, and other important features geared toward enterprises. As a developer you have to weigh performance versus security, or reliability. At what cost does adding transactional capabilities add to your solution? Up until now, having the best of all worlds was a mere pipe dream.

Well, now it is time to wake up and smell the technology because WCF provides the ability to have security and reliability without sacrificing performance. And you can throw transactions into the mix as well.

A lot of this comes from the standards of the web service architecture, allowing you to build enterprise-class applications.

Now that you know what makes WCF tick, the chapter wraps up by discussing some of the great things you can do with WCF.

## WCF Features

A lot of the topics just discussed can almost be included here, because things such as communication, scalability, and the enterprise are very much considered features as well. Yet they were put in their own section because those are things that define the great characteristics of WCF. As you read this section, keep those topics in mind also, because they are indeed features of WCF.

This section, however, discusses a few of those topics that make WCF feature rich. You know, those topics that make you say "whoa, that is cool." Though this list is by no means complete, it hopefully lists the top few. This chapter has already discussed communication and the programming model (all of which are discussed in greater detail in later chapters), so what are those features?

### Transactions

A transaction is a unit of work. A transaction ensures that everything within that transaction either succeeds as a whole or fails as whole. For example, if a transaction contains three items of work to perform, and during the execution of that transaction one of those items fails, then all three fail. The transaction succeeds only if all three statements of work succeed, unless a Checkpoint is issued. You commonly see this in database operations.

WCF incorporates this same transactional processing into its communication. You as a developer can now group communications into transactions. On the enterprise level, this feature lets you execute transactional work across different platforms. Transactions are discussed in Chapter 9.

### Hosting

WCF hosting allows services to be hosted in a handful of different environments, such as Windows NT Services, Windows Forms, and console applications, and well as IIS (Internet Information Services) and Windows Activation Services (WAS).

Hosting a service in IIS has added benefits in that the service can take full advantage of many of the native IIS features. For example, IIS can control the starting and stopping of the service automatically. Hosting is discussed in Chapter 15.

### Security

What good would Windows Communication Foundation be without security? Trust me on this, WCF certainly isn't lacking in this department. Everything from messages to clients and servers get authenticated, and WCF has a feature that ensures messages aren't messed with during transit. WCF includes message integrity and message confidentiality.

WCF also enables you to integrate your application into an existing security infrastructure, including those that extend beyond the standard Windows-only environments by using secure SOAP messages. Security is discussed in Chapter 10.

### *Queuing*

If you are at all familiar with MSMQ (Microsoft Message Queuing), this topic will sound familiar. WCF provides queuing, allowing messages to be safely stored, providing a consistent state of communication. Queuing collects and stores sent messages from a sending application and forwards them on to the receiving application. This provides a safe and reliable message delivery mechanism.

WCF enables queuing by providing support for the MSMQ transport. Queuing is discussed in Chapter 9.

Of course there are other features. Each of us can put together our own list of nice features, so hopefully you will add your favorite features to this list.

## Summary

Windows Communication Foundation is Microsoft's next step in building service-oriented applications. This chapter began by building a nice foundation, delving a little bit into Service-Oriented Architecture. This foundation is important because it helps you understand the rest of the chapter, the importance of Widows Communication Foundation, and what it can do for you.

The discussion of SOA talked about what SOA is and some of the principles that SOA exhibits. An example was given, the book publisher, which showed how using SOA a system can be upgraded and improved without disrupting other components, keeping the entire process intact.

From there the chapter moved on to introducing Windows Communication Foundation (WCF) and discussed its architecture, benefits, and capabilities. This section took a look at the layered architecture of WCF and explained what each layer provides and their purpose. The makeup of WCF was then discussed to better understand what makes WCF tick. Lastly, the chapter covered some of the great features of WCF.

The next chapter discusses the basic Windows Communication Foundation concepts needed to proceed through the rest of the book.