1

An Introduction to Workflow and Windows Workflow Foundation

This chapter gives you an overview of how business applications were and are traditionally developed, as well as an introduction to workflow and the Windows Workflow Foundation platform.

A Little Background

Initially, computers were used at universities to solve mathematically complex problems. The use of computing power was limited to the world of academia for a period of time before the world realized computers could be applied to solve business problems. Thus began the era of business applications.

If you are reading this book, you likely have some involvement in the world of business applications. Maybe you write .NET code, C++, Java, SQL, or other language to help businesses achieve strategic or cost savings goals. If so, you play an important role in the success of modern business.

Traditionally, a business decides that a given project involving information technology is worth doing because it will give the organization a competitive advantage, trim operating costs, or automate a complex manual process. Generally, the project's software developers gather requirements from the business, perform system and software design, and create the source code. Of course, any software development process worth its salt is much more complicated than that, but you get the idea.

The software development process has evolved greatly in the 50-plus years that businesses have been using computers to help solve business processes. In the not-too-distant past, procedural code was the de facto method for implementing software solutions. Over the past 10–15 years,

object-oriented code has given developers a great way to build reusable blocks of code that can be mapped to real-world objects. This building-block approach, when used correctly, can lend itself to helping developers build software solutions more efficiently and quickly.

Order processing, new employee processing, and insurance claim processing are just a few examples of the types of business processes that can be automated. These processes are modeled and documented, at which point a software developer creates his or her interpretation of the process with source code. It is very common in the stages before the actual coding begins for a business analyst to capture the steps in a process and represent the process graphically or with a list of tasks that must be completed and in what order. At this point, the perfectly nice list or visual representation is reduced to source code so that a machine is able to perform the process.

This approach might work in many instances, and in reality, it has been working for many years now. That doesn't mean there isn't a better way. Although this book is dedicated to conveying the technical aspects of Windows Workflow Foundation, understanding the problem domain of workflow and business processes can help you, as a developer, relate the technology to its business drivers.

What Is Workflow?

Workflow is a word that means different things to different people. As you may suspect by the name, workflow defines a process flow or a set of tasks that produces a result. Although this is a good general definition, workflow is also commonly discussed in the context of software systems.

Workflow systems are often associated with the domain of document management. In general, document-management systems handle the transfer of documents between people and other software systems according to a set of policies and procedures. Although document management may be the default concept that comes to mind, it is by no means the only area associated with workflow.

In addition, an archetypal feature of workflow is graphical representation. Workflows are frequently shown as a series of shapes and icons that represent discrete actions. Arrows are also commonly used to represent the flow of work from one step to the next. Although document management and visual representation may represent a large part of workflow, they are really so much more.

Before going too deep into workflow, you need to understand why it is important. One of the top priorities of IT organizations in recent years automating and supporting key business processes. Business process management (BPM) is the framework that describes how organizations can achieve these goals.

Business Process Management

BPM has gained a lot of attention in recent years. Essentially, it is a set of activities that organizations can follow to better understand, implement, and monitor business processes. There are many resources available to you that discuss BPM, its traits, and implementation methods. However, in the context of this book, you need only a good understanding of BPM's high-level qualities and how it relates to workflow.

BPM is commonly broken down into discrete phases that take an organization from one level of maturity to the next. You can find many different descriptions of what these phases are by searching online. However, the following phases are generally represented in one form or another:

1. Design
2. Execution/deployment
3. Management/monitoring
4. Optimization

During the design phase, an organization takes a good hard look at the processes that support the business. Emphasis is placed on fully understanding individual processes and the steps they entail. This might sound fairly obvious, but to fully understand a process and determine where there may be efficiency gains, this phase is crucial. The output of this phase is typically documentation that details the design discoveries.

Typically, the next phase in the BPM lifecycle is to implement the processes that were documented in the design phase. This happens by either making personnel behavior modifications or by implementing or updating technical solutions. There are commercial products available to assist in this phase of the BPM lifecycle from vendors such as Microsoft and TIBCO. These systems are specifically geared toward the problem domain of process development, execution, and management.

Process monitoring describes the step in the lifecycle in which processes are tracked and examined during normal day-to-day operations. For example, the business may be interested to know how many orders are currently in the shipping stage of an order fulfillment process. This is a very important quality of a BPM implementation because if you cannot monitor what is going on with business processes, the metrics that the processes generate cannot help an organization learn and improve.

Monitoring is crucial for the last phase: optimization. Monitoring, when implemented effectively, can help expose issues in processes that were not identified during the design and deployment phases. As you might imagine, these phases are not performed just once — this is an iterative cycle of refinement and improvement.

Of course, for an organization to consider implementing BPM, there have to be some benefits to doing so. If a company better understands its processes and process components, the most obvious advantage is a decline in errors related to performing processes throughout the organization. In addition, because processes implemented through the BPM lifecycle produce valuable business metrics, it is much easier to monitor this information with reports, alerts, and other types of human-consumable data. Better yet, this data can be made available in real time so that adjustments to the process can occur much quicker than in the past, saving precious time and money. Overall, BPM can provide organizations with a larger degree of understanding of its processes while lending itself to better decision making and a higher degree of agility.

Workflow Tenets

According to Microsoft, there are four major tenets that architects and developers can use when considering workflow-based applications. Furthermore, a workflow platform, which can be defined as a software

framework to assist in the development of workflow-based applications, should embody these features. The tenets are as follows:

- Workflows coordinate work performed by people and software.
- Workflows are long running and stateful.
- Workflows are based on extensible models.
- Workflows are transparent and dynamic throughout their lifecycle.

Workflows Coordinate Work Performed by People and Software

This tenet tells us that people play a vital role in the world of software systems related to workflow and processes. Human interaction often occurs through e-mail, web pages, mobile devices, or other front ends. In some instances, the interface between people and software can be part of the normal flow of a process. For example, a workflow might require a human to approve every transaction that comes through an e-commerce website. Human interaction may also be necessary to handle exceptions that cannot be managed in an automated fashion. This scenario may arise when a piece of information necessary for workflow progression is missing.

Because of this requirement, a workflow platform should provide features and infrastructure to effectively handle human interaction and all the issues that come along with it. This includes sending and receiving messages in an untimely manner. People cannot always be relied upon to act quickly or consistently.

Workflows Are Long Running and Stateful

This tenet is important due in large part to the previous one. Because humans are inherently not reliable and tend to interact with software systems on an ad hoc basis, workflows need to be able to run for long periods of time. More specifically, they should be able to pause and wait for input. This could be for hours or months or even longer. Consider a vacation-request workflow at a company. If an employee's manager is out of town, and the employee is requesting a vacation occurring nine months from now, the manager may not respond for weeks.

In addition, a workflow that coordinates software services that are external to an organization cannot rely on instant responses from these peripheral systems. Because of these characteristics, workflows need to be able to run, or at least be waiting to run, for an undetermined amount of time.

The stateful requirement means that the context of a workflow instance should remain intact while the workflow is waiting for feedback. Consider the vacation workflow that goes on for many weeks. If the workflow is not able to save its state, it may be lost forever if the server it is running on is rebooted due to a power outage or other issue.

Workflows Are Based on Extensible Models

The purpose of workflow is to automate a business process, and because each type of business has a wide range of problems, a workflow platform should be extensible at its core. What works for one business problem domain may not apply to another. It is reasonable that a workflow platform cannot account for every kind of activity that needs to occur in every type of workflow. The great thing about a well-architected, extensible system is that developers are able to build the components that were not included out of the box.

To apply this idea to another area, the workflow management architecture should also be extensible. Most workflow platforms provide functionality for persistence, tracking, and dynamic modifications. These areas should be open for extension by developers so that an organization's needs that were not covered in the platform's base functionality can be achieved.

Workflows Are Transparent and Dynamic Throughout Their Lifecycle

This tenet is easier to understand when compared to the traditional software development paradigm. In conventional software, the code itself defines the behavior of the system. As you know, writing and understanding code are very specialized skills that a very small percentage of the business population possesses. Because of this fact, software systems are generally considered to be a black box. People in the business cannot look at a block of code and ascertain what is happening behind the scenes. This can even be difficult for a developer if the code was written by someone else. Workflows should provide the advantage of being able to quickly and easily determine functionality at design time — that is, when the workflow is not running.

Additionally, workflows should be transparent during runtime. This means that a workflow should be able to be queried so that progress can be monitored while it is running. If you take this transparent runtime concept a step further, a running workflow's steps that have not yet occurred should be modifiable. Compare this to traditional code, which, after it is compiled, cannot change itself. This notion of a workflow that can be changed during runtime is very powerful and can open a new set of possibilities related to process automation.

Types of Workflows

At the risk of being too general, there are two types of workflow: ordered and event-driven. Both types provide distinct functionality and would be used in different situations, depending on the requirements of a given system.

Ordered Workflows

When people think of workflow, the first thing that comes to mind is probably an ordered workflow. An *ordered workflow* represents a process that starts with a trigger and flows from one step to the next in a predefined order. The process likely contains control or decision-making logic that includes `if-then` statements and `while` loops.

The steps in an ordered workflow and the order in which they occur are non-negotiable based on the workflow's definition. For example, in an order processing scenario, a customer's credit check *always* occurs before the order is shipped. It wouldn't make much sense to swap those two tasks.

Event-Driven Workflows

Event-driven workflows, sometimes called *finite state machines (FSM),* are based on the idea that you start in one state and jump to another state based on a specific activity or event. For example, a light switch starts in the *off* state. When someone performs the *turn on light* action the light switch transitions to the *on* state.

Like ordered workflows, event-driven workflows have rules that describe which states can transition to other states based on some predefined events. It would not make sense for the light switch to transition to the *on* state if someone unscrewed the face plate.

Workflow Scenarios and Examples

Now that you have a good handle on what traits a workflow platform should possess and what types of workflows exist, the following sections discuss various workflow scenarios and illustrate a few supporting examples.

Human-to-System Interaction

The aforementioned workflow tenets mentioned that human-to-system interaction is vital in the context of workflow-based systems. The importance of this principle cannot be overemphasized.

System-to-System Interaction

As you now know, it is important for workflows to be able to interact with people. It is equally important that systems are able to interact with other systems through a defined workflow. This process of tying external systems together in one cohesive set of steps is sometimes referred to as *orchestration.* If you are familiar with Microsoft's server product BizTalk, you have probably heard this term.

This type of workflow is commonly associated with service-oriented architecture (SOA). SOA is a large topic that is discussed briefly later in this chapter and in subsequent chapters of the book.

Application Flow

A workflow does not have to be an interaction between entities that are external to each other. A common scenario is defining the order in which to display data entry forms or other interface to a user. Think about a college application website. The order in which pages are presented to the applicant is important and is probably based on the education program to which the user is applying.

Another scenario might be a system in which data is processed from a beginning state to an end, cleansed state. Along the way, the data is placed into different buckets that represent different states of being processed. If during a given state, the data cannot be processed because of an extraneous issue, an exception is raised so that a human is forced to inspect and manually fix the problem. This scenario could be applied to customer data coming in from multiple outside sources.

A Few Examples

To get you in the overall workflow mindset, the first example is not a model for a software system at all. It is a set of instructions for an IT recruiter to follow related to an employee hiring process. Figure 1-1 illustrates what this type of process might look like.

The steps could obviously go on from here, but this should give you an idea of what an *ordered workflow* might look like. The first step is always the receipt of an application and résumé, and the next step is always the review for job requirements. The order of the tasks is non-negotiable.

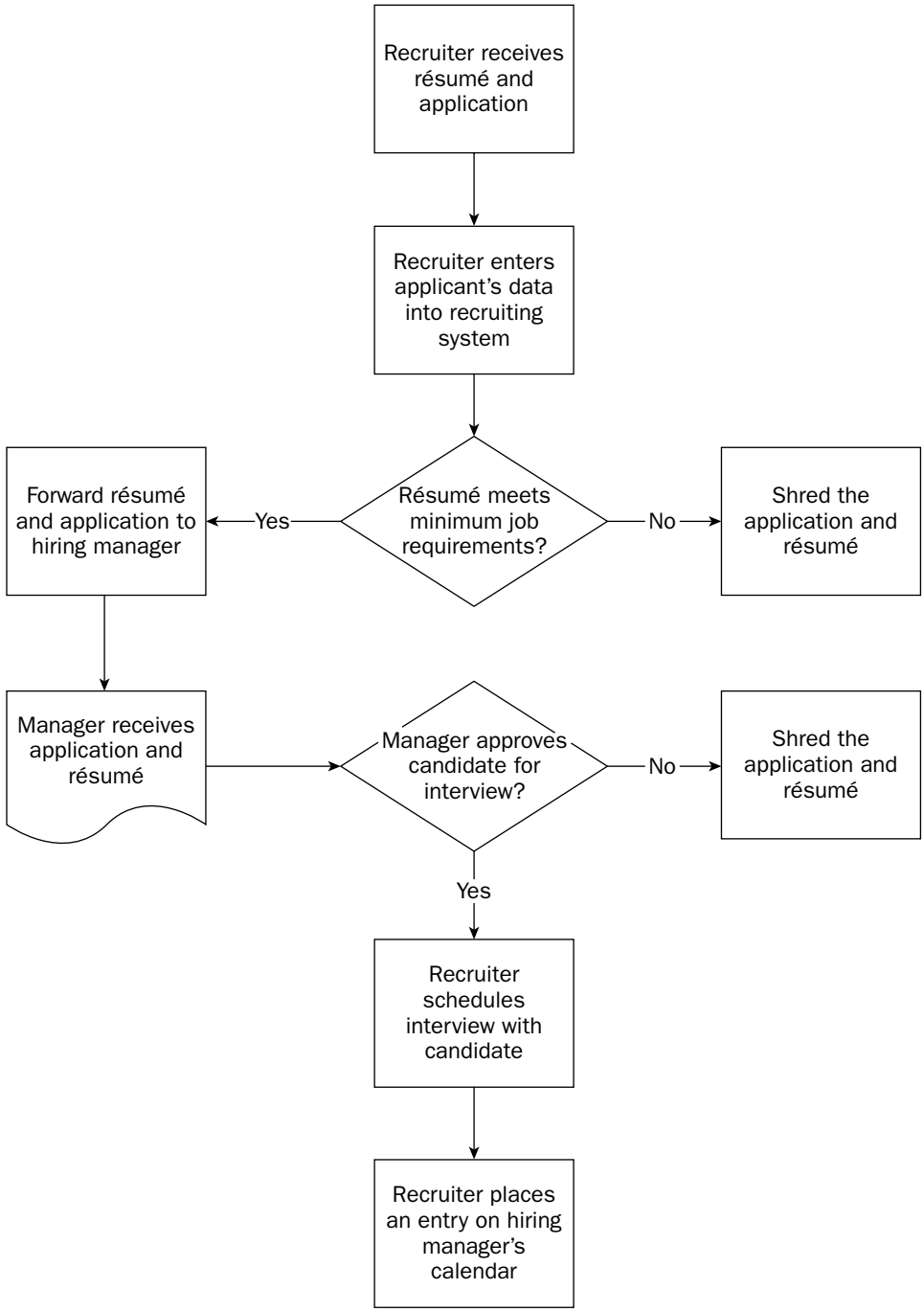
Recruiter receives résumé and application
Recruiter enters applicant's data into recruiting system
Résumé meets minimum job requirements?
Yes
No
Forward résumé and application to hiring manager
Shred the application and résumé
Manager receives application and résumé
Manager approves candidate for interview?
No
Shred the application and résumé
Yes
Recruiter schedules interview with candidate
Recruiter places an entry on hiring manager's calendar

Figure 1-1

Conversely, a workflow does not have to be a set of ordered tasks. The *event-driven workflow* concept introduced previously still has discrete steps that occur to advance a process, but these steps take place based on some action or trigger. For example, a workflow that tracks software bugs may have states to represent bugs that are active, fixed, and irreproducible (see Figure 1-2). A developer is first assigned a bug by a tester, at which point the bug is active. After the code has been modified, the bug enters the fixed state, a tester verifies the fix, and the bug is closed. However, what if the bug was not really fixed by the developer? Instead of progress the bug to the closed state, the tester sends it back to the active state. This cycle could happen over and over until the bug is actually eradicated. Of course, this state-based workflow needs to jump logically from one state to another. It doesn't make sense for a bug to start in the closed state and then transition to the fixed state.

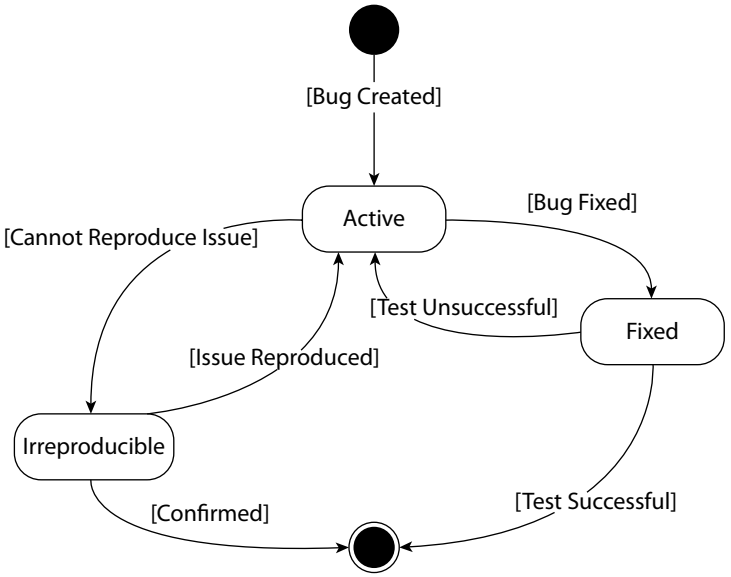

Figure 1-2

As previously mentioned, a workflow can define the interaction between people and systems. For example, an order processing software application might notify a consumer if his or her credit check fails, prompting him or her to provide an alternative form of payment. In addition, this same order processing workflow could call a remote inventory web service to make sure the desired item is available for shipment. Both examples describe a scenario in which the workflow hands off a portion of work to an external entity.

Workflow Implementation

You can implement a workflow in many different ways. The following sections describe a few examples and how each relates to software systems.

Conceptual Implementation

Throughout history, the most common implementation of workflows probably involved no code or software. Imagine all the process flow diagrams developed over the years that represent manual processes. Just because a process is not automated does not mean it is not a workflow.

These types of workflows were developed using a variety of tools, from paper and pencil to Microsoft Visio or other modeling medium. Imagine the court system over 100 years ago or some kind of medical procedure. They all involve a series of steps and a decision-making process.

Workflows with Code

Take a look at the following code listing:

```
public void ProcessOrder(Order order)
{
   if (order.Lines.Count > 0)
   {
      if (order.Customer.CreditCheck())
      {
         if (order.Lines.AllItemsInStock())
         {
            order.ShipOrder();
            return;
         }
      }
   }

   order.Cancel();
}
```

Although this is not a graphical diagram with shapes and arrows, it is a workflow. It uses specific information, in this case an `Order` object, to make decisions and produce an outcome. As long as the order has line items and all items are in stock, it will be shipped. Otherwise, the order will cancel itself, perhaps sending the customer an apology letter.

Object-oriented or procedural code is probably the most common way to implement workflows in modern software. Think about the projects you've worked on. Most likely you have automated some kind of business process.

Although this is probably the most popular way to implement workflows today, it comes with its own set of issues. One of the biggest problems with code is that the only people who can understand it are software developers. Although this may provide job security for some individuals, it doesn't make for a happy business community.

Line of Business Systems

A *line of business* (LOB) system helps a business do something it is probably already doing. LOB systems can range from the monstrous SAP and Oracle Financials to smaller, more specific applications that handle tasks such as customer relationship management, order processing, or human resource-related activities.

LOB systems can solve business problems common across many organizations, but they solve only very specific problems. Although you may be able to customize an out-of-the-box process, you would probably not develop a workflow from scratch inside an LOB. That is the kind of problem better left for workflow engines.

Workflow Engines

A *workflow engine* is a piece of software that provides a way to declare, modify, and monitor a workflow process. For example, you could create a workflow for an engine that defines a patient admittance process in a hospital emergency room. Then when the process changes due to a new privacy law, you could modify it accordingly. Also, because the hospital staff is interested in potential bottlenecks in the procedure, you could use metrics to report how many people are waiting to see the doctor.

Custom Workflow Engines

Because there has not been a widely accepted, compelling answer to the problem of workflow, many developers have approached the problem individually. You may have worked on a project where the concept of workflow was so prevalent that a custom engine was developed just for that initiative.

Commonly, these engines are data driven. For example, steps in a process may be represented by rows in a database table or nodes in an XML file. This makes changing a process relatively easy, and using this model does not require changes to code. Another advantage to developing a custom engine is that it is tailored to the exact needs of the project at hand.

Although developing a custom workflow engine has some benefits, such as ease of process definition and modification (depending on how well your engine is architected), there are a few problems with this approach. First and most obvious, developing a workflow engine takes time. This is time that could have been used for solving a business problem, which is what you're getting paid to do. Second, because you wrote the workflow engine, you have to maintain and support it. Again, this shifts the focus from solving a business problem to supporting a technology problem. Finally, a custom workflow engine probably cannot be all things to everyone. Off-the-shelf engines, which are discussed next, commonly contain functionality for tracking and visualization. It would be quite a task to develop such software when that is not your main business.

Off-the-Shelf Workflow Engines

The free-market system is a wonderful thing. When a need arises, invariably someone invents something to fill the void. Workflow technology is not exempt from this idea. Many vendors offer workflow products to solve the problems discussed in this chapter.

Generally, these products offer features such as workflow definition, management, tracking, persistence, and so on. Going with an off-the-shelf solution has many advantages. Aside from the advantages associated with workflow engines in general (mentioned previously), using a third-party vendor frees you from having to maintain an extra set of code. This relates to the classic buy-versus-build dilemma. In most cases, it is cheaper and more efficient to purchase a workflow product rather than build it yourself. Of course, there are exceptions to the rule, and you should evaluate the needs of your organization before making any decision.

A potential disadvantage of buying a workflow platform is the reliance on the vendor for support and updates. If the vendor goes out of business or decides to start manufacturing hats for small dogs, you are left with the responsibility of supporting an obsolete technology.

This book is not meant to be a commercial for any particular company; however, it is probably prudent to mention at least the most prevalent workflow products. K2.net (`www.k2workflow.com`) provides a platform to develop, run, monitor, and administer workflows. Skelta Workflow.NET (`www.skelta.com`)

provides similar functionality. Interestingly, since the announcement of Windows Workflow Foundation, both of these companies have stated that the next generation of their respective products will be built on top of the Microsoft workflow software.

Windows Workflow Foundation

Windows Workflow Foundation, sometimes called Windows WF, was developed by Microsoft to provide developers with a single platform on which to develop workflow- or process-based software solutions. Windows Workflow Foundation is built on top of .NET and is a major component of the .NET Framework 3.0.

An Introduction to .NET and the .NET Framework 3.0

Because Windows Workflow Foundation is built on .NET and is part of the .NET Framework 3.0, the following sections provide some background and context related to those technologies.

The .NET Framework

At its core, the .NET Framework is a platform that supports application development and the ability to run software built on the framework. The types of software that you can build on the .NET Framework include web applications with ASP.NET, smart client applications, and XML web services, to name a few. Figure 1-3 is a graphical representation of the .NET Framework stack.

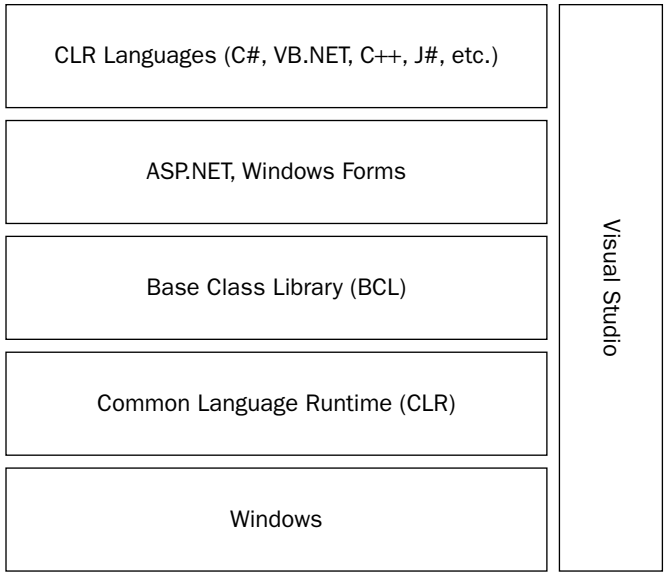

Figure 1-3

The .NET Framework also provides a managed runtime that allows developers to not worry about memory management as part of writing software. Source code is compiled into *Intermediate Language*

(IL), which is a just-in-time (JIT) compilation at runtime. This means that the IL is changed into machine language that the current operating system can understand and run. This concept enables you to develop software in virtually any language that compiles to IL. Microsoft provides several languages, such as C#, Visual Basic .NET, and managed C++.

Another main pillar of the .NET Framework is the Base Class Library (BCL). The BCL is an extremely large and rich set of classes that provide out-of-the-box functionality for developers. This includes classes for string and XML manipulation, ADO.NET classes for database interaction and structured data operations, and much more.

The .NET Framework 3.0

The .NET Framework 3.0 is a next-generation development platform that is available out of the box on Windows Vista and can also be downloaded for other versions of Windows, including Windows XP SP2 and Windows 2003 Server. Essentially, the .NET Framework 3.0 is an extension of .NET that provides developers with a library of managed APIs. The .NET Framework 3.0 components are as follows:

- Windows Presentation Foundation (WPF; formerly known as *Avalon*)
- Windows Communication Foundation (WCF; formerly known as *Indigo*)
- Windows CardSpace (WCS; formerly known as *InfoCard*)
- Windows Workflow Foundation (WF; formerly known as *WinOE*)

Figure 1-4 illustrates the .NET Framework 3.0 architecture.

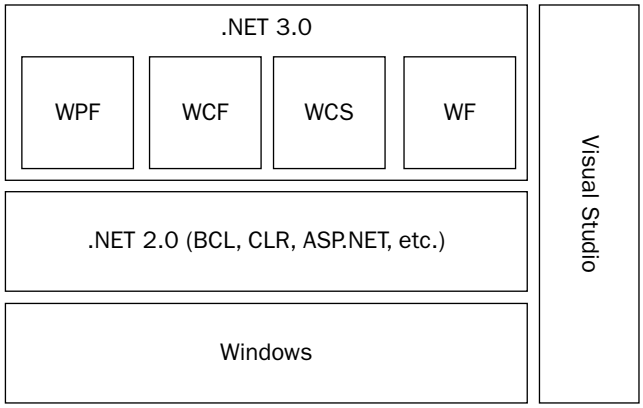

Figure 1-4

Windows Presentation Foundation

Windows Presentation Foundation (WPF), formerly known as Avalon, provides a framework for developing a rich user interface experience. Unlike Windows Forms, which is driven by code, WPF can be developed with a declarative markup model. This markup is called *XAML* (pronounced *zamel*), which stands for *Extensible Application Markup Language.*

XAML is XML based, which means that all you need to develop a user interface layout is your handy Notepad executable. Check out the following code listing for a simple example:

```
<Window x:Class="XamlSample.Window1"
 xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="XAML Sample"
 Height="150" Width="200">
  <DockPanel>
    <Button Width="100" Height="50">I am a button!</Button>
  </DockPanel>
</Window>
```

This creates a form that looks like Figure 1-5.

Figure 1-5

Although WPF and XAML are separate Windows Workflow Foundation components, XAML plays a role in the declarative model of workflow development (which you learn more about in later chapters).

Windows Communication Foundation

Windows Communication Foundation (WCF), formerly known as Indigo, is a framework on which you can create SOAs based on industry standards.

SOA is a buzzword of recent years that can cause a lot of controversy when people of the technical persuasion get together to discuss it. Essentially, SOA describes an architecture of loosely coupled services (usually web services) that expose a unique and discrete functionality. Usually, these services perform one very small task, but they perform it well.

Building an SOA at an enterprise is a lot different from building a singular application. SOA provides the foundation to build applications — it is not an application itself. Imagine a company that manufacturers and sells widgets at retail outlets. The organization might have services to submit a bill of materials, to order new widget glue, or to receive and process purchase orders.

Now back to Windows Communication Foundation. One of the biggest advantages of WCF is that it provides a unified platform on which you can develop distributed applications. In the past, if you wanted to develop a distributed application on the Microsoft platform, your options included ASP.NET Web Services (ASMX), .NET Remoting, Enterprise Services, WSE, and MSMQ, to name a few. That's a daunting list, isn't it?

WCF uses industry standards such as SOAP to bring all these technologies together. This enables applications built on the Microsoft stack to interact with technologies from other vendors, such as Java. Essentially, WCF is the replacement and glue for these technologies of yesteryear.

Chapter 14 discusses SOA and WCF in greater detail.

Windows CardSpace

Windows CardSpace is a next-generation identity platform. CardSpace is built on open standards and tries to succeed where Microsoft Passport failed. Passport was not able to live up to its grand promises basically because it required your personal information to be stored with Microsoft. CardSpace reverses the location of personal information to your local machine in *identity cards.* An easy analogy is to consider the cards carried in your wallet, such a driver's license or library card. Whenever you get pulled over, your license acts as proof of who you are. This is because the card was issued by the state you live in, and the friendly police officer trusts the state, not you.

CardSpace uses this same concept. An identity provider issues identities that are trusted by other sites on the Internet. For example, a credit card company could issue an identity to you that represents a physical card you already have. When you purchase something from an online store, you could present the issued card instead of signing in with a user name and password as you would traditionally. The benefits are twofold in a situation like this. First, you do not have to maintain a user name and password combination for every site you frequent. Second, there is an added layer of security because you are not actually entering credit card information. Rather, because the online store trusts the identity issued by the credit card company, it can use it to take care of the payment details.

In addition to the identity cards issues by third-party providers, you can create self-issued cards. This enables you to create a lightweight identity on a site that requires you to register to obtain access to content, such as a forum. Then the site, if it chose to implement CardSpace, could allow you to authenticate by presenting your self-issued card. This again has the benefit of not requiring multiple user names and passwords. In addition, you can create multiple self-issued cards. This enables you to have identities with varying levels of detail for different types of sites. Figure 1-6 shows an example of a self-issued card in Windows Vista.

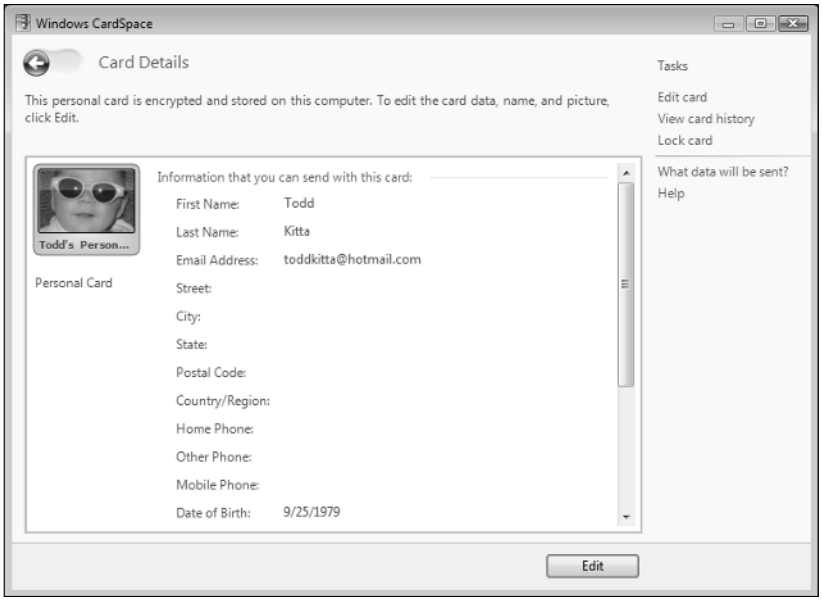

Figure 1-6

Windows Workflow Foundation

Last but certainly not least, Windows Workflow Foundation rounds out the .NET Framework 3.0 platform and is the main topic of discussion in this book.

Why Windows Workflow Foundation?

After reviewing all this information about workflows and process-oriented software, you might be wondering why Windows Workflow Foundation matters. The following paragraphs relate Windows Workflow Foundation to the four tenets of workflow discussed earlier. This should help you determine whether to use Windows Workflow Foundation in your software development efforts.

- **Workflows coordinate work performed by people and software.** This tenet is a key piece of a workflow platform, and Windows Workflow Foundation provides several features that can help you achieve this goal. Workflows built on this framework allow human interaction with basically any interface imaginable — e-mail, web forms, windows forms, instant messaging, InfoPath — the list goes on and on.

  The important thing to remember is that the workflow realizes when it is requesting feedback and waits until the required data has been received into the context of the application before progressing. With the Windows Workflow Foundation platform, the possibilities are limited only by your imagination.

- **Workflows are long running and stateful.** Windows Workflow Foundation provides a rich framework of runtime services. These services, which are discussed in detail in Chapter 7, offer you an extensible framework to, among other things, persist running workflows to a durable medium. This is important because workflows can run for long periods of time, and storing a workflow's context in live memory isn't practical for many reasons.

  If every running workflow in an enterprise had to be stored in memory while waiting for something to happen, the scalability of the system would be nonexistent. The server would run out of memory in no time. In addition, if the server crashed, the volatile memory would be cleared and all data would be lost.

  Workflows need to be reliably stateful. Out of the box, Windows Workflow Foundation provides a way for you to persist a workflow's state to a stable channel such as a database. You can extend these persistence services to store workflow state just about anywhere.

- **Workflows are based on extensible models.** This is a large part of Windows Workflow Foundation — just about every part of the platform is extensible. Workflows are made up of discrete actions called *activities.* Windows Workflow Foundation provides base activities as well as basic and generic workflow functions. You can extend these activities to meet the needs of essentially any requirement. You can also develop new activities from scratch.

  There are many other parts of the platform that are extensible as well. Runtime services provide functionality related to tracking, management, and persistence — which are all extensible.

- **Workflows are transparent and dynamic throughout their lifecycle.** Windows Workflow Foundation meets this requirement in two areas: design time and runtime. Because Windows Workflow Foundation is based on a declarative and visual design-time model, processes are easier to understand. This means you can modify existing workflows without having to change source code.

During runtime, you can query the status and overall general health of workflows. The tracking features of Windows Workflow Foundation also enable you to log information to a persistent medium for later inspection. Finally, and very important, you can modify workflows built on this platform even during runtime. This provides for extremely flexible workflow scenarios.

Aside from the features related to workflow tenets, Windows Workflow Foundation supports both ordered and event-driven workflows. The ordered workflow is implemented as a *sequential* workflow (as depicted in Figure 1-7), and the event-driven workflow is implemented at a *state machine* (as depicted in Figure 1-8).

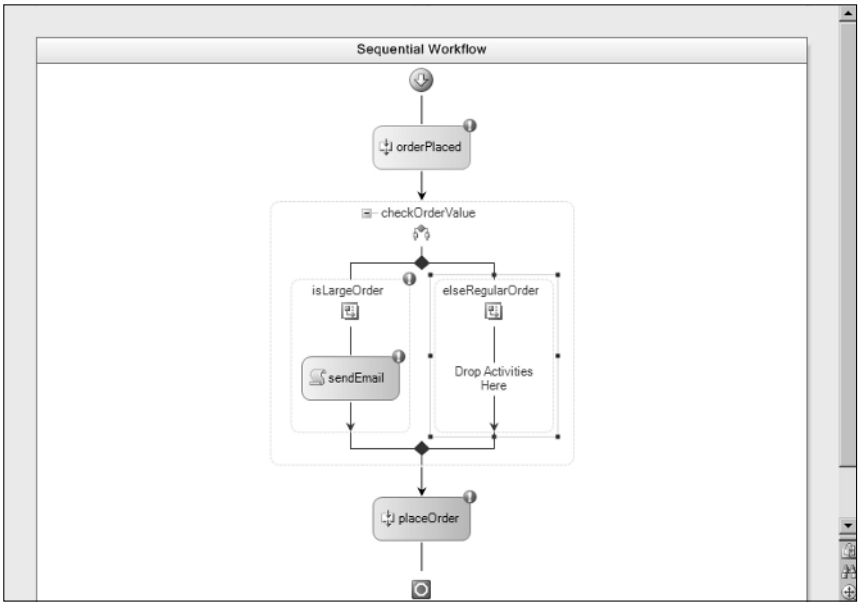

Figure 1-7

Who Should Care About Windows Workflow Foundation?

The following sections describe those who should consider using Windows Workflow Foundation and why. (This is by no means an exhaustive list.)

.NET Developers

The majority of people who can benefit from using Windows Workflow Foundation on a day-to-day basis are .NET developers. Just like it is a good idea to use the latest version of ASP.NET and SQL Server, boning up on workflow can add to your value as a software professional.

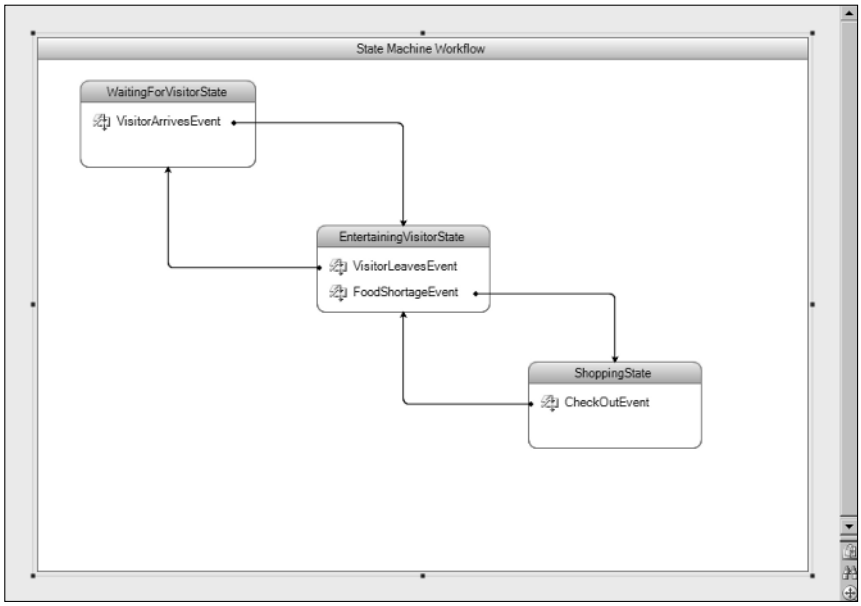

Figure 1-8

Think about some of the applications and projects you've worked on during your career as a developer. You can probably think of several instances where a workflow platform could have come in handy. Now think about workflow in the context of ongoing maintenance of a system that supports a business process. One of the biggest advantages workflow provides to developers is a more efficient maintenance model. Not only is it easier to work with a graphical representation of a process that you developed, but coming up to speed on a workflow someone else developed is also much simpler.

The tools that are part of the .NET Framework, such as ASP.NET, provide you with a rich development environment and enable you to focus on solving business problems rapidly. Windows Workflow Foundation is no exception. By using a platform explicitly geared toward the specific domain of business process, developers can garner a great deal of efficiency and focus.

Architects

It is the architect's never-ending job to evaluate what systems are going to look like from a macro and often a micro level. New technologies come along all the time, and it is up to the architect to determine how these technologies fit in the overall system landscape.

Architects can use workflow to add value to their development efforts and the makeup of a particular enterprise or individual project. They can also use workflows to interact with other pieces of technology. For example, a workflow could coexist and communicate with existing LOB systems.

Technology Leadership

Although most CIOs don't care about how Windows Workflow Foundation works or its technical architecture, they should care about some of the benefits such a tool can bring to an enterprise. For one, workflows provide a unified process-oriented development platform. The integration and consolidation of technologies is a key objective for many in technical leadership positions.

In addition, workflow helps bring process to the forefront of software projects. As anyone in a leadership position knows, recent legislation such as Sarbanes-Oxley requires organizations to have a good handle on their processes. Windows Workflow Foundation provides a framework for tracking and monitoring these processes.

ISVs and Service Providers

One thing Microsoft has been very good at, whether intentionally or not, is fostering a community of vendors that build software on and for Microsoft technologies. Even though Microsoft is the behemoth that it is, it cannot always build software that meets everyone's needs. This is where Independent Software Vendors (ISVs) and service partners come into play.

ISVs are great at catering to the needs of niche and even mainstream markets. For example, the web controls that come out of the box with ASP.NET, such as the GridView, are great and provide a nice foundation on which to create web applications. However, many industries require more robust controls with client-side capabilities and enhanced editing features. There are many grid controls on the market today that, for a relatively inexpensive price, provide developers with the flexibility and functionality they require.

No doubt it will be the same story with workflow. For example, Microsoft is including a few prebuilt workflows with Office and SharePoint 2007 for documentation and management. However, there may be complex scenarios for specific industries, perhaps healthcare, that will allow technology service providers and ISVs to meet a need not yet met by Microsoft.

Summary

This chapter gave you a little background on the history of software development and a crash course in workflow. You learned that because workflow is already a predominant function of traditional software development, it makes sense for workflow-specific platforms to exist.

There are several traits that a workflow platform should possess. Workflows should coordinate work performed by people and software, and should be long running and stateful, based on extensible models, and transparent and dynamic throughout their lifecycle.

You were also introduced to Windows Workflow Foundation, Microsoft's answer to the problem of workflow in software. Windows Workflow Foundation is part of the .NET Framework 3.0.

Finally, this chapter discussed who should consider using Windows Workflow Foundation and why.