Microsoft

# Workflows, Services, and Models

A First Look at WF 4.0, “Dublin”, and “Oslo”

David Chappell, Chappell & Associates

October 2008

# Contents

# Creating Modern Applications: Workflows, Services, and Models

Software gets more important every day. Organizations that once viewed IT solely as a support function now see it as essential to their business strategy. Yet building new applications, running those applications, and keeping track of what's going on across the IT environment remain demanding challenges.

To help organizations get better at meeting these challenges, Microsoft is releasing a group of new technologies. Used separately or together, these technologies can help in creating and running workflow-based, service-oriented, and model-driven applications. This overview provides an early look at this new wave.

## A Technology Overview

The focus is on three things: workflows, services, and models. Accordingly, this set of forthcoming technologies has three main aspects:

- A new and expanded version of Windows Workflow Foundation (WF), a .NET Framework technology for coordinating work done in software. Because it's part of the .NET Framework 4.0, this new release is known as WF 4.0.

- Extensions to the Windows Server application server, codenamed "Dublin", that provide improved server support for running and managing service-oriented business logic. "Dublin" extends the Windows Process Activation Service (WAS) and Internet Information Services (IIS) 7 with support for Windows Communication Foundation (WCF) and WF-based applications. "Dublin" will initially be made available for download and use by Windows Server customers, then later included directly in future releases of Windows Server.

- A group of technologies, codenamed "Oslo", aimed at creating and running model-driven applications and more. The "Oslo" technologies include a repository, providing a common place to store a range of information about your IT environment; a modeling language family, codenamed "M", for describing that information; and a modeling tool, codenamed Visual Studio "Quadrant", for working with repository information.

Each of these technologies can be applied independently, but they're also designed to be used in concert. Figure 1 shows one way to think about how the three fit together.

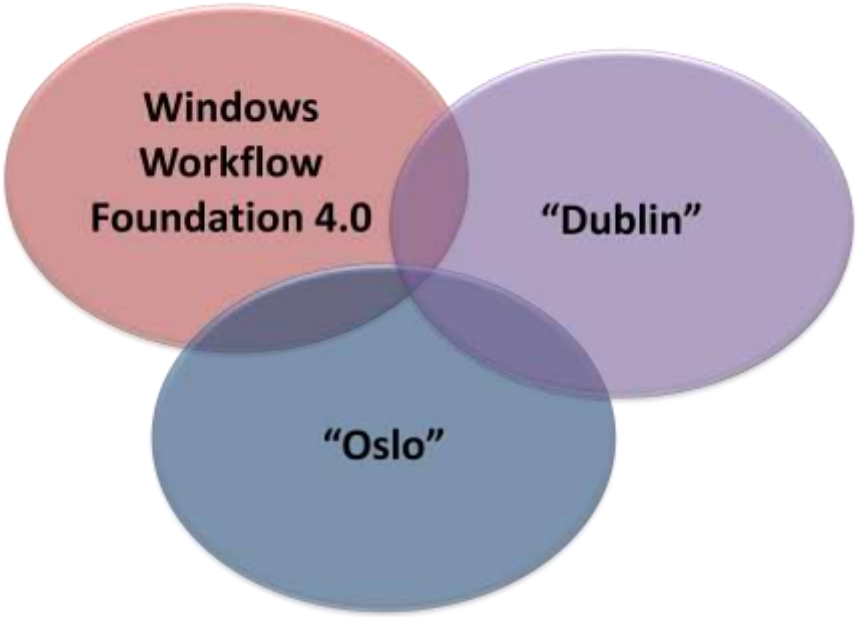


*Figure 1: WF 4.0, "Dublin", and "Oslo" can be used independently or together.*

This section provides a brief look at each technology, followed by a simple scenario showing how all of them might be used together.

### Workflows: Windows Workflow Foundation 4.0

What does software really do? In general, every application performs a set of actions in some order. Workflow technology formalizes this simple idea, providing explicit support for coordinating an application's work.

The word "workflow" means different things to different people. The first thing that comes to mind for many is human workflow, controlling tasks such as approving a document. Others think of system workflow, such as orchestrating interactions between various applications. WF isn't focused specifically on either of these areas. Instead, its goal is to provide a general foundation for all kinds of workflow-based software. This includes both human and system workflow, but it's also broader than that. The truth is that pretty much any kind of business logic might be built using WF. It's up to each application designer to determine when this technology is the right choice for solving a particular problem.

In WF, every workflow consists of one or more *activities*, as Figure 2 shows. These activities can be big, small, or in-between, and each one performs some part of the workflow's tasks. Depending on what kind of workflow a developer chooses to create, the activities might be organized in different ways, as described later.

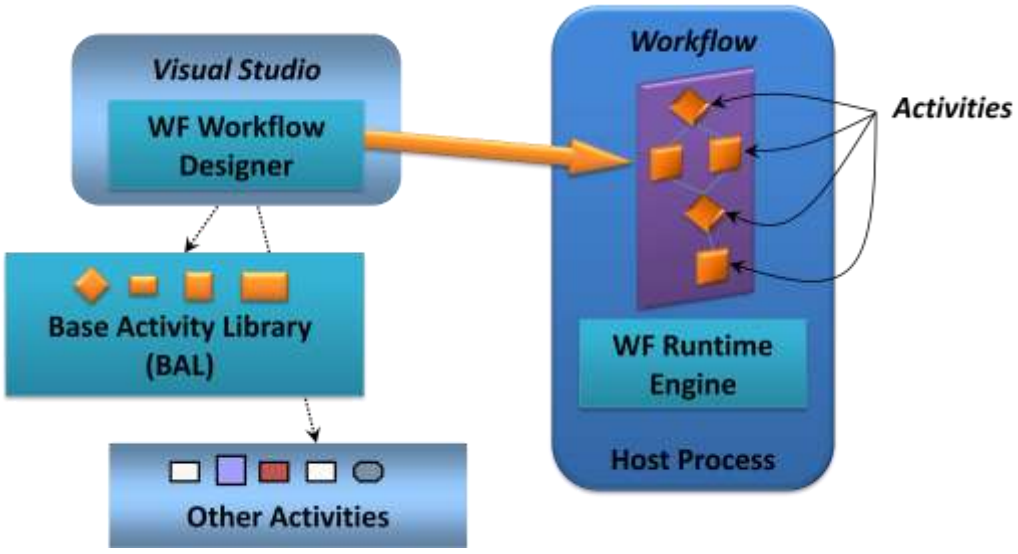


*Figure 2: A WF workflow is built from activities.*

As the figure shows, every WF workflow is executed by the *WF runtime engine*. This engine and any workflows that use it must run in some host process, but WF itself doesn't mandate what this process should be. This lets WF be used in a range of applications, from simple desktop apps to large-scale server processes—it's up to the application's creator.

Developers are free to create WF workflows directly in a language such as C# or Visual Basic. It's also possible—and usually more efficient—to create and modify at least parts of a workflow graphically using the *WF workflow designer*. This tool is hosted in Visual Studio, making it straightforward to switch between a graphical approach and working directly in code.

The WF runtime engine isn't wedded to any particular group of activities; developers are free to create any that they like. Microsoft does provide a standard set of activities, however, called the *Base Activity Library (BAL)*. Using these, a developer can define a workflow's logic, communicate with software outside of the workflow, and more.

WF 4.0 provides a new release of this workflow technology. As described in more detail later, a primary goal of this new version is to make WF easier for developers to use. While it's most often been applied in somewhat specialized contexts so far, the intent of this new release is to make WF a mainstream technology for application development.

## Services: "Dublin"

Where should an application's business logic run? Answering this question is sometimes easy. Many .NET Framework applications, for example, are just ASP.NET pages that access data in

a database. In simple situations like this, implementing logic directly in those pages and perhaps in a few stored procedures is fine.

For applications with more complex requirements, however, this approach doesn't work. Scaling logic created solely in ASP.NET can be challenging, for example, as can using it from other contexts. How can logic embedded in an ASP.NET page be accessed directly from another application, for example? In the increasingly service-oriented world we live in, allowing this access can be a fundamental requirement. What's needed in situations like these is a technology designed explicitly for hosting scalable, accessible, service-oriented business logic.

The goal of "Dublin" is to provide this. While it certainly isn't required to run WF or WCF applications, the support "Dublin" provides is likely to be useful in a variety of scenarios. Figure 3 shows a simple view of how this new technology fits into a typical application.

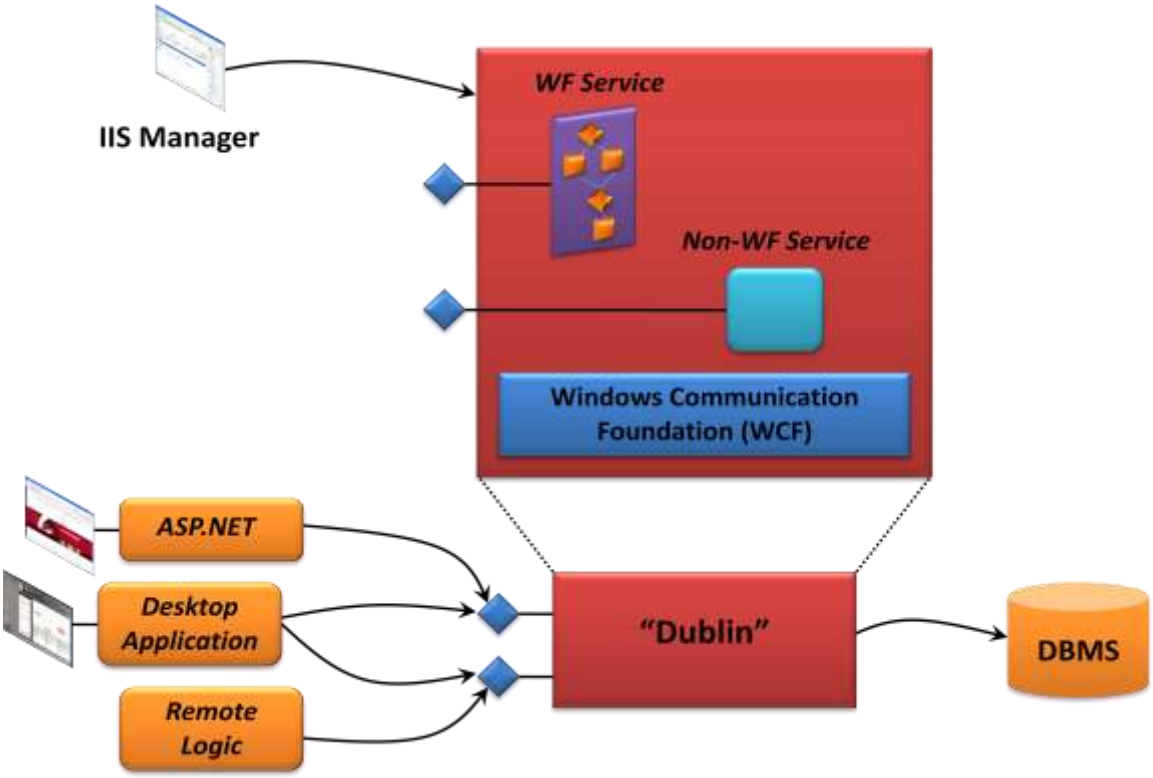


*Figure 3: "Dublin" provides a home for service-oriented business logic.*

As the figure shows, logic that runs in "Dublin" is implemented as one or more WCF services. Since each service is built on WCF, it can be exposed via Web services (using SOAP or a RESTful approach), through Microsoft Message Queuing (MSMQ), or in some other way. Those services can be accessed by ASP.NET applications, desktop applications, or other

remote logic (including software running on non-Windows systems). And like any other logic, services running in “Dublin” can access data in a database or invoke other services.

While “Dublin” can run any WCF-based service, it provides extra support for long-running services, i.e., software that stores its state, shuts down, then begins again where it left off. This kind of logic is exactly what WF is designed to create, and so it’s fair to say that the biggest beneficiary of “Dublin” will be people who create WF applications. Without “Dublin”, those developers are frequently faced with the task of building their own infrastructure and tooling for their workflows. One of Microsoft’s primary goals with “Dublin” is to provide an effective server for WF applications as part of Windows Server itself.

Yet “Dublin” isn’t only for WF applications. It’s possible to build long-running WCF services without WF, and these can take advantage of the “Dublin” support for persisting and restoring state as well. “Dublin” also provides functions that are more broadly useful, such as the ability to monitor and manage services. As Figure 3 shows, an IT pro can do this using IIS Manager, a standard component of IIS today. Using this tool, the administrator can start and stop services, control their operation, and more.

## Models: “Oslo”

All of us who work in IT face the same reality: We live in a complicated world. Even small organizations work with lots of information, while large organizations can find themselves drowning in it. Understanding business processes, creating applications that support those processes, keeping track of where those applications run, and thousands of other details threaten to overwhelm many IT departments.

One approach to taming this complexity is to provide a common place to store information about our world, modeling it in a more abstract and consistent way. Once this has been done, this information can be used in a variety of ways by a variety of tools. The goal of “Oslo” is to make this possible.

Toward this end, “Oslo” includes several related technologies:

- The “Oslo” repository, a common store for information represented as models.
- The “M” language family, providing a way to describe the kinds of information in the repository and more.
- The Visual Studio “Quadrant” modeling tool, helping people interact with the information in the “Oslo” repository.

Figure 4 shows how these three fit together.

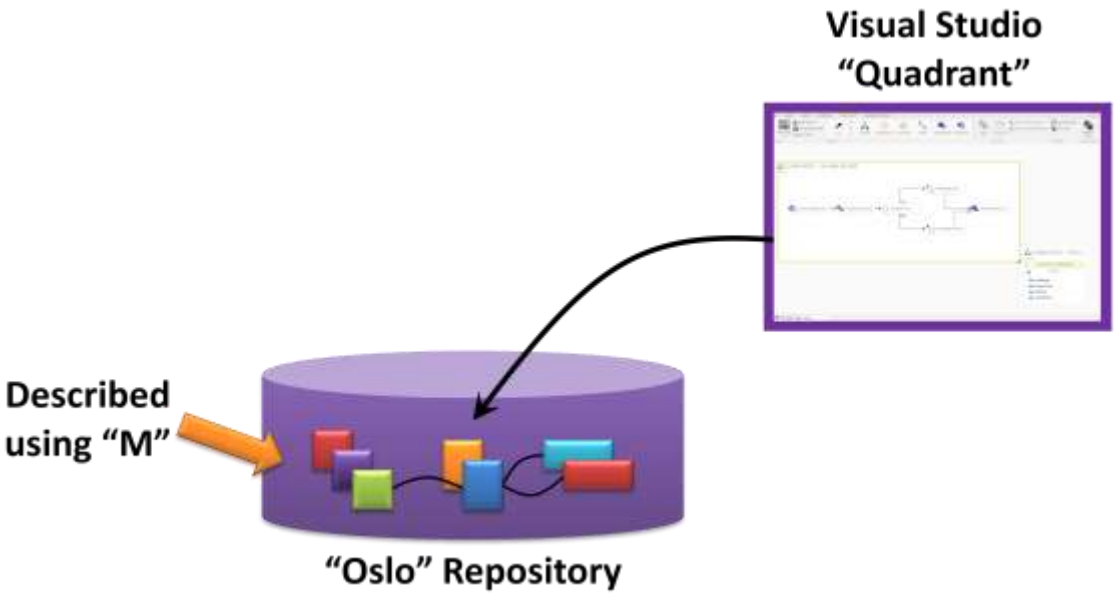


*Figure 4: "Quadrant" is an editor for working with "M"-defined information in the "Oslo" repository.*

The "Oslo" repository can store all kinds of information: WF workflows, models of applications that use those workflows, descriptions of the business processes those applications are part of, and more. And as Figure 4 suggests, the repository stores more than just raw information: It also stores relationships among this information. For example, a business process can be associated with an application it depends on, which in turn can be linked with the WF workflows it contains.

A primary goal of "Oslo" is to make models a fundamental part of how applications are created, deployed, and managed. In "Oslo", a model is an abstract representation of something, such as a business process, an application, or a workflow. (Don't confuse this use of "model" with how the term is used in other contexts, such as the Unified Modeling Language—it's not quite the same idea.) Rather than limiting models to descriptive diagrams used solely during the design process, "Oslo" can allow a model to be part of the application itself. For example, a WF workflow can be created using "Quadrant" and stored in the repository. This workflow is a model—it's in the repository—but it's also the actual logic of the workflow. Changing the model means changing the workflow itself, which means that the model and this part of the application can't get out of sync. The "Oslo" repository can hold models for more than applications, of course, and parts of an application might still reside outside of the repository. Still, the idea of moving models from just *describing* an application to actually *being* the application is fundamental to "Oslo".

Another goal of "Oslo" is to make information in the repository available to all kinds of tools for a variety of purposes. "Quadrant" provides one example of a tool that can be used with the repository. This general-purpose editor lets users create, read, update, and delete information in the repository, giving them a graphical interface to its contents. And since applications can be stored in the repository, "Quadrant" can also be useful for creating and modifying some kinds of application logic. The next section looks at how this idea fits into the larger context of this new technology wave.

## Combining Technologies: A Scenario

Even though each of the technologies described in this overview can be used on its own, getting a feel for the vision that underlies them requires seeing how they can work together. The scenario in Figure 5 illustrates one possibility.

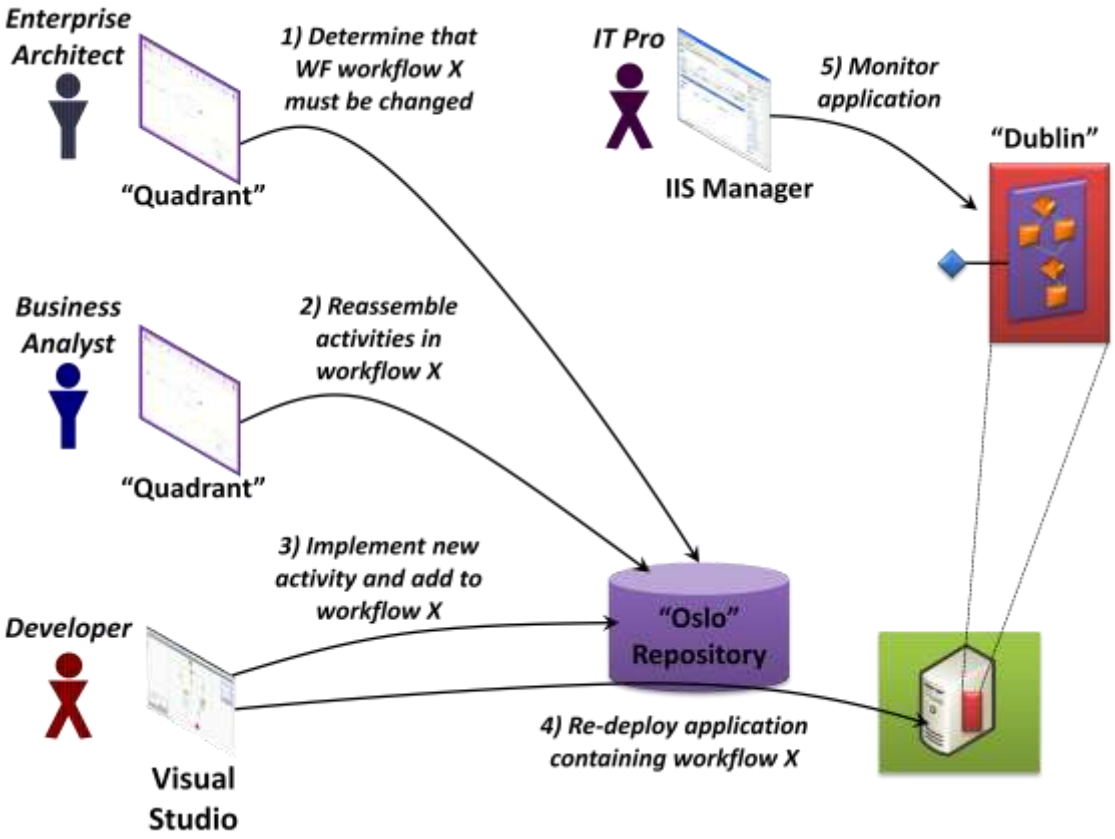


*Figure 5: An application can use workflows, services, and models together, combining WF 4.0, "Dublin", and "Oslo".*

Suppose an organization decides that one of its business processes can be improved by changing some existing steps in that process and adding a new one. Doing this requires modifying an application that supports the business process, which means that people in several different roles must work together. Figure 5 shows how the three technologies described here might help do this.

To get started, an enterprise architect uses "Quadrant" to figure out which application must be modified. The "Oslo" repository can contain descriptions of business processes, applications, and WF workflows these applications use, together with relationships among them. By navigating through these relationships, the architect might begin with the business process, then find the supporting application and the specific WF workflow, here called *workflow X*, within the application that must be changed (step 1).

Once the right workflow has been identified, a technically oriented business analyst who understands this part of the business process can get involved. Rather than referring to documentation, then telling a developer what to do, the analyst might work directly with the activities in workflow X using "Quadrant". In this case, suppose the analyst deletes one activity, changes the order of two others, and adds a new activity that's already present in the repository (step 2).

Reassembling what already exists is important, but changing business processes often requires writing new code. In this example, suppose that an entirely new activity must be added to workflow X to carry out the new step in the process. The business analyst can't do this, so he passes the task on to a developer. Working with Visual Studio, the developer begins with the same instance of workflow X that the business analyst has just modified. Because Visual Studio and "Quadrant" can both work with the same information, people in different roles can use it to work together. The developer can now implement the new activity and add it to the workflow (step 3).

Next, the developer re-deploys the modified business logic (step 4). Once again, this might be done via the repository, which can help deploy the application to "Dublin". After the modified application begins running, an IT pro can monitor its behavior using IIS Manager (step 6).

Making this kind of business process change is a fundamental part of what IT departments do. Making that change simpler, faster, and less expensive is a primary goal of this family of related technologies.

## A Closer Look

Having a big-picture understanding of what's coming is useful, but it's also useful to know a bit more about the three technologies in this initiative. This section looks at each of them in a little more detail.

## Windows Workflow Foundation 4.0

First released in 2006, WF is now used in Windows SharePoint Services and other products from Microsoft and other vendors. It's also been adopted by end user organizations to create their own applications. As this section shows, Microsoft's goal with WF 4.0 is to make this technology more attractive still to a broad population of developers.

### Using Workflows in Applications

If you really want to, pretty much anything can be implemented as a workflow. Still, the most common example of a WF-based application is one that implements some kind of long-running process. Because WF supports things such as storing a workflow's state persistently, then reloading it when the application resumes sometime later, it's a good match for software that does some work, shuts down for a while, then picks up where it left off.

One common example of this is applications that interact with people. Think of a document approval process, for instance, where an approver might not respond for an hour or a day or a week. The application needs to wait for her response, then continue with the next step in the process. Implementing this kind of behavior from scratch isn't simple, and so using WF makes good sense in cases like this.

The idea of assembling activities into workflows can also be useful in other situations. Because workflows can be created graphically, for instance, modifying existing applications or creating new ones can be easier than writing everything in code. Even non-developers can potentially get in on the act, as in the scenario described earlier. Being able to view a workflow's activities directly can also help in understanding its logic.

It's common today for WF workflows to expose and consume services. In the .NET world, using services means using WCF, and so Microsoft sometimes uses the term *WCF workflow service* to refer to software built using this combination. Figure 6 illustrates the relationship between workflows and services.

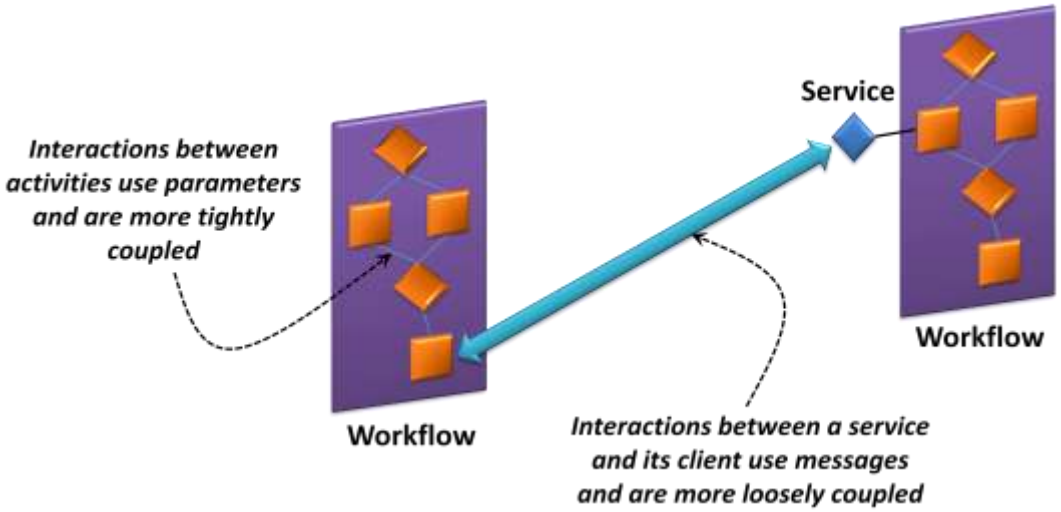


*Figure 6: Workflow activities implement behavior, while services provide communication.*

As the diagram shows, activities in a workflow interact via parameters. This is a quite tightly coupled relationship, and it's exactly what's needed in some cases. Alternatively, when a workflow must communicate with external software (implemented as a workflow or in some other way), this kind of close relationship isn't possible (or even desirable). Instead, the paradigm provided by services is likely to be a better fit. Service interactions rely on messages, and they allow a more loosely coupled relationship. For .NET applications that need both kinds of interaction, using WF workflows and WCF services together can make sense.

## Making Workflows Easier: What WF 4.0 Provides

While WF is used in a range of applications today, a large share of its initial adopters have been independent software vendors (ISVs), including Microsoft itself. To expand the technology's popularity among end users, WF must become easier to use. Toward this end, the next version of WF includes a number of new things:

- A new WF workflow designer with a more effective user experience. As shown earlier, this designer runs inside Visual Studio, and it allows developers to create workflows and activities graphically. (Using the designer isn't required, however—a developer is free to create workflows and activities entirely in code or in the XML-based language XAML.) This designer can also be re-hosted in other environments, such as products from ISVs, something that gets simpler in the WF 4.0 release.

- A larger set of built-in activities. WF's Base Activity Library was originally focused on control flow and communication. With WF 4.0, Microsoft will also provide activities for things such as accessing data and invoking PowerShell cmdlets. The intent is to make it easier to create useful applications by assembling existing activities into workflows.

- A new Flowchart workflow type. In its original incarnation, WF provided two built-in workflow types: Sequential, useful for relatively simple processes, and State Machine, which is more broadly applicable but harder to use. Adding Flowchart as a third built-in workflow type is meant to combine the advantages of Sequential and State Machine while still being straightforward to use.

- Changes in various other areas. For example, how data is handled within a workflow is simplified, making it easier to understand and work with. How a workflow interacts with its host process is also made easier, as is writing activities.

The next version of WF brings other improvements as well. Performance is much better, for example, and it’s now possible to invoke WF activities directly without explicitly creating a workflow. Overall, Microsoft’s intent is clear: They’d like to make every developer think about using WF whenever the problem they’re solving requires coordinating work over a period of time. In a world of distributed services, this workflow-based perspective is likely to become increasingly valuable.

## “Dublin”

Today, a .NET developer who needs a home for WCF-based business logic can use IIS, perhaps relying on WAS to activate services. While these technologies are certainly useful, offering something with more capability would be helpful for many developers. Accordingly, Microsoft has created “Dublin”.

### What “Dublin” Provides

The “Dublin” extensions to Windows Server build on the foundation provided by IIS and WAS, adding a number of new functions. As described earlier, “Dublin” can be useful both for long-running services, such as those built using WF, and for services that don’t use WF. Figure 7 illustrates this situation.

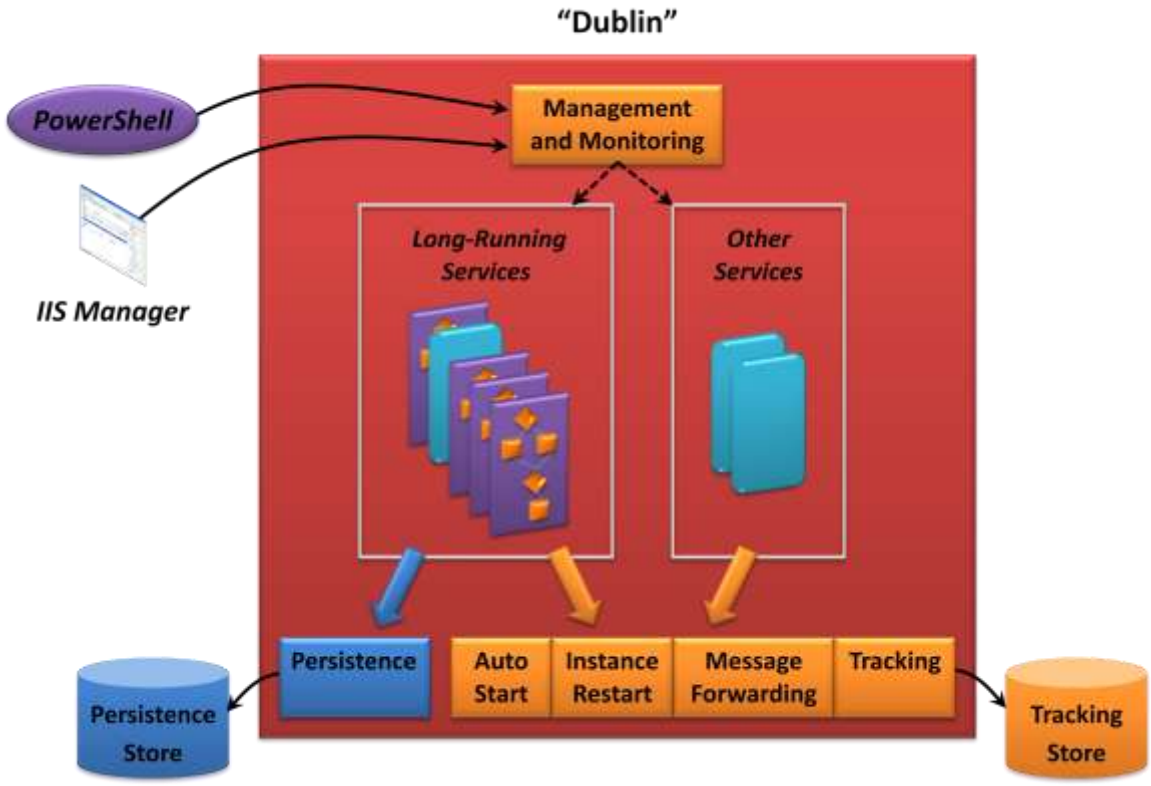


*Figure 7: "Dublin" provides supporting functions for long-running services, such as WF workflows, and others.*

As the figure suggests, "Dublin" is most attractive for long-running services built as WF workflows. A long-running service (which can also be implemented using just WCF—WF isn't required) relies on a persistence store, as shown in the figure. WF itself provides some support for persistence—it's built into the technology. This support relies on SQL Server, and it includes a schema for storing a workflow's persistent state along with a persistence provider for reading and writing that state. The persistence function in "Dublin" uses this same schema, adding a few extensions. It also provides extra support for making long-running services more scalable and reliable. For example, WF allows multiple machines to share the same persistence store. This lets a workflow whose state has been persisted be re-started on any of these machines. On this foundation, "Dublin" adds support for isolation of the data owned by services that are running in separate application pools.

Making persistence better is a good thing, but "Dublin" also offers other benefits to WF services. Recall that WF itself doesn't mandate any particular host, which has both positives and negatives. Since there's no required host, WF can be used in pretty much any context, from simple client applications to scalable servers. This flexibility is a good thing, but actually

creating a scalable server to host WF workflows isn't simple. "Dublin" provides this kind of server, relieving developers of the need to build one. To make it simpler to build WF applications that use "Dublin", Microsoft also adds a project template to Visual Studio for creating "Dublin"-hosted WF applications.

As Figure 7 shows, "Dublin" offers more supporting functions that are useful for both long-running services and others. Those services include:

- Auto start: By default, a WCF service starts running when the first message is received for this service. The "Dublin" auto start function allows one or more services to be automatically loaded as soon as the service is configured. This is useful for services that use non-activating channels, i.e., WCF communication mechanisms that don't automatically start a service, such as FTP or SMTP. Auto start can also improve response time for services that must carry out initialization tasks before handling their first request.

- Instance restart: An application might require that one or more service instances always be available. To provide this, the instance restart function monitors a heartbeat from all "Dublin" services. Any service that doesn't respond within the heartbeat expiration interval is automatically restarted. For a WF-based service, the workflow will resume from the last time its persistent state was saved.

- Message forwarding: This function provides content-based routing, letting a developer define a set of routing rules based on a message's contents. Once this is done, "Dublin" can route messages to different services based on what they contain.

- Tracking: It's often useful for a service to write events to a store to track what it's doing. The "Dublin" tracking service allows any service, whether or not it's built using WF, to do this. For WF-based services, "Dublin" builds on WF's existing support, adding tooling and other capabilities.

As Figure 7 shows, "Dublin" also provides management and monitoring functions for both long-running and other services. These can be accessed through IIS Manager or via PowerShell. (In fact, the IIS Manager functions are built on the "Dublin" PowerShell cmdlets.) An administrator can use either approach to deploy applications, then start, stop, resume, suspend, and query existing workflows.

Taken as a whole, what "Dublin" provides is useful for many WCF services. For WF developers in particular, "Dublin" fulfills one of their most frequent requests: It's a fully featured server, complete with tooling and infrastructure, for WF applications.

## Applying "Dublin": An Example

As with any technology, a good way to get a sense of how "Dublin" can be used is to walk through a representative example. The built-in "Dublin" support for creating scalable WF-based

applications is an important part of what this technology provides, and so this example illustrates how this might be used. Figure 8 shows the scenario.

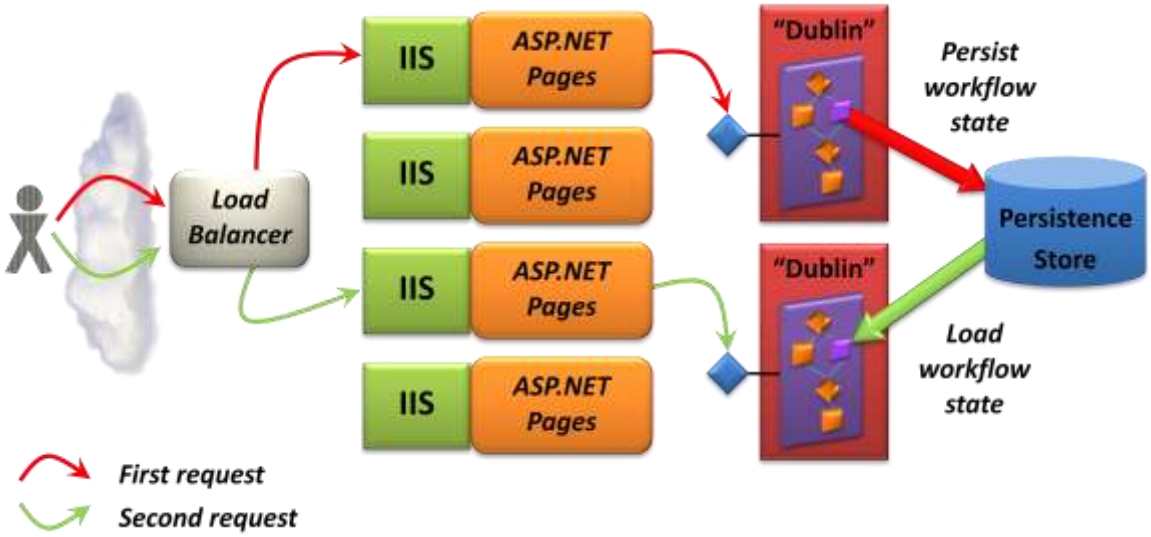


*Figure 8: Because multiple instances of "Dublin" can share a persistence store, a workflow can be restarted on any of those instances.*

Suppose that the application shown in the figure implements an on-line shopping process. Users access a group of replicated Web servers, each with the same set of ASP.NET pages. Requests from these users are spread across the servers by some load balancing technology. The ASP.NET code provides the user interaction, but it relies on a WF-based service hosted in "Dublin" to implement the shopping process using a workflow. As the figure shows, this WF service is also replicated, running on two different servers.

The first request from a particular user, shown in red in the figure, might get sent to the top IIS server, which in turn starts a WF workflow running in one instance of "Dublin". This workflow executes an activity for the first step in this process, such as creating a shopping cart for this user, then returns a result to the calling ASP.NET code. Once the workflow returns this result, the WF runtime automatically persists this workflow, storing its state—the user's shopping cart—in the "Dublin" persistence store.

Now suppose this user issues a second request, shown in green in the figure. This request might be to purchase another item or check out or do something else. Whatever its purpose, the request is load balanced to a different IIS server, which in turn invokes this application's workflow in a different instance of "Dublin". This isn't a problem, however. When "Dublin" receives this request, it can determine which workflow instance to restart (that is, it can figure out what persistent state it should load). "Dublin" then finds this state in the persistence store, reloads it, and sets the workflow running from the point at which it left off.

Because all of the replicated servers share a common persistence store, an instance of a workflow can be restarted on any of them. This lets the application scale horizontally—adding new "Dublin" instances on new machines lets it handle more users.

### "Dublin" and BizTalk Server

For anybody familiar with BizTalk Server, looking at "Dublin" might cause a slight sense of déjà vu. Supporting workflow-based logic, providing a monitoring and management infrastructure: These are things that BizTalk Server does today. What's the future of BizTalk Server in a "Dublin" world?

The key thing to understand is that "Dublin" doesn't directly target traditional BizTalk scenarios. For example, enterprise application integration and business-to-business connections via EDI will still use BizTalk Server. Similarly, bringing existing applications into the service-oriented world by exposing their functions and/or data through BizTalk Server will continue to make sense. While the reach of "Dublin" may grow over time, BizTalk Server remains important for connecting the Microsoft application platform to the diverse systems common in most enterprises.

If an organization needs an application container for WCF services, however, especially those implemented using WF, "Dublin" is a better choice—this is what it's designed for. And expect BizTalk Server's connection with "Dublin" to get stronger: Look for a future release after BizTalk Server 2009 to build on the "Dublin" infrastructure. Even though the functions of BizTalk Server and "Dublin" appear to overlap in some ways, each technology has a clear role to play.

## "Oslo"

The idea of modeling information, storing it in a repository, then using it in diverse ways has been around for some time. With "Oslo", Microsoft is jumping into this world with both feet. Yet the "Oslo" technologies aren't meant to stand on their own. Instead, the goal is to provide a foundation that can make model-driven applications and other uses of models central to how IT organizations work.

### Storing Models: The Repository

Viewed from high above, the "Oslo" repository is simple: It's a common store for diverse information about your environment and for relationships among that information. As is common in databases today, each piece of information the repository contains is an instance of some schema. Figure 9 illustrates this.

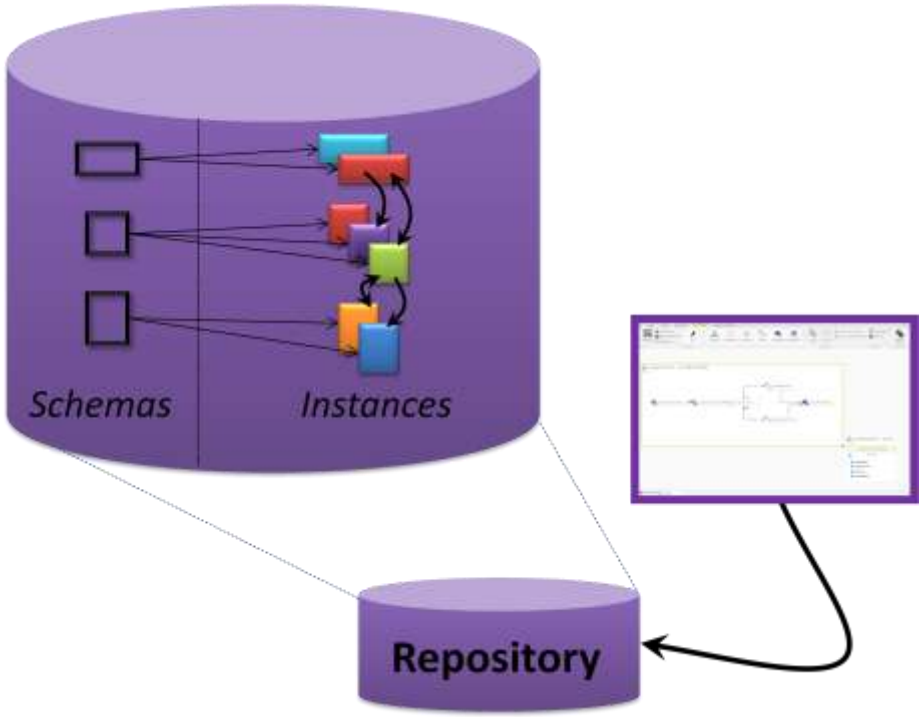


*Figure 9: The repository stores schemas, instances of those schemas, and relationships among those instances.*

ISVs and end users are free to define their own repository schemas, but Microsoft will also ship some of its own. Those pre-defined schemas include the following:

- *Process*: The steps in some process, e.g., a business process
- *Application*: A (possibly composite) application
- *Workflow*: A WF workflow
- *Activity*: A specific activity in a WF workflow
- *Service*: A service provided by an application, such as a WCF service
- *Environment*: A set of computers, software, etc. on which an application can be deployed

The “Oslo” repository is built on SQL Server, and so its schemas can be defined using standard SQL statements. (In fact, everything in the repository is accessible via SQL and standard database tools.) To make schema creation more approachable, however, “Oslo” also introduces an alternative way to define schemas. The next section looks at this new approach.

## Defining Models: "M"

Every piece of information stored in the "Oslo" repository must conform to some schema. Since the repository is built on SQL Server, it's possible to define these schemas using ordinary T-SQL statements. Yet doing this is reasonably complicated, especially for people who aren't database experts. Why not instead define a more approachable language for doing this, one focused entirely on this specific domain?

The "M" language family includes a capability called "MSchema" for doing this. A primary goal of "MSchema" is to make it easy for developers and architects—not just database administrators--to define new schemas for the "Oslo" repository. Toward this end, "MSchema" uses a syntax and a style that will be familiar to most developers.

For example, suppose you wish to define a type called ServiceContract for the "Oslo" repository. As the type's name suggests, its purpose is to store descriptions of the operations that a service exposes. In "MSchema", you might define that type like this:

```
module Contracts {
  import Messages;
  type ServiceContract {
    Name : Text;
    Operations : ServiceOperation*;
  }
  type ServiceOperation {
    Name : Text;
    Action : Text;
    Request : Messages.MessageContract?;
    Response : Messages.MessageContract?;
  }
}
```

The ServiceContract type is defined inside a module called Contracts. An "MSchema" module provides a container for types, and it can import types from and export types to other modules. In this example, the Contracts module begins with an import statement that makes available the types defined in another module named Messages.

The definition of ServiceContract appears next, and it's simple: A contract has a Name, which is just text, and some number of Operations, each of which has the type ServiceOperation. As in regular expressions, the asterisk that follows ServiceOperation indicates that this element can appear zero or more times.

Next is the definition for the ServiceOperation type. This definition says that each operation contains a name and an action, both of which are text strings. Each one can also contain a Request and/or a Response. Once again, "MSchema" uses the familiar regular expression syntax, where a trailing question mark indicates that the item can appear zero or one time. Request and Response are themselves defined using the type Messages.MessageContract, which is defined in the imported Messages module.

This simple example gives a bit of the flavor of this aspect of "M". "MSchema" is different from SQL, but the language's "Oslo" implementation nonetheless generates ordinary T-SQL statements. Although it's not shown here, "MSchema" also provides a way to specify relationships among schemas, define constraints, and more. And to make it easier to work with "MSchema", "Quadrant" includes a textual editor for creating and modifying schemas defined in this language.

## Working with Models: "Quadrant"

Storing schematized data in the repository is not an end in itself. This connected set of information has value only if people and other software use it. Over time, the intent is that many other components, from Microsoft and others, will rely on the repository. Still, having a general tool for creating, reading, updating, and deleting repository data has obvious value.

That tool is "Quadrant". By providing a consistent visual way to work with data in the repository, "Quadrant" allows a common approach to the diverse information it contains. Yet because the repository's contents can be so diverse, a generalized visual editor also needs to offer a broad set of options for viewing data. "Quadrant" does this by providing a standard set of viewers, each displaying repository information in a different way. This standard set allows viewing instances as diagrams, trees, property sheets, lists, and more. The creator of a schema can specify the default viewer to be used with instances of that schema, and other viewers can also be used. The same instance can even be examined using different viewers, each exposing different aspects of the underlying data.

It's worth reiterating that "Quadrant" and the other "Oslo" technologies provide a general platform for model-based software. This platform can be applied in many different domains. For example, an ISV might use "MSchema" to define a new set of schemas for some problem domain, such as an aspect of accounting or manufacturing. The ISV can then associate various "Quadrant" viewers with each schema, letting users work with instances of those schemas in different ways. An ISV is also free to create its own tools that use the repository—using "Quadrant" isn't the only option.

Over time, Microsoft plans to support access to the repository from other Microsoft tools. One good example of this is Visio, probably today's most popular choice for drawing business process diagrams. Figure 10 shows how Visio and "Quadrant" might be used together to create and modify the description of a business process.

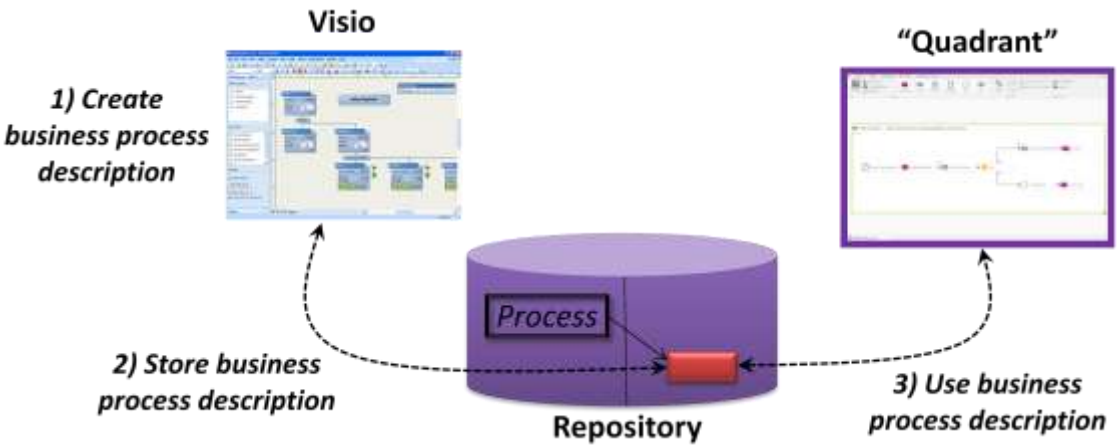


*Figure 10: A business process defined in Visio can potentially be used in "Quadrant" and vice-versa.*

In this example, the process starts with a business analyst creating a business process description in Visio (step 1). Once this is done, the analyst saves the process description in the repository (step 2). It's not saved as just a diagram, however. Instead, Visio transforms the picture this analyst has created into an instance of the Process schema. This schema can now be read by "Quadrant" (step 3). Just as important, this business process can be connected to other information in the repository, such as applications that support the process and WF workflows that implement those applications. Although Microsoft doesn't promise support for this scenario in the initial "Oslo" release, the clear intent is to allow using Visio and other tools with the repository.

## Applying "Oslo": Some Examples

"Oslo" can be used in a variety of ways. This section walks through a few examples of how these modeling technologies might be applied. The goal isn't to provide an exhaustive list of possible domains—there are too many of them—but rather to illustrate some of the ways in which Microsoft expects to see "Oslo" used.

Initially, the most important use of "Oslo" is likely to be working with WF-based applications. As described earlier, a technically oriented business analyst might use "Quadrant" to create and modify a WF workflow. Figure 11 shows how a simple flowchart workflow appears in this tool.

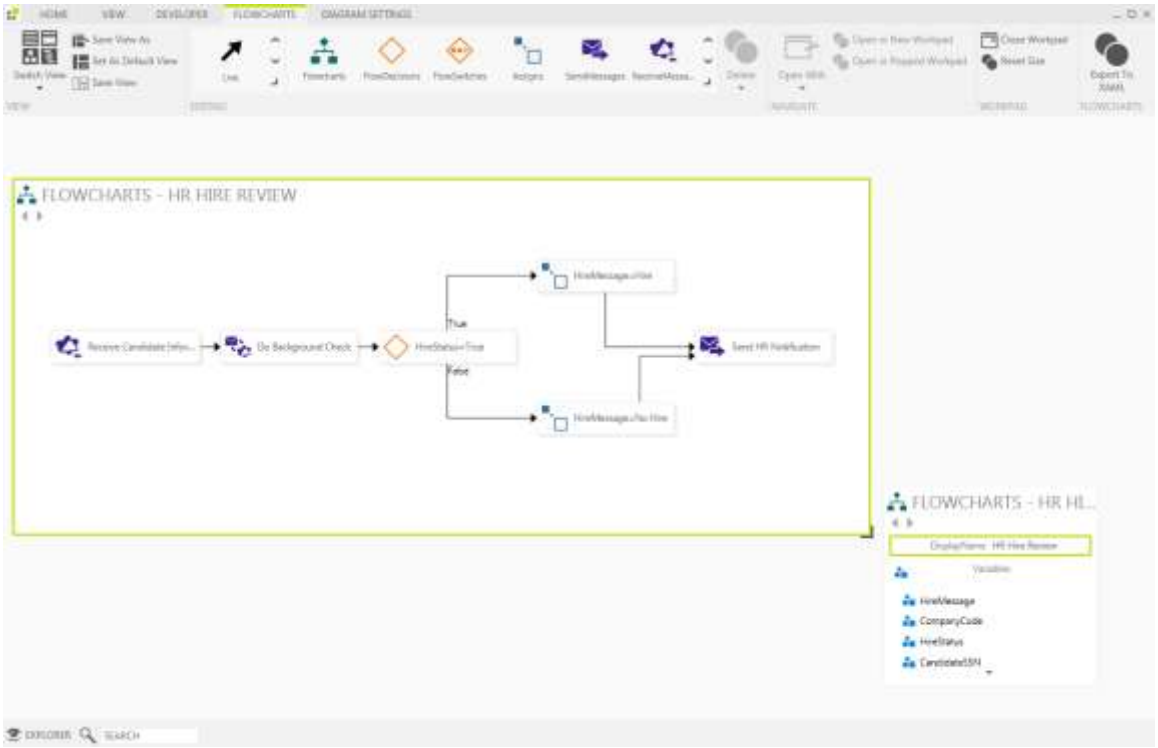


*Figure 11: "Quadrant" lets a user create and modify WF workflows.*

As the figure shows, "Quadrant" has a ribbon-style user interface, much like Microsoft Office 2007. How an instance in the repository looks is determined entirely by the viewers associated with that instance's schema—"Quadrant" itself doesn't define its appearance. Also, notice the icons in the ribbon: they're specific to workflows. "Quadrant" is context-sensitive, and so what's displayed to a user here depends on the schema of the data being edited.

In this example, a WF workflow is seen in two ways: the diagram viewer on the left, showing the activities in the workflow and the execution flow through them, and the property sheet viewer on the right, showing the variables used with this workflow. Each viewer allows working with the workflow in a different way. For instance, the user might rearrange the activities or add new ones from those in the repository with the diagram viewer, then modify its variables using the property sheet viewer. Since both are views onto the same underlying instance, changing a value in one viewer, such as the workflow's display name, will also cause it to change in the other.

Another possibility is for a technically oriented business analyst and a developer to work together on a single WF workflow, each using a different tool. The simple scenario described in the first part of this overview described this, but Figure 12 shows how it works in a little more detail.

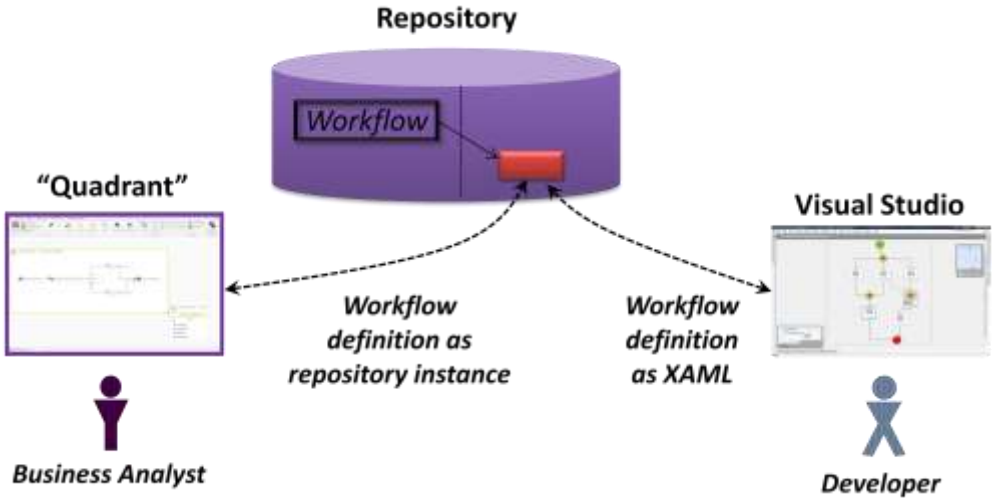


*Figure 12: A technically oriented business analyst and a developer might create or modify a workflow jointly, each using an appropriate tool.*

Using "Quadrant", the business analyst might assemble existing activities in the repository into a new workflow, then save it as an instance of the Workflow schema. A developer might then read this workflow instance into Visual Studio, as the figure shows. To let a developer use this workflow, it's converted into XAML, the XML-based language that WF uses to express workflow logic. The developer can now modify the workflow, perhaps adding details that are too technical for the business analyst to care about. Using Visual Studio, a developer can also create new activities in code, something that's not possible with "Quadrant".

The goal is to help business analysts and developers work together more effectively to create WF-based applications. This approach is analogous to how Expression Blend and Visual Studio help designers and developers work together on a user interface, each using a tool designed for his or her role. As with this earlier example of paired tools, the aim is to make creating and modifying WF-based applications easier.

Using "Oslo" to work with applications is far from the only possibility. More abstract things can also be usefully modeled. Suppose, for example, that an organization wished to describe one of its business processes in a more formal way. Some steps in this process might be done in software, while others are done by people. However the process is carried out, the whole thing can be represented by an instance of the "Oslo" Process schema. Figure 13 shows a simple example of how this might look in "Quadrant".

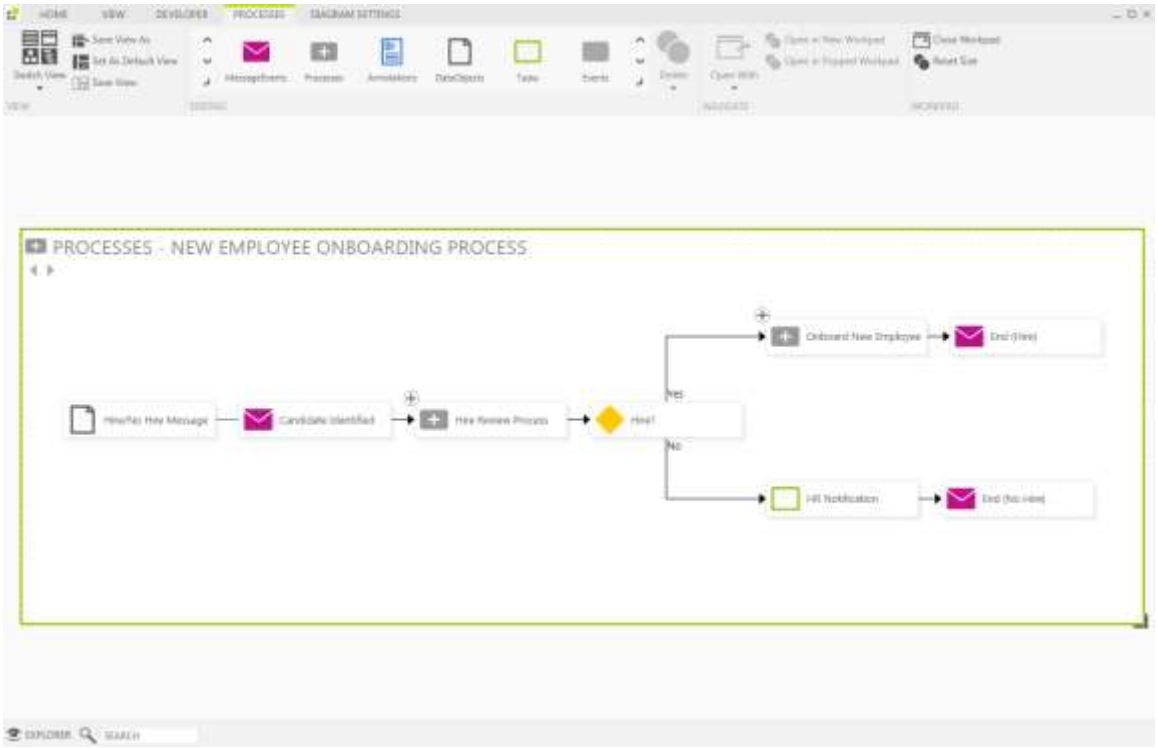


*Figure 13: "Quadrant" can display a business process using common conventions, such as representing each sub-process with a box containing a plus sign.*

Notice that the icons in the ribbon have changed—they're now showing appropriate tools for working with a business process. Unlike the previous example, however, this process doesn't produce anything that's executable. This is perfectly fine. While some models, such as those for applications, will produce artifacts that can be executed in "Dublin" or another runtime environment, others act purely as aids for understanding and documentation, as in this example.

Here's one more example of how "Oslo" might be used: Imagine that an organization models both WF-based applications and a business process that relies on them, then links the two together. If the applications are in turn linked to models of the computers on which they run, tracing problems in the business process back to their root causes can get easier.

Suppose, for instance, that a step in this process isn't meeting its service level agreement. Figuring out what's wrong can be a very complex undertaking, one that probably requires finding the right people and asking them the right questions. But if the business process, the applications it relies on, and the rest of the IT environment are all represented as linked instances of schemas—as "Oslo" models—tracing the problem could be significantly simpler.

Using a common tool such as "Quadrant", people in various roles could navigate through the repository to discover these connections, making it easier to find the root cause of the problem.

As these scenarios illustrate, "Oslo" is a general-purpose modeling platform that can be applied in a variety of domains. The examples just described—WF applications, business processes, and links among different parts of an IT environment—provide representative illustrations of how these technologies can be used. Going forward, expect to see Microsoft, independent software vendors, and others apply "Oslo" to these and a range of other problems.

## Defining DSLs with "M"

As described earlier, the "M" family includes "MSchema", a language designed for the particular domain of defining repository schemas. "MSchema" is an example of what's often referred to as a *domain-specific language (DSL)*. Other commonly used DSLs include SQL, a language for working with relational data, and regular expressions, a language for working with text.

DSLs can be useful, as these examples show. But creating new DSLs isn't simple, since it requires building a parser and more. Having a generalized way to do this would make it easier to define these special-purpose languages. To address this need, "Oslo" provides tools for creating DSLs.

The heart of this support is a part of "M" known as "MGrammar". "MGrammar" allows the creator of a DSL to define the syntax for his new language as a set of rules. Taken together, these rules define all of the legal statements allowed in this DSL. Using an "Oslo"-provided tool, the DSL's creator can then generate a parser for this new language. (This is similar to what's done by earlier tools such as YACC and ANTLR.)

Given a program that conforms to this set of rules, the parser can generate an *abstract syntax tree*, a data structure reflecting the program contents as a hierarchy. This tree can then be used to generate other information, such as C#, XML, SQL statements, or something else. "Oslo" doesn't define any standard way to generate code from an "MGrammar"-defined DSL, however—it's up to the organization defining the DSL how this is done. In fact, an "MGrammar"-defined DSL can be useful even when no code generation is required—it's an optional step.

"MGrammar" is used to define "MSchema", and it could be used to create other "Oslo"-related DSLs as well. Suppose, for example, that Microsoft wished to create a DSL for defining WF workflows. "MGrammar" would be a natural tool for doing this, perhaps generating instances of the Workflow schema in the "Oslo" repository. "MGrammar" can also be used independently, however—it's not bound to "Quadrant" or the repository.

"MGrammar" is focused on DSLs expressed in text, known as *textual* DSLs. To help create them, "Quadrant" includes an "M" editor to define grammars for new textual DSLs. Yet

"Quadrant" can also be seen as a tool for creating *visual* DSLs, special-purpose languages expressed graphically. Recall that the creator of a new schema can associate any viewers she likes with that schema. This can be thought of as defining a visual DSL for working with instances of this schema. In fact, given the right textual and visual DSLs, an "Oslo" user can potentially work with the same repository models using either style.

## Conclusion

The technologies described in this overview have many moving parts. The effort is large enough that Microsoft doesn't plan to release everything at once. Instead, expect the three main technologies described here to ship in chunks:

- WF 4.0 will ship with the .NET Framework 4.0 and Visual Studio 2010. These new releases of the Framework and Visual Studio will also contain other things, such as an updated version of WCF.

- "Dublin" will appear first as an independent download, then be included as part of Windows Server. It's likely to be available not long after the release of the .NET Framework 4.0 and Visual Studio 2010.

- The "Oslo" modeling technologies will be released together, including the repository, "M", and "Quadrant".

As these new technologies become available, organizations can adopt them in any combination they'd like. For example, a WF 4.0 application might be created using "Oslo" or solely with Visual Studio. Once it's written, the application can be deployed in a user-written host or in "Dublin". And, of course, this WF 4.0 application might be created using "Oslo", then deployed in "Dublin", as shown earlier. While all of these technologies can be used together, they remain independent from one another.

With WF 4.0, "Dublin", and "Oslo", Microsoft is laying the foundation for the next generation of distributed applications on Windows. In this view of the future, application logic is executed as workflows, exposed via services, and defined through models. The goals of "Oslo" also extend well beyond this, envisioning a world where model-based IT is the norm for many domains. Across all of these technologies, the goal remains the same: improving the lives of everybody who works in the Windows environment.

## About the Author

David Chappell is Principal of Chappell & Associates (www.davidchappell.com) in San Francisco, California. Through his speaking, writing, and consulting, he helps people around the world understand, use, and make better decisions about new technology.