

Windows Workflow Foundation

by Richard Blewett

The Windows Workflow Foundation (WF) was the last “pillar”, of what was then WinFX, to be announced. The covers were removed at PDC 2005 in Los Angeles and, in so doing, Microsoft revealed a very different approach to architecting applications. Most conference demos show the fall-through flowchart style of WF. This demonstrates one of the powerful features of workflow where the design of the processing and the processing itself are, basically, one and the same thing. However, the WF infrastructure is designed to run applications that are asynchronous by nature; in other words, applications that go into wait states for “external” stimuli to arrive. We have some examples of this style of programming in smart client and web applications which cover specific problem domains, but WF generalises the model to any type of application that has this requirement. It also goes beyond traditional event based models in that during the lifetime of the processing the code may run in different processes and, possibly, on different machines

What sort of applications need workflow?

Getting the message across about where workflow should be used is possibly the biggest challenge for Microsoft, when it comes to workflow, as it is an unfamiliar model of processing for many. Paul Andrew, from the workflow team wrote a blog post recently talking about some of the scenarios. You can read the post here <http://blogs.msdn.com/pandrew/archive/2007/02/01/what-to-use-windows-workflow-foundation-for.aspx>. The two most important ones, to me, are long running stateful processing and situations where the processing must be transparent. I will talk about how long running statefulness is enabled shortly; the transparency, however, is something often overlooked. The crucial thing is that a workflow can be defined completely declaratively and so without examining reams of code the functionality of a workflow can be deduced. In highly regulated and audited businesses this feature can be an enormous advantage over traditional development techniques.

In terms of enabling long running workflows a major requirement here is to be able to have processing that survives a machine restart. This requires that some form of durable persistence is available during processing. To understand how this works we need to look a little at the internals of WF.

Activities

Activities are the building blocks of a workflow. An activity models an individual step in a workflow and these are linked together to form a flow. Out of the box there are two models for driving a workflow: sequential, like the traditional flowchart where the sequence of events is reasonably well defined; state driven, that models a flow that has no inherent, predetermined order. Even though sequential workflow is the most commonly demonstrated form, it is state driven workflow that actually shows the new programming model more obviously. A state in a state driven workflow is an activity that can respond to external stimuli and that can then transition to another state as a result. This is an over-simplification but is the general pattern of state driven workflow.

Bookmarks

But how does a workflow enter a wait state? This is done by an activity setting up, what is internally known as, a bookmark. When the workflow engine executes an activity the activity responds by

saying “I have started execution but I have not finished, I’ll tell you when I am done” This means that the workflow engine can’t move the flow on until the activity asynchronously tells the engine its finished and so the workflow enters a wait state. During a wait state it’s possible for the engine to take the workflow, serialize its state and persist it to, say, a database and then unload it from memory – bringing it back into memory when the external stimuli arrives.

Reactivation

Now we have a problem. How on earth can the workflow receive an external stimulus if it’s not currently running in memory? The final piece of the puzzle is that all communication to the workflow is performed via in-memory queues. Before it bookmarks, the activity registers interest in receiving messages from a specific queue. When the external event takes place the data for that event must be posted on to the same specific queue. The process of doing this causes the workflow engine, if necessary, to drag the workflow out of persistent storage and back into memory. At this point, the activity then can pull the data off the queue, process it and then tell the workflow engine that it is finished – allowing the engine to move the processing on to the next activity in the flow.

Conclusion

Workflow is a powerful new tool in your toolbox. It can reduce the complexity of the code you write significantly. It, of course, is not a silver bullet and is not suitable for all situations. For example, I have heard it argued that you can replace the “if” statements in your code with workflow. This would be a huge processing overhead. But learn workflow and embrace it – many previously complex problems now have elegant solutions with the Workflow Foundation.