Egbert Imhof, Mark Onstwedder en Marcel de Vries

Workflows, maar dan dynamisch!

WORKFLOW FOUNDATION IN DE PRAKTIJK

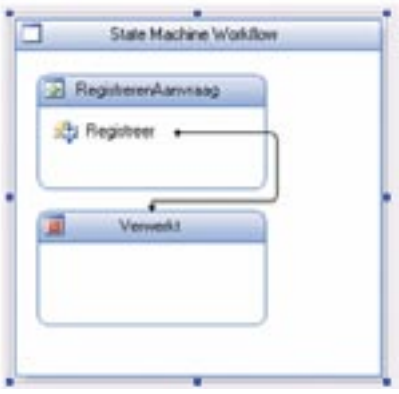
De termen ‘flexibiliteit’ en ‘onderhoudbaarheid’ hoor je vaak in het kader van softwareontwikkeling. Voor de automatisering van bedrijfsprocessen bij een organisatie is er onlangs gekozen voor Windows Workflow Foundation (WF). Natuurlijk kregen we te maken met diverse eisen betreffende flexibiliteit en onderhoudbaarheid van de oplossing. In dit artikel gaan we in op één aspect van de totaaloplossing om te voldoen aan de eisen van de klant: dynamisch op te bouwen workflows.

Een van de eigenschappen van WF is dat er een strikte scheiding is tussen het definiëren van een workflow met de Workflow Designer in Visual Studio en het moment waarop deze definitie geladen en uitgevoerd wordt door de workflow-engine. Er is sprake van een scheiding tussen de designtime- en runtime-omgeving. Vanuit de klant komt de vraag naar voren dat men de mogelijkheid wil hebben om, na het in productie nemen van het systeem, alsnog veranderingen door te kunnen voeren aan de stappen die in een workflow worden uitgevoerd. Daarbij moet wel worden vastgelegd welke stappen uiteindelijk runtime ooit zijn uitgevoerd ten behoeve van auditing. Om aan die eis te voldoen met WF als onderliggende technologie, zijn er twee mogelijke opties.
Optie één is het beschikbaar stellen van een omgeving waar de klant zelf de structuur van de workflow kan aanpassen. Daarbij maakt de klant feitelijk onder water een nieuwe definitie aan van een assembly en die asssembly moet vervolgens opnieuw worden uitgeleverd. Daarnaast moet de klant de beschikking krijgen over een client-applicatie waarin men de workflowdefinitie kan aanpassen. Een mogelijke oplossing daarvoor is om de designer-infrastructuur die geleverd wordt met .NET 3.0 zelf te hosten in een client-applicatie. Deze optie vereist dat er .NET 3.0 op de desktop beschikbaar is waar de workflow designers dan worden gehost in de eigen applicatie. Het beleid bij de klant staat dit echter voor een langere tijd nog niet toe, waardoor deze optie vrij snel als onbruikbaar moest worden bestempeld.
De tweede optie is een bepaald deel van de workflow als een losse definitie op te nemen in een database. Daarbij stelt de klant met een eenvoudige Windows-applicatie in welke activities voor een bepaalde workflow worden uitgevoerd en in welke volgorde. Het resultaat van de aanpassingen wordt in een database vastgelegd. Met deze oplossing voor het designtime-probleem moeten we vervolgens een manier vinden om een workflow uit te voeren die niet is gedefinieerd in een assembly of Xoml (een workflowdefinitie in XML-formaat) maar gedeeltelijk in een database. Daarbij is er voor gekozen om een hoofdworkflow te definiëren die altijd wordt uitgevoerd en die zelf een aantal substappen heeft dat dynamisch wordt ingevuld op basis van de database-instellingen.

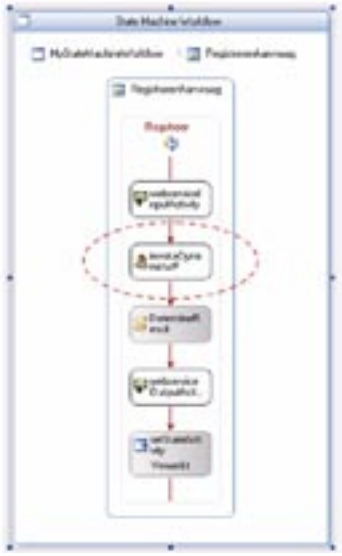
Afbeelding 1. Datamodel voor het opbouwen van een dynamische workflow

Datamodel

Voor het opbouwen van een dynamische workflow is informatie nodig over de activities die in de dynamische flow worden gebruikt en de volgorde waarin deze worden uitgevoerd. Hierbij moet worden opgemerkt dat voor de oplossing die is gerealiseerd, de dynamische workflow momenteel altijd een sequence is, die op basis van het resultaat van een activity wel of niet stopt met de verdere uitvoering van de sequence. Vandaar dat er in het datamodel een true en false action is gedefinieerd, waarbij wordt gespecificeerd of er op basis van de true of false verder wordt gegaan met de volgende activity. Een versimpelde weergave van het datamodel is weergegeven in afbeelding 1. De eerste entiteit in het model is de Activity waarin een workflow-activity wordt beschreven die binnen een dynamische workflow gebruikt wordt. Het belangrijkste hierbij is dat de typenaam en de assemblynaam van de te gebruiken activity hier opgegeven worden, waarbij uiteraard ook met versienummers van een specifieke assembly gewerkt kan worden. Op deze manier kunnen verschillende versies van een specifieke activity tegelijkertijd dynamisch geïnstantieerd worden om te gebruiken in verschillende contexten.
De tweede entiteit die van belang is, is de ActivityContext. Dit beschrijft de daadwerkelijke context waarbinnen één of meer activities in een specifieke volgorde aanwezig zijn. Deze context



Afbeelding 2. State-machineworkflow



Afbeelding 3. Registreer Aanvraag-workflow met aanroep van een dynamische workflow

Afbeelding 4. Relatie tussen onderdelen van de oplossing

kan zo complex gemaakt worden als gewenst, maar voor de eenvoud hebben we in dit artikel slechts één afhankelijkheid meegenomen: het AanvraagType. In principe kun je zo veel kolommen toevoegen als je wilt, zolang de gegevens worden meegegeven op het moment dat de workflow wordt aangemaakt. Vanuit de ActivityContext kunnen we nu alle activities opvragen die zijn ingesteld voor een specifiek AanvraagType, op volgorde gesorteerd. Op basis van deze lijst wordt vervolgens dynamisch de workflow opgebouwd.

Dynamische workflows

Zoals eerder is aangegeven, is er sprake van een hoofdworkflow, waarin op bepaalde plaatsen andere dynamisch aangemaakte workflows worden uitgevoerd. In het voorbeeld is gebruikgemaakt van een state-machineworkflow waarbinnen één of meer dynamische sequentiële workflows worden gestart. Op basis van het resultaat van deze subworkflow gaat de hoofdworkflow over naar een andere state. De state-machineworkflow is een statisch proces waar een aanvraag, die van buitenaf ingediend wordt, in zijn geheel doorheen wordt geloodst; zie afbeelding 2. We hebben met één parent-workflow te maken, die een gerelateerde dynamische child workflow afvuurt. Voor het aanmaken en besturen van de dynamische workflow hebben wij een InvokeDynamicWF-activity gebouwd; zie afbeelding 3.

InvokeDynamicWF-activity

De InvokeDynamicWF-activity bestaat uit drie onderdelen die nauw met elkaar samenwerken voor de opbouw en de uitvoering van de workflow en voor de communicatie van het eindresultaat. Het betreft hier de InvokeDynamicWF-activity, de DynamicWFService-runtime-service en de DynamicWF-workflow. In afbeelding 4 is de relatie tussen de diverse onderdelen schematisch weergegeven. De stappen die worden uitgevoerd zijn:

1. Aanmaken van een WorkflowQueue
2. Abonneren op OnQueueItemAvailable-event
3. Aanroepen service voor aanmaken en uitvoeren dynamische workflow (geef de event-queue door waarop we wachten voor resultaat)
4. Opbouwen en uitvoeren van de nieuwe workflow, op basis van de databasedefinitie
5. Signaleren dat de workflow gereed is en melden van het resultaat in de queue
6. Beschikbaar komen van de WorkflowQueue, uitlezen van het resultaat door InvokeDynamicWF-activity en aan de runtime doorgeven dat deze klaar is met executie.

Voor de communicatie tussen de activity en de service wordt gebruikgemaakt van een workflow-queue. Deze queues dienen als standaard mechanisme om activities te laten wachten op resultaten vanuit een service zonder daarbij de threads uit de threadpool (of ManualWorkflowScheduler) te blokkeren. De InvokeDynamicWF-activity creëert een WorkflowQueue, en geeft vervolgens de DynamicWFService de opdracht een nieuwe sequentiële workflow op te bouwen en het resultaat van de uitgevoerde workflow terug te communiceren in de aangemaakte workflow-queue. Vervolgens wacht de InvokeDynamicWF-activity tot het WorkflowQueueItem weer beschikbaar komt en de dynamische workflow dus klaar is. Tot slot zet de InvokeDynamicWF-activity het resultaat van de dynamische workflow in een property, ruimt de WorkflowQueue op en meldt de workflow-runtime dat zijn eigen state naar 'Closed' kan worden gezet (zie codevoorbeelden 1 en 2).

DynamicWFService-service

De DynamicWFService is verantwoordelijk voor het opbouwen en starten van een nieuwe sequentiële workflow van het type DynamicWF. Deze nieuwe workflow krijgt naast het door de InvokeDynamicWF aangemaakte workflow ID tevens de door de

```
protected override ActivityExecutionStatus Execute(
  ActivityExecutionContext executionContext)
{
  DynamicWFService wfService =
        executionContext.GetService<DynamicWFService>();

  WorkflowQueuingService wfqs =
        executionContext.GetService<WorkflowQueuingService>();

  WorkflowQueue wfq = wfqs.CreateWorkflowQueue(WorkflowInstanceId +
      "_CloseSync", false);

  wfq.QueueItemAvailable +=  new
       EventHandler<QueueEventArgs>(wfq_QueueItemAvailable);

  Dictionary<string, object> parameters = new Dictionary<string, object>();

  // RelatedWorkflowId en Aanvraag zijn dependency properties
  // van activity
  parameters.Add("Aanvraag", Aanvraag);
  RelatedWorkflowId = Guid.NewGuid();
  wfService.ExecuteWF(RelatedWorkflowId, parameters, wfq);

  return ActivityExecutionStatus.Executing;
}
```

Codevoorbeeld 1. Execute-methode van InvokeDynamicWF

```
// Het item in de queue is weer beschikbaar, ofwel de
// dynamische workflow is klaar.
void wfq_QueueItemAvailable(object sender, QueueEventArgs e)
{
   ActivityExecutionContext aec = sender as ActivityExecutionContext;

   if (aec == null) throw new Exception("Dit is fout!");

   WorkflowQueuingService wfqs = aec.GetService<WorkflowQueuingService>();
   WorkflowQueue wfq = wfqs.GetWorkflowQueue(e.QueueName);

   wfq.QueueItemAvailable -= new
        EventHandler<QueueEventArgs>(wfq_QueueItemAvailable);

   WFResult = (DynamicWFResult)wfq.Dequeue();
   wfqs.DeleteWorkflowQueue(e.QueueName);
   aec.CloseActivity();
}
```

Codevoorbeeld 2. wfq_QueueItemAvailable-methode van InvokeDynamicWF

```
public void ExecuteWF(Guid workflowId, Dictionary<string,object>
   parameters, WorkflowQueue closeQueue)
{
   WorkflowInstance wfinstance = Runtime.CreateWorkflow(
      typeof(DynamicWF), parameters, workflowId);

   waitList.Add(workflowId, closeQueue);
   wfinstance.Start();

   ManualWorkflowSchedulerService mwss =
      workflowRuntime.GetService<ManualWorkflowSchedulerService>();

   // Indien we gebruik maken van het thread donation model,
   // moeten we hier de thread doneren
   if (mwss != null)
      mwss.RunWorkflow(wfInstance.InstanceId);
}
```

Codevoorbeeld 3. ExecuteWF-methode van DynamicWFService

```
public void FireCompleteEvent(Guid workflowInstanceId,
   DynamicWFResult result)
{
   if (waitList.ContainsKey(workflowInstanceId))
      {
        if (waitList[workflowInstanceId] != null)
           waitList[workflowInstanceId].Enqueue(result);
      }
}
```

Codevoorbeeld 4. FireCompleteEvent-methode van DynamicWFService

aanroepende partij opgevoerde aanvraag mee in de parameters-Dictionary; zie codevoorbeeld 3. De DynamicWFService moet zoals gebruikelijk aan de workflow-runtime worden toegevoegd in code of via configuratie van de workflow-runtime. Daarna is deze voor de activity beschikbaar via de ActivityExecutionContext die wordt meegegeven bij de aanroep van de methode Execute door de runtime.

Je vraagt je misschien af waar hier de dynamische workflow wordt aangemaakt. Er is voor gekozen om het opbouwen van de dynamische workflow te laten uitvoeren door de workflow die wordt gestart. Dat is in het codevoorbeeld de workflow 'DynamicWF'. Een alternatieve aanpak zou zijn om zelf een nieuwe workflowdefinitie op te bouwen in de service en deze vervolgens naar Xoml om te zetten en deze Xoml-workflow te starten. Beide alternatieven zijn uitgewerkt en uitgebreid getest. Het blijkt dat beide oplossingen een gelijk resultaat opleveren ten aanzien van performance. Er zijn twee redenen waarom we niet voor de Xoml-optie hebben gekozen. De eerste reden is dat het nu ook mogelijk is een dynamische workflow direct uit te voeren zonder dat deze onderdeel is van een hoofdworkflow. Indien de workflow voldoende contextparameters meekrijgt tijdens creatie, zal deze altijd zijn werk kunnen uitvoeren en dus dynamisch een workflow kunnen opbouwen. De tweede reden is dat de huidige versie van windows-workflow een vervelende bug bevat bij de SqlTracking van pure Xoml-workflows. Er treedt namelijk na 32767 workflows een fout op in de tracking, waardoor alle workflows daarna in de terminated status terechtkomen (zie voor meer informatie http://blogs.infosupport.com/marcelv).
Uiteindelijk wordt nog het resultaat van een dynamische workflow teruggemeld. Hiertoe wordt de methode FireCompleteEvent van de DynamicWFService gebruikt; zie codevoorbeeld 4.

DynamicWF-activity

De DynamicWF-activity is een sequential workflow waarbinnen op basis van de meegegeven opstartparameters wordt bepaald welke activities in de sequence worden geplaatst. Het opbouwen van de

```
protected override ActivityExecutionStatus Execute(
   ActivityExecutionContext executionContext)
{
   // Creëer een WorkflowChanges object
   WorkflowChanges changes = new WorkflowChanges(this);

   DynamicWFService wfService = executionContext.GetService<DynamicWFService>();

   // Haal de activity gegevens op uit de database
   DbActivity[] activitiesToExecuteForContext =
      wfService.GetActivitiesForContext(Aanvraag.AanvraagType);

   // Creëer de dynamische workflow
   foreach (DbActivity dbActivity in activitiesToExecuteForContext)
   {
      // Creëer een activity
      BaseActivity activity = (BaseActivity)Activator.CreateInstance(
      dbActivity.ActivityTypeAssembly, dbActivity.ActivityType).Unwrap();

      // Voeg de activity toe aan de set met wijzigingen
      changes.TransientWorkflow.Activities.Add(activity);
   }

   // Wijzigingen toepassen
   ApplyWorkflowChanges(changes);

   // Als je geen custom scheduling logica nodig hebt,
   // maar alleen sequentiele executie, dan volstaat hier
   // de aanroep van base.execute(executionContext)

   // Start sceduling van de eerste activity
   // gebasseerd op de gegevens uit de database
   BaseActivity firstChild = GetNextChildToExecute();
   if (firstChild != null)
   {
      firstChild.Closed += new
        EventHandler<ActivityExecutionStatusChangedEventArgs>(ContinueAt);
      // Schedule child voor executie en bepaal de volgende activity
      executionContext.ExecuteActivity(firstChild);
      return ActivityExecutionStatus.Executing;
   }
   else
      return ActivityExecutionStatus.Closed;
}
```

Codevoorbeeld 5. Execute van de DynamicWF-activity

```
void ContinueAt(object sender,
  ActivityExecutionStatusChangedEventArgs e)
{
  ActivityExecutionContext aec = sender as ActivityExecutionContext;

  if (aec == null) throw new Exception("Dit is fout!");

  // Resultaat bepalen en zetten
  Action previousAction = (BaseActivity)e.Activity).Result;
  SetResult(previousAction, (BaseActivity)e.Activity);

  // unbind event handler
  e.Activity.Closed -= new
    EventHandler<ActivityExecutionStatusChangedEventArgs>(ContinueAt);

  // volgende child zoeken als we nog executing zijn
  if (aec.Activity.ExecutionStatus == ActivityExecutionStatus.Executing)
  {
    // als we door mogen gaan
    if (previousAction == Action.Continue)
    {
      BaseActivity nextChild = GetNextChildToExecute();
      if (nextChild != null)
      {
        nextChild.Closed +=
          new EventHandler<ActivityExecutionStatusChangedEventArgs>(
            ContinueAt);
        // schedule child voor executie en bepaal
        // de volgende activity
        aec.ExecuteActivity(nextChild);
        return; // doorgaan met executie
      }
    }
  }
  aec.CloseActivity();
}
```

Codevoorbeeld 6. ContinueAt-methode van de DynamicWF-activity

dynamische workflow wordt gedaan met behulp van de WorkflowChanges-class. De toe te voegen activities worden aan de eerder besproken DynamicWFService opgevraagd. Alle activities die in een dynamische flow worden opgenomen, erven van een BaseActivity. Deze BaseActivity heeft een vaste set aan properties die het resultaat van de executie teruggeven. Door de overerving van de base-class wordt ervoor gezorgd dat alle activities die geladen worden altijd dezelfde properties hebben waarop kan worden gecontroleerd of de executie wel of niet moet doorgaan. Wanneer de workflow is opgebouwd, wordt de eerste activity bepaald en uitgevoerd. Wanneer deze activity zich gereed meldt, wordt de volgende activity uitgevoerd, totdat de laatste klaar is; zie codevoorbeelden 5 en 6. Ten slotte meldt de DynamicWF zichzelf gereed (zie codevoorbeeld 7), zodat de InvokeDynamicWF verder gaat op basis van het resultaat van de dynamische workflow. De reden dat we zelf extra logica voor de execute hebben geschreven, is dat we op basis van het resultaat van één activity willen bepalen of we verder moeten met de volgende activity (true en false actions uit de database). Indien je zelf geen extra criteria nodig hebt om te bepalen of een child-activity moet worden uitgevoerd, kun je na de aanroep van ApplyChanges() gewoon de Execute van de base-class aanroepen die het verdere werk doet. De logica voor het bepalen welke volgende child wordt uitgevoerd, wordt verder buiten beschouwing gelaten omdat dit zeer oplossingspecifiek is.

WorkflowChanges-class

Windows Workflow Foundation biedt via de WorkflowChanges-class de mogelijkheid at runtime veranderingen door te voeren aan een lopende instantie van een workflow. Een WorkflowChanges-object is een representatie van een set van wijzigingen op de lopende workflowinstantie. Wanneer je een WorkflowChanges-object verkrijgt, krijg je in werkelijkheid een kopie van de lopende workflowinstantie. Door het aanroepen van de ApplyWorkflowChanges-methode op de root-activity (in dit geval een SequentialWorkflowActivity), waarbij het WorkflowChanges-object als parameter wordt meegegeven, worden de wijzigingen aan de workflowinstantie doorgevoerd; zie codevoorbeeld 5. Het WorkflowChanges-object kent in de huidige versie van WF nog wel een belangrijke beperking. Om deze beperking te verduidelijken, is het noodzakelijk om te weten dat een workflow een boom met activities is die je samenstelt door een sequential workflow of state-machineworkflow te tekenen. Deze boom heeft de SequentialWorkflowActivity of StateMachineWorkflowActivity als root. De beperking bij dynamic changes ligt in het feit dat je altijd alleen aanpassingen op de root van de workflow kunt doorvoeren. Dit is wel een beperkende factor als je een state-machineworkflow dynamisch wilt aanpassen. In zo'n workflow is iedere state namenlijk een leaf van de root. Hierdoor kun je alleen volledige states vervangen in de workflow en niet alleen maar een kleine aanpassing in bijvoorbeeld de afhandeling van een event binnen een EventDriven-activity. Voor een kleine aanpassing in de eventafhandeling van een state moet je hiervoor eerst de bestaande state verwijderen en exact de zelfde state opnieuw toevoegen, inclusief de aangepaste versie van de nieuwe EventDriven-flow. Dit maakt het uitvoeren van DynamicChanges redelijk bewerkelijk als het gaat om state-machineworkflows. Bij het gebruik van de WorkflowChanges-class wordt ook iedere verandering in de tracking-database bijhouden, indien in de runtime de trackingservices zijn geladen om de stappen te volgen die door een workflow doorlopen worden. Dit is zeer nuttig als na afloop van het uitvoeren van de workflow op een later tijdstip moet worden bepaald welke veranderingen zijn doorgevoerd.

Resultaten

Voor het inzichtelijk maken van de resultaten van een dynamische workflow kan heel goed gebruikgemaakt worden van een in de SDK meegeleverde tool, genaamd de Workflow Monitor, waarvan de sources door Microsoft worden meegeleverd, zodat deze gemakkelijk zelf uit te breiden is. Deze tool werkt op basis van de informatie die wordt opgeslagen door de standaard SqlTrackingService. Met de Workflow Monitor is grafisch inzichtelijk te maken hoe workflows uitgevoerd zijn/worden. De Workflow Monitor hebben wij aangepast om de relatie tussen de state-machineworkflow (hoofdworkflow) en de dynamische sequentiële workflow weer te geven. In afbeelding 5 is het eindresultaat van zo'n dynamische workflow te zien. De drie weergegeven activities zijn (uiteraard) in de database geconfigureerd.

InvokeDynamicWF vs. InvokeWorkflow-activity

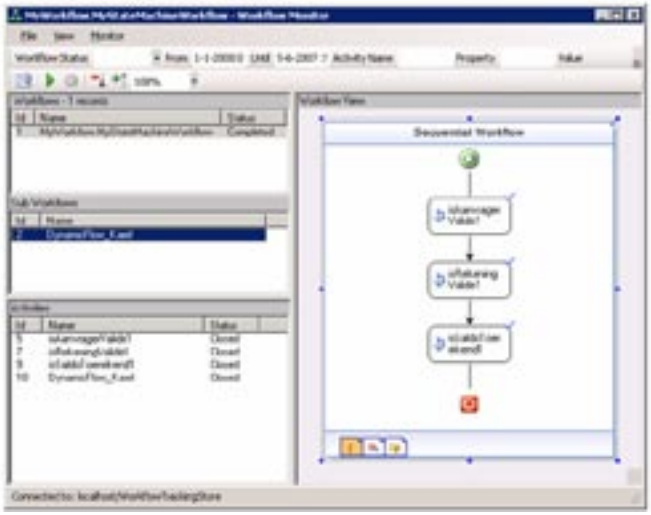
Indien je zelf al het een en ander gedaan hebt met Windows Workflow Foundation kun je je misschien afvragen waarom niet gebruik-gemaakt is van de InvokeWorkflow-functionaliteit die standaard out-of-the-box door Microsoft wordt geleverd. De reden hiervoor is dat de InvokeWorkflow-activity geen standaard mogelijkheid biedt om te wachten op het resultaat van de subflow. Dit

```
protected override void OnClosed(IServiceProvider provider)
{
  DynamicWFService wfService =
    (DynamicWFService)provider.GetService(typeof(DynamicWFService));

  // Signaleer aan de service dat de dynamische workflow
  // klaar is met executie
  wfService.FireCompleteEvent(WorkflowInstanceId,Result);

  base.OnClosed(provider);
}
```

Codevoorbeeld 7. OnClosed-methode van de DynamicWF-activity



Afbeelding 5. Resultaat van een dynamisch opgebouwde workflow

lijkt wel het geval te zijn als je de workflow host binnen bijvoorbeeld ASP.NET, maar dit komt omdat daar de ManualWorkflowSchedulerService wordt gebruikt die er voor zorgt dat de gehele workflow altijd sequentieel wordt uitgevoerd. Als een andere scheduler wordt toegepast, is de InvokeWorkflow plotseling asynchroon van de rest van de workflow en moet je zelf een andere manier bouwen om de correlatie met een subflow te coördineren. Feitelijk is de zelf gemaakte InvokeDynamicWF-activity niet meer dan een uitbreiding van de InvokeWorkflow-activity die de correlatie regelt tussen de hoofdworkflow en de subworkflow. We hadden dit ook graag gerealiseerd als een uitbereiding op de InvokeWorkflow-activity, maar deze is door Microsoft als 'sealed' gedefinieerd waardoor dit niet mogelijk is.

Waardevolle uitbreiding

Gezien onze ervaringen met Windows Workflow Foundation kunnen we niet anders concluderen dan dat dit een waardevolle uitbreiding is van het .Net Framework. Het biedt niet alleen standaard al heel veel functionaliteit om zowel state-machineworkflows als sequentiële workflows uit te voeren, maar het biedt ook heel veel mogelijkheden om zelf uitbreidingen op deze standaard functionaliteit toe te voegen. Kortom: een belangrijk instrument voor de toekomst.

Egbert Imhof is senior consultant bij Capgemini (www.nl.capgemini.com). Egbert heeft veel ervaring in het ontwikkelen van enterprise-applicaties op verschillende soorten platformen. De laatste jaren focust hij zich met name op het .Net-platform. Hij is te bereiken via egbert.imhof@capgemini.com.

Mark Onstwedder is senior consultant bij Capgemini (www.nl.capgemini.com). Hij houdt zich graag bezig met de nieuwste Microsoft-technologieën. Mark is te bereiken via mark.onstwedder@capgemini.com.

Marcel de Vries is IT-Architect bij Info Support voor de Businessunit Finance (www.infosupport.com). Marcel heeft al vele jaren ervaring opgedaan met het .NET-platform bij het bouwen van Enterprise administratieve applicaties voor grote bedrijven in Nederland. Naast het schrijven van artikelen is hij een veelgevraagde spreker op seminars en conferenties waaronder Microsoft DevDays en SDC. Naast zijn werkzaamheden bij diverse klanten geeft Marcel trainingen bij Info Support. Verder is hij bij Info Support architect van de softwareontwikkelstraat Endeavour. Marcel is te bereiken via marcelv@infosupport.com.

Referenties

http://wf.netfx3.com/
http://blogs.infosupport.com/marcelv