Jan Pieter Sonnemans en Jim Teunissen

State Machine Workflow als de motor van het proces

VUURSTEUN MET BEHULP VAN .NET FRAMEWORK 3.0

Soms komen nieuwe ontwikkelingen net te laat. Dat klinkt wellicht een beetje negatief, maar je kunt het natuurlijk ook omdraaien: soms ben je je tijd iets vooruit. En weer een andere insteek is dat Microsoft goed onderkent wat zijn klanten nodig hebben. In ons geval geldt dat voor het onderwerp wat we hier bespreken: State Machine Workflow, een belangrijk onderdeel van Workflow Foundation (WF), dat is opgenomen in .NET Framework 3.0.

Dit artikel is gebaseerd op een applicatie die we ontwikkeld hebben voor Defensie. We zijn daar twee jaar geleden mee begonnen, nog voordat het .NET Framework 3.0 gereleased was. Veel zaken die we bij Defensie zelf hebben ontwikkeld, kunnen nu eenvoudiger gerealiseerd worden met behulp van het .NET Framework 3.0. De technieken waar we gebruik van hadden kunnen maken zijn Windows Communication Foundation (WCF), en WorkFlow (WF). In dit artikel tonen we het principe van een State Machine Workflow, waarbij we laten zien hoe dit onderdeel kan worden ingezet aan de hand van een werkelijke applicatie. Deze workflow wordt gehost in een service die is ontwikkeld met behulp van WCF. Het voorbeeld is gebaseerd op een applicatie die nu gebruikt wordt door de landmacht en ook operationeel ingezet wordt. Het toont aan dat .NET-technologie toegepast kan worden in meer dan alleen kantoorapplicaties.

State Machine Workflow

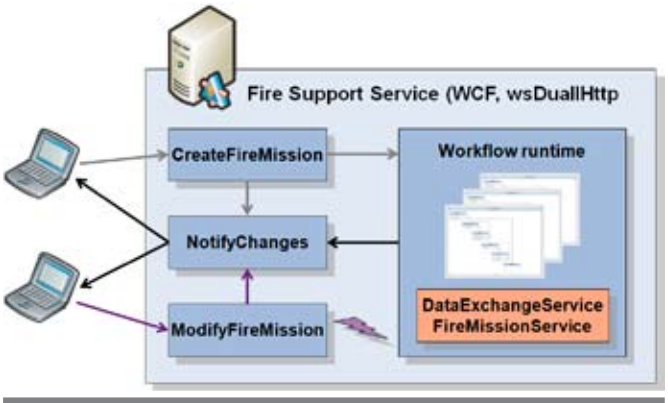
In het .NET magazine #12 (maart 2006) is een prima artikel geplaatst van Anko Duizer waarin hij duidelijk uitlegt wat State Machine Workflow inhoudt. Dit gaan we hier dus niet herhalen. Het hier beschreven proces dat door de applicatie wordt ondersteund, is een versimpelde versie van het werkelijke proces, maar demonstreert dat State Machine Workflow een prima techniek is voor deze situatie. We hebben het over het proces van vuursteun door de artillerie. Het gaat hier om houwitsers die ruim veertig kilometer ver kunnen schieten. Ze hebben dus geen zicht op het doel. Een doel wordt namelijk bepaald door een *Forward Observer (FO)* die de vuuraanvraag inbrengt in het systeem. Een zogenaamde *FireMission* wordt daarmee gecreëerd. Op een doel wordt niet zomaar gevuurd. Daar moet eerst autorisatie voor verleend worden door functionarissen die een beter beeld hebben van het slagveld, deze rol heet *FireSupport Coordination Center*. Nadat er toestemming voor is verleend, komt de vuuraanvraag bij een *Fire Direction Center*. Daar wordt het werk verdeeld onder de beschikbare *Guns*. Zij richten en melden zich gereed waarna een opdracht tot vuren wordt gegeven. Nadat de granaten zijn gevallen, bepaalt de FO of er meer granaten nodig zijn of dat het doel voldoende bestreden is en de *Firemission* beëindigd kan worden.
Bovenstaand proces, gemodelleerd in de State Machine Workflow-designer van Visual Studio, wordt getoond in afbeelding 1. Een State Machine Workflow is opgebouwd uit de elementen states en transities. Een state geeft de toestand van een workflow aan op een bepaald moment, de transities zijn de overgangen die mogelijk zijn van een state naar een andere state. In ons voorbeeld betekent dit dat zodra een Forward Observer een 'vuurtje' wil op een bepaald doel, dat hij dat met de applicatie inbrengt. Dit resulteert in de State Machine Workflow in een state CallForFire. In het model kun je zien dat er twee transities mogelijk zijn op die state: Approved en Rejected. De volgende schakel in het vuursteunproces beoordeelt namelijk de vuuraanvraag en keurt hem af dan wel goed. In het laatste geval gaat het proces verder en wijzigt de state van de workflow naar EngagementOrder.
Het reguliere proces van een vuursteunaanvraag is het doorlopen van iedere stap uit afbeelding 1 van boven naar beneden. Daar zijn drie uitzonderingen in opgenomen:

- Een CallForFire kan geweigerd worden (Rejected) en daarmee is het proces beëindigd.
- Een FireOrder kan niet opgepakt worden door een gun (Cant-Comply), omdat er bijvoorbeeld een technische storing is of omdat men tijdelijk door de munitie heen is. In dat geval kan een andere gun aangewezen worden om de missie uit te voeren.
- Het doel is niet voldoende vernietigd, door een 'Repeat' aan te vragen, kan er opnieuw op geschoten worden.

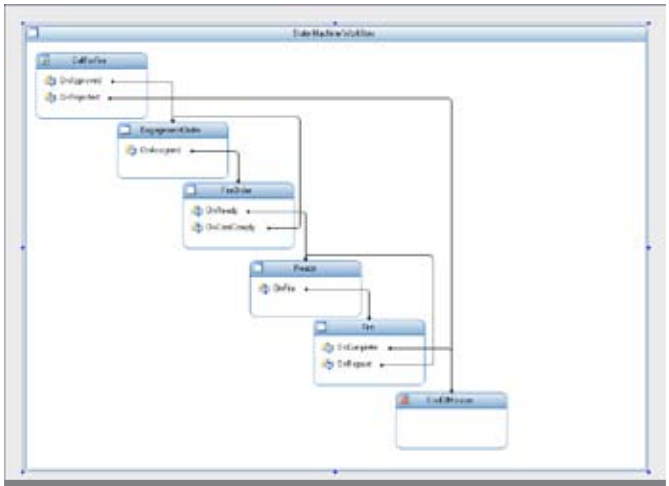
Samenvattend: het proces bestaat uit diverse stadia waar iedere keer specifieke acties uitgevoerd moeten worden, waarbij verschillende transities mogelijk zijn naar een vervolgstadium. Een typisch voorbeeld van State Machine Workflow.

Implementatie in .NET

De opzet en werking van de applicatie worden getoond in afbeelding 2. De workflow zoals gedefinieerd in afbeelding 1 reageert op events zoals het op Approved zetten van een CallForFire. De



Afbeelding 1. Het reguliere proces van een vuursteunaanvraag



Afbeelding 2. De opzet en werking van de applicatie

mogelijke events worden kenbaar gemaakt door een interface te implementeren die gemarkeerd is met het attribuut [ExternalDataExchange]. Voor iedere transitie naar een state is een event opgenomen. Codevoorbeeld 1 laat die interface en een gedeelte van de implementatie ervan zien.

Bij iedere state (behalve de EndState EndOfMission) is voor elke transitie een EventDriven Activity opgenomen, zie afbeelding 3 voor een voorbeeld daarvan. Een dergelijk EventDriven-activity is als het ware een 'normale' Sequential Workflow, de andere vorm van workflow die door .NET Framework 3.0 wordt ondersteund. Deze wordt getriggerd door een event, zoals de naam al impliceert. Iedere EventDriven-activity begint met een HandleExternalEvent-activity. Hierin wordt de workflow gekoppeld aan een event, dat uiteindelijk geraised wordt door het aanroepende proces. In ons geval worden de HandleExternalEvent activities en daarmee de workflow gekoppeld aan de events van de interface IFireMissionService. De laatste activity die in de EventDriven-activity is opgenomen is een SetState-activity. Hiermee wordt de nieuwe state gezet die het gevolg is van de transitie.

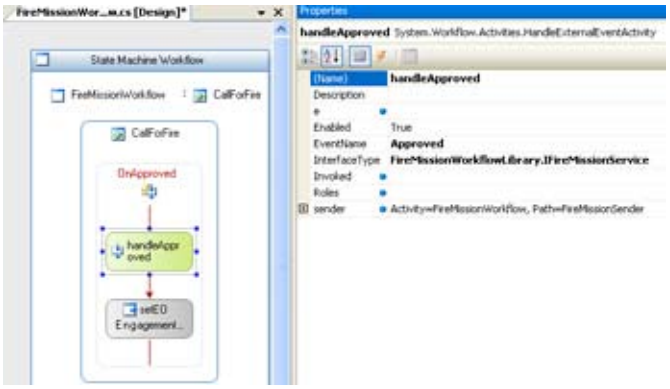
```
[ExternalDataExchange]
public interface IFireMissionService
{
   event EventHandler<FireMissionEventArgs> Created;
   event EventHandler<FireMissionEventArgs> Approved;
   event EventHandler<FireMissionEventArgs> Rejected;
   event EventHandler<FireMissionEventArgs> Assigned;
   event EventHandler<FireMissionEventArgs> Ready;
   event EventHandler<FireMissionEventArgs> CantComply;
   event EventHandler<FireMissionEventArgs> Fire;
   event EventHandler<FireMissionEventArgs> Repeat;
   event EventHandler<FireMissionEventArgs> Completed;
}

[Serializable]
public class FireMissionService : IFireMissionService
{
   public event EventHandler<FireMissionEventArgs> Created;
   public event EventHandler<FireMissionEventArgs> Approved;
   … etc. voor de overige events.

   public void OnCreated(FireMissionEventArgs eventArgs)
   {
      if (Created != null)
      {
         Created(this, eventArgs);
      }
   }

   ... etc. voor de overige events.
}
```

Codevoorbeeld 1. De interface en een gedeelte van de implementatie ervan



Afbeelding 3, Voor elke transitie is een EventDriven-activity opgenomen

Windows Communication Service

Wat duidelijk moet zijn, is dat het proces van een vuuraanvraag niet op één machine wordt afgehandeld. Er zijn minimaal vier verschillende functionarissen bij betrokken die op verschillende locaties met hetzelfde proces bezig zijn. Zij moeten worden genotificeerd als er van hen actie wordt verwacht en zij moeten de voortgang van het proces kunnen monitoren. Hierbij komt WCF om de hoek kijken. Er is een service nodig waar de applicaties zich op aanmelden en die de FireMissions en de wijzigingen ontvangt. Vervolgens laat de service de workflow-engine zijn werk doen en wordt het resultaat teruggestuurd naar alle betrokken partijen. Een eenvoudige aanroep met het direct teruggeven van het resultaat als returnvalue is hierbij niet voldoende. Het is namelijk niet alleen de leverancier van de informatie, maar juist *alle* client-applicaties die hiervan op de hoogte gesteld moeten worden. Daarvoor is een CallBack-mechanisme nodig om alle betrokken partijen te notificeren.

```
[ServiceContract(CallbackContract =
       typeof(IFireSupportCallback))]
public interface IFireSupportService
{
   [OperationContract(IsOneWay = true)]
   void Connect();

   [OperationContract(IsOneWay = true)]
   void Disconnect();

   [OperationContract(IsOneWay = true)]
   void CreateFireMission(
        string fireMissionId,
        string location);

   [OperationContract(IsOneWay = true)]
   void ModifyFireMission(
        string fireMissionId,
        string transition);

   [OperationContract]
   FireMission[] GetFireMissionList();
}

public interface IFireSupportCallback
{
   [OperationContract(IsOneWay = true)]
   void StateChanged(FireMission fireMission);
}
```

Codevoorbeeld 2. De interface IFireSupportService

Het contract

Codevoorbeeld 2 laat de interface IFireSupportService zien die het contract beschrijft van de service. De methodes Connect en Disconnect worden gebruikt om de verschillende clients te registreren die genotificeerd worden, wanneer de toestand van de workflow verandert. Met behulp van de methode CreateFireMission kan een nieuwe FireMission (en daarmee een nieuwe workflow) worden gecreëerd. Door middel van de ModifyFireMission wordt een actie op de FireMission door de client doorgegeven. Resteert nog de GetFireMissionList. Deze methode retourneert de lijst van FireMissions, zodat een client die zich later aansluit op de service ook een lijst van FireMissions ontvangt die up-to-date is. In het attribuut ServiceContract boven de interfacedefinitie is een CallbackContract opgenomen. Het verwijst naar de interface IFireSupportCallback. Alle wijzigingen in de FireMissions die plaatsvinden worden met behulp van de hierin opgenomen methode teruggemeld aan alle clients die zich hebben geregistreerd bij de service.

De binding

Omdat we met een callback-mechanisme werken, is er sprake van 'Two-Way-Communication'. Daarom hebben we twee opties voor de binding: NetTcp en wsDualHttp. NetNamedPipe-binding kan ook gebruikt worden in combinatie met callbacks, echter alleen voor 'Same-Machine-Communication' en dat is hier niet aan de orde.

De service

Omdat we alle data willen delen met de verschillende clients, werken we met een Singleton-service. Dit realiseren we door een ServiceBehavior aan de service toe te voegen, waarbij de InstanceContextMode op Single wordt gezet. Clients die zich aanmelden aan de service, worden vastgehouden om later via de callback genotificeerd te kunnen worden. Dat gebeurt door bij iedere geregistreerde client de methode StateChanged uit de callback-interface aan te roepen; zie codevoorbeeld 3. Bij de callback wordt gebruikgemaakt van een FireMission-object. Omdat dit object geserializeerd over de lijn naar de clients wordt verstuurd, is dit object gemarkeerd met het attribuut [DataContract] en de properties die voor een client nodig zijn met het attribuut [DataMember].

Hosting van workflow in de service

De workflow wordt gehost in de WCF-service. Daarmee creëren we één omgeving, waarmee meer clients communiceren en waarbij ze gezamenlijk niet alleen hun **data** maar ook **het proces** delen. De WCF-service moet daarvoor de WorkflowRuntime initiëren en starten. De workflow moet gaan reageren op gebeurtenissen die van buiten de workflow geïnitieerd worden. Dit wordt mogelijk gemaakt door aan de WorkflowRuntime een ExternalDataExchangeService te koppelen. De FireMissionService uit codevoorbeeld 1 bevat alle mogelijke events waar onze workflow op gaat reageren. Door de FireMissionService aan de ExternalDataExchangeService te koppelen, is communicatie tussen de WCF-service en de workflow mogelijk; zie codevoorbeeld 4. Voor de duidelijkheid: er is één WorkflowRuntime, voor iedere FireMission wordt telkens een nieuwe workflow gecreëerd.

```
private void NotifyStateChanged(FireMission fireMission)
{
  Action<IFireSupportCallback> action =
        delegate(IFireSupportCallback fireSupportCallback)
          {
            fireSupportCallback.StateChanged(fireMission);
          };

  _fireSupportCallbackList.ForEach(action);
}
```

Codevoorbeeld 3. Het aanroepen van de methode StateChanged uit de callback-interface

Het creëren van een FireMission is het startpunt van de workflow. In de methode CreateFireMission wordt dan ook een WorkflowInstance gecreëerd op de workflow-runtime. Omdat we gebruikmaken van State Machine Workflow moet er ook een StateMachineWorkflowInstance gecreëerd worden die vervolgens aan de WorkflowRuntime en de WorkflowInstance wordt gekoppeld. In ons voorbeeld wordt een FireMission-object gecreëerd dat de eigenschappen van de aangevraagde vuuraanvraag bevat. We nemen daarin ook de StateMachineWorkflowInstance op en wrappen de CurrentState en de PossibleTransitions. Om de statusveranderingen goed naar de clients te krijgen, hebben we een vreemde constructie moeten toepassen. We hebben een pauze (Thread.Sleep) moeten toevoegen aan de methode NotifyStateChanges; zie codevoorbeeld 3. Het blijkt namelijk dat StateChanges door de

```
private WorkflowRuntime _workflowRuntime;
private FireMissionService _fireMissionService;

public FireSupportService()
{
  _workflowRuntime = new WorkflowRuntime();

  ExternalDataExchangeService dataExchangeService =
        new ExternalDataExchangeService();
  _workflowRuntime.AddService(dataExchangeService);

  _fireMissionService = new FireMissionService();
  dataExchangeService.AddService(_fireMissionService);

  _workflowRuntime.StartRuntime();
}

public void CreateFireMission(
          string fireMissionId,
          string location)
{
  Type workflowType = typeof (FireMissionWorkflow);

  WorkflowInstance workflowInstance =
          _workflowRuntime.CreateWorkflow(workflowType);

  StateMachineWorkflowInstance stateMachineInstance =
          new StateMachineWorkflowInstance(
               _workflowRuntime, workflowInstance.InstanceId);

  workflowInstance.Start();

  FireMission fireMission = new FireMission(
          fireMissionId, location, stateMachineInstance);
  _fireMissionCollection.Add(fireMissionId, fireMission);

  NotifyStateChanged(fireMission);
}
public void ModifyFireMission(
          string fireMissionId, string transition)
{
  FireMission fireMission = _fireMissionCollection[fireMissionId];
  FireMissionEventArgs eventArgs =
          new FireMissionEventArgs(fireMission.InstanceId, fireMissionId);

  switch(transition.ToLower())
  {
    case "approved":
      _fireMissionService.OnApproved(eventArgs);
      break;

    ... etc. voor de overige transitions.
  }
  NotifyStateChanged(fireMission);
}
```

Codevoorbeeld 4. De FireMissionService wordt aan de ExternalDataExchangeService gekoppeld

Afbeelding 4. V.l.n.r.: Jan Pieter Sonnemans en Jim Teunissen

WorkflowEngine niet direct worden doorgevoerd. Er is ook geen event aanwezig dat afgaat als een status is gewijzigd. Deze workaround wordt door Microsoft zelf toegepast in haar voorbeelden (de constructie is rechtstreeks overgenomen uit hun StateMachineTrackingService-voorbeeld). Laten we hopen dat in een volgende versie van WF dit manco is opgelost.

De client

Met behulp van bovenstaande services kunnen verscheidene clients naar dezelfde data kijken en er veranderingen in aanbrengen. In de client worden afhankelijk van het aantal transities enkele knoppen getoond waar de transitie als tekst van de knop wordt weergegeven. Bij het indrukken van die knop wordt de transitie door middel van de methode ModifyFireMission doorgegeven aan de WCF-service die het juiste event raised, waarop de State Machine zijn nieuwe state bepaalt. Afhankelijk van hun rollen zullen voor een gebruiker bepaalde zaken wel of niet zijn toegestaan. Dit hebben we in dit artikel verder niet benoemd, omdat zoiets voor dit moment te ver gaat. Ook moet duidelijk zijn dat in een echte vuuraanvraag veel meer gegevens opgenomen worden en dat het hier geschetste proces erg versimpeld is.

Robuust regelen

Met dit voorbeeld hopen we dat we de kracht van de State Machine Workflow hebben aangetoond. Mocht het proces aangepast worden, dan hoeven de clients niet aangepast en opnieuw uitgerold te worden. Het centraal aanpassen van de workflow is dan voldoende. Verder is het gebruik van WCF zo eenvoudig dat het robuust regelen van de two-way communicatie op zich geen uitdaging meer is. En dat wil je ook. Bij dit artikel hebben we het voorbeeld compleet werkend opgeleverd. Je kunt het vinden op de site van .NET magazine.

Jim Teunissen is softwarearchitect en werkt bij Atos Origin BAS Microsoft Technologies. Hij is te bereiken op jim.teunissen@atosorigin.com.

Jan Pieter Sonnemans is softwarearchitect en werkt eveneens bij Atos Origin BAS Microsoft Technologies. Hij is te bereiken op jan-pieter.sonnemans@atosorigin.com.

Referenties:

Microsoft .NET Framework 3.0 Community (NetFx3): http://www.netfx3.com/

.NET Framework Developer Center: http://msdn2.microsoft.com/en-us/netframework/default.aspx

IDesign: .NET Design and Process Solutions: http://www.idesign.net/idesign/DesktopDefault.aspx?tabindex=5&tabid=11

Thinktecture blogs: http://blogs.thinktecture.com/

.NET Magazine #12, Artikel Anko Duizer over StateMachineWorkflow: http://www.microsoft.nl/netmagazine12