Marcel de Vries is werkzaam bij Info Support

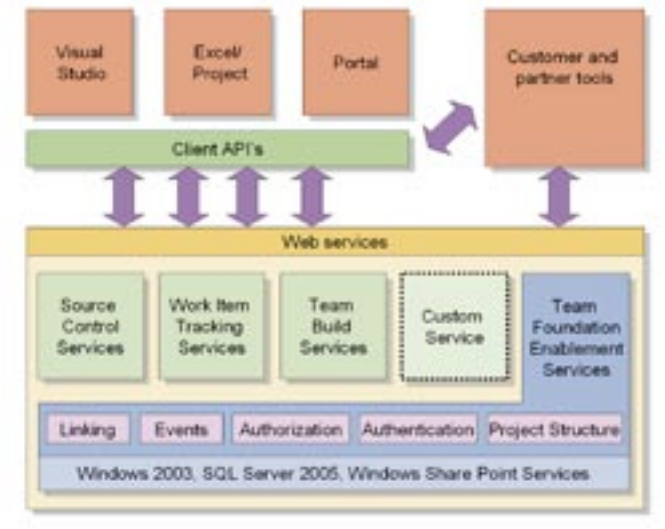
Visual Studio Team System Extensibility

IMPLEMENTEER CONTINUOUS INTEGRATION MET BEHULP VAN EVENTS, TEAM BUILD EN CUSTOM CHECK-IN POLICIES

Visual Studio Team System (VSTS) is het nieuwe product dat Microsoft gaat leveren om ons te ondersteunen bij softwareontwikkeling. Het kan tevens als basis dienen voor de inrichting van een software-ontwikkelstraat. Bij de inrichting van zo'n software-ontwikkelstraat op basis van VSTS zullen de diverse productonderdelen moeten worden ingesteld, aangepast of worden uitgebreid. Als je VSTS als basis gebruikt, dan moet deze wel alle eigenschappen bezitten om aanpasbaar en uitbreidbaar te zijn. Gelukkig heeft het ontwikkelteam dat verantwoordelijk is voor de bouw van Team System hier vanaf het ontwerp van het product rekening mee gehouden.

In dit artikel wil ik ingaan op enkele onderdelen van VSTS die kunnen worden gebruikt om zelf functionaliteit aan VSTS toe te voegen. Daarbij zal ik beschrijven hoe je zelf een continuous build kunt bouwen waarbij de build direct wordt gestart als reactie op een check-in event in de Team Foundation Server (TFS). Dit doe ik door gebruik te maken van onder andere het event-systeem en de Team Build-service. Ook zal ik beschrijven hoe je zelf een check-in policy kunt maken die controleert of code voldoet voordat deze wordt ingecheckt. Op die manier wordt op voorhand voorkomen dat onnodig een build wordt gebroken. Om een goed beeld te krijgen van de mogelijkheden die VSTS biedt op het gebied van uitbreidbaarheid en aanpasbaarheid wil ik in het kort de productarchitectuur bekijken. In afbeelding 1 is de globale architectuur van VSTS weergegeven.

In de architectuurschets is te zien dat er een aantal centrale services is waar Team System gebruik van maakt. Deze services bevinden zich in de Team Foundation Server. Verder is te zien dat de Team



Afbeelding 1. De globale architectuur van Visual Studio Team System

Foundation Server volledig communiceert met de verschillende clienttools door middel van webservices. De communicatie van de clienttools echter, waaronder de Visual Studio IDE, gaat niet rechtstreeks via de webservices, maar door middel van een client-API. Deze client-API is volledig gedocumenteerd, zodat we daar ook gebruik van kunnen maken voor het bouwen van uitbreidingen; dit in tegenstelling tot de webservices die worden aangeboden. De client-API biedt een volledig objectgeoriënteerde benadering van de Team Foundation Server die tevens een hoop extra programmeerwerk uit handen neemt, waaronder zaken als caching en authenticatie. Het wordt aanbevolen om gebruik te maken van de beschikbare client-API in plaats van de webservices rechtstreeks te benaderen, als je wilt communiceren met de Team Foundation Server. De basisservices die de Team Foundation Server bevat staan in tabel 1 beschreven.

De client-API is beschikbaar via een grote set assemblies. Al deze assemblies zijn terug te vinden in de folder: 'c:\Program Files\Microsoft Visual Studio 8\Common\IDE\PrivateAssemblies' [1] De assembly die de benodigde classes bevat om te communiceren met de Team Foundation Server is: 'Microsoft.VisualStudio.TeamFoundation.Client'. Als vertrekpunt van de communicatie maken we gebruik van de **TeamFoundationServer**-class. Deze class kan ons een referentie geven naar één van de services die in tabel 1 staan beschreven. Een referentie naar een service kan worden verkregen door een aanroep van de methode **GetService,** waarbij we als argumenten het type van de service opgeven. Stel dat we een referentie willen naar de linking service, dan kunnen we die verkrijgen door de volgende aanroep: **tfs.GetService(typeof(ILinking))**. Om de servicetypedefinities te kunnen benaderen, moeten we een additionele assembly refereren die in dezelfde folder is terug te vinden. De assemby heet 'Microsoft.VisualStudio.TeamSystem.Elead.SDK'. Deze bevat een namespace Microsoft.VisualStudio.BIS.Services [2] waar alle servicetypedefinities van de standaardservices van Team Foundation Server in staan. Een voorbeeld van het opvragen en gebruiken van een service is terug te zien in codevoorbeeld 1.

We willen een uitbreiding maken op Team Foundation Server waarbij we er voor zorgen dat, na het inchecken van source-code

Service	Beschrijving
Common Structure Service	Geeft informatie die is ingesteld t.a.v. de classificaties die in de server kunnen worden ingegeven. Classificaties bevatten o.a. iteraties die gemaakt worden in het proces en meestal ook architectuuronderdelen van de applicatie die wordt gebouwd. Deze classificaties worden gebruikt om een gedetailleerder inzicht te krijgen in de afkomst van work-items om op die manier een beter beeld te krijgen van de gegevens in het datawarehouse.
Linking Service	Deze service geeft de mogelijkheid diverse artefacten aan elkaar te relateren. Iedere service kan registreren welke artefacten hij beschikbaar stelt. De Linking Service zorgt er voor dat op een generieke manier verwijzingen naar deze artefacten kunnen worden vastgehouden. Deze service wordt o.a. gebruikt om work-items te koppelen aan change-sets in de source-control-service.
Group Security Service	Geeft informatie over de groepen en gebruikers die zijn geregistreerd in Team Foundation Server
Authorization Service	Geeft autorisatie-informatie voor gebruikers en gebruikersgroepen
Event Service	Geeft andere services de mogelijkheid events af te vuren of te ontvangen. Deze service garandeert de aflevering van events.
Work item Store	De service waarin de work-items worden opgeslagen. De service geeft ook mogelijkheden de work-items te bewerken en zorgt voor de statusovergangen en het afdwingen van de regels die zijn vastgelegd in het work-item voor een statusovergang.
Source Control Services	De service waar we de broncode kunnen opslaan en informatie over de broncode kunnen terughalen.
Team Build Services	Deze service biedt de mogelijkheid een build te starten op een van de geregistreerde buildmachines en geeft de statusinformatie over de diverse builds die plaatsvinden of hebben plaatsgevonden.

Tabel 1. Een overzicht van de basisservices in de Team Foundation Server

in de source-repository, we automatisch een nieuwe build starten die is geconfigureerd in Team Build. Hiervoor moeten we met behulp van de eventservice een event ontvangen zodra er source-code wordt ingecheckt. Laten we voor het verkrijgen van een event eens verder kijken naar de eventservice.

Team Foundation Server Events

Bij het gebruik van de Team Foundation Server vinden er in de diverse services veranderingen plaats die interessant kunnen zijn voor andere services. Dit zijn veranderingen zoals het opvoeren van work-items, statusveranderingen van work-items, inchecken van source-code, starten en uitvoeren van builds, enzovoort. Om ervoor te zorgen dat de services in de Team Foundation Server onafhankelijk van elkaar kunnen functioneren, maar toch op elkaar kunnen reageren indien dat is gewenst, is een eventservice beschikbaar waarmee events kunnen worden uitgewisseld tussen de verschillende services. Daarbij wordt de garantie gegeven dat de events worden afgeleverd en een asynchroon mechanisme beschikbaar is. Hiervoor maakt de eventservice gebruik van de Microsoft SQL Server-database en wordt ieder event opgeslagen voordat het wordt afgeleverd bij de geïnteresseerde partijen.

Bij de eventservice kan een willekeurige service duidelijk maken dat hij een bepaald type events wil versturen of ontvangen. Om events te kunnen ontvangen, moet de service een abonnement nemen op het type events waarin hij is geïnteresseerd. Bij de registratie van een abonnement moeten we duidelijk maken welk eventtype we willen ontvangen en waar eventnotificaties naartoe

```
using Microsoft.VisualStudio.TeamFoundation.Client;
using Microsoft.VisualStudio.Bis.Services;

TeamFoundationServer tfs = new TeamFoundationServer("tfsdata");
ICommonStructureService css =
   (CommonStructureService)tfs.GetService((typeof(ICommonStructureService)));

ProjectInfo[] projects = css.ListProjects();

foreach (ProjectInfo project in projects)
   Console.WriteLine(project.Name);
```

Codevoorbeeld 1. Het opvragen en gebruiken van een service

```
[SoapDocumentMethod(Action = "http://Microsoft.VisualStudio.Bis/Notify",
     RequestNamespace = "http://Microsoft.VisualStudio.Bis")]
[WebMethod]
public EventResult Notify(string eventXml)
{
   EventResult retval = new EventResult();
   retval.ReturnCode = true;
   return retval;
}
```

Codevoorbeeld 2. Het prototype van de service-interface die eventnotificaties kan ontvangen

gestuurd moeten worden. Voor de afleverlocatie kunnen we een webservice aangeven die een vast beschreven service-interface moet aanbieden. Het prototype van de service-interface die event-notificaties kan ontvangen, is terug te vinden in codevoorbeeld 2. Belangrijk om hierbij op te merken is dat bij de webservice de **Action** en de **RequestNamespace** moeten worden aangegeven. Bij het weglaten van één van beide zal er geen event worden afgeleverd bij de webservice.

Op welke events kunnen we een abonnement nemen? Welke events zijn standaard beschikbaar in een Team Foundation Server? De antwoorden op die vragen zijn terug te vinden in de manier waarop een service aan de eventservice kenbaar kan maken welk type events hij wil versturen. Bij de aanmelding van een eventtype moet een eventschema worden aangegeven. De geregistreerde schema's zijn terug te vinden in de folder: 'C:\program files\Microsoft. Visual Studio 2005 Enterprise Server\BISIISDIR\Bisserver\Event-Schemas'. Als je in deze folder kijkt, zie je een aantal .xsd- bestanden. Ieder schema dat daar staat, beschrijft een eventtype waar we een abonnement op kunnen nemen. In ons geval zijn we geïnteresseerd in het event: **CheckinEvent**.

Om een abonnement te nemen op het CheckinEvent moeten we aan een instantie van **TeamFoundationServer** een referentie vragen aan de eventservice. Dit doen we door de Methode call GetService(**IEventService**). De **IEventService** bevat een methode voor het registreren van een abonnement door middel van de methode Subscribe(). Hierbij krijgen we een subscription ID terug die we kunnen gebruiken als we de registratie van de service op een later tijdstip ongedaan willen maken. In codevoorbeeld 3 is een voorbeeld gegeven van het registreren van een webservice voor het ontvangen van events van het type **CheckinEvent**. In dit codevoorbeeld wordt eerst een verbinding gemaakt met de Team-FoundationServer 'tfsdata'. Vervolgens wordt een **DeliveryPrefe-rence**-object aangemaakt die de informatie bevat over de aflevering van events. Er wordt in dit geval aangegeven dat we dit direct na het optreden van het event willen ontvangen (**DeliverySchedule. Immediate**) en dat we graag een SOAP-call willen ontvangen op de opgegeven URL. Houd in de gaten dat we hier een locatie aangeven van een webservice die op een port 8090 moet luisteren. Dit is in het geval van deze code een webservice die wordt gehost in de Visual Studio-omgeving. Zorg er voor dat als je dit soort code wilt

```
using Microsoft.VisualStudio.TeamFoundation.Client;
using Microsoft.VisualStudio.Bis.Services;

TeamFoundationServer tfs = new TeamFoundationServer("tfsdata");

//zet the delivery preference naar immediate delivery via SOAP
DeliveryPreference dp = new DeliveryPreference();
dp.Schedule = DeliverySchedule.Immediate;
dp.Type = DeliveryType.Soap;

//de URL voor de callback WS
dp.Address = @"http://localhost: 8090/SubscribetoEvents/BuildGauntlet.asmx";

//user name van de subscriber
string userName = "vsts\\tfsSetup";

IEventService esProxy = (IEventService)tfs.GetService((typeof(IEventService)));
int id = esProxy.Subscribe(userName, "CheckinEvent", null, dp);
```

Codevoorbeeld 3. Er wordt een verbinding gemaakt met de TeamFoundationServer

testen. Als je geen IIS als webserver gebruikt, zorg er dan ook voor dat je de Visual Studio webprojectinstellingen aanpast, zodat het portnummer niet dynamisch wordt gegenereerd. Dat is namelijk de defaultinstelling.

Team Build starten als reactie op Check-in event

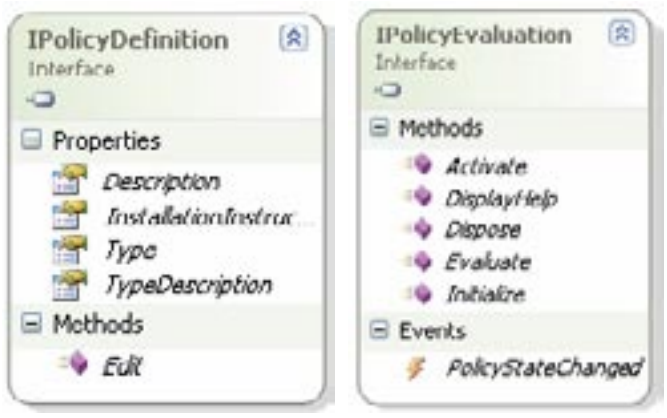
De team build-service is ook via een objectmodel te benaderen. Als we vanuit de eventnotificatie een build willen starten, dan moeten we daarvoor een instantie van de **BuildController** zien te verkrijgen. Deze is aan te maken als je de locatie van de build-controller weet. (Dit is een URL van een webservice op de application tier) In de assembly 'Microsoft.Visualstudio.TeamSystem.TeamBuild.Client' is de **BuildController**-class beschikbaar. Door de juiste url bij de **BuildController**-webservice aan te geven kunnen we een build starten. Bij het starten van een build moeten we nog een aantal argumenten meegeven op basis waarvan de BuildController de juiste build kan starten. In het codevoorbeeld 4 is een voorbeeld van de eventnotification-implementatie gegeven, waarbij een build van het type 'DailyBuild' wordt gestart op de buildmachine 'tfsdata'. 'DailyBuild' moet wel bestaan en geconfigureerd zijn in Team Build.

Voorkomen van foutieve check-in

Tijdens het inchecken van source-code in de Team Foundation Server kunnen ontwikkelaars worden geholpen bij het voorkomen van fouten die een eventuele build kunnen breken. Dit is mogelijk met behulp van policies die worden geëvalueerd bij het inchecken. De policies die we willen afdwingen kunnen per project worden ingesteld. Microsoft levert met Team System zelf standaard een viertal policies mee. Dit zijn:

- Clean Build
- Code Analysis
- Testing Policy
- Work Items

Deze policies dekken natuurlijk niet alle mogelijke scenario's die je zou willen controleren. Omdat Microsoft onmogelijk alle policies kan implementeren die een ontwikkelaar tijdens het inchecken zou willen laten controleren, is ook hier aan uitbreiding gedacht. Stel je het scenario voor waarbij moet worden gecontroleerd dat bij het inchecken van C#-projectfiles XML-documentatiegeneratie is aangezet. Zou de documentatiegeneratie niet aan staan, dan zou dit kunnen resulteren in het blokkeren van de build, als documentatie wordt gegenereerd tijdens de build met behulp van tools zoals NDOC [3]. Om zelf een policy te kunnen maken, moeten we ook hier eerst kijken naar de architectuur achter het afdwingen van de policies.



Afbeelding 2. Interfacedefinities

Policy-architectuur

De policy-architectuur kan worden opgesplitst in twee fasen van gebruik, te weten:

1. Vastleggen van gewenste policies
2. Het afdwingen van een policy tijdens het inchecken van source-code.

In tegenstelling tot wat je misschien zult verwachten worden policies niet op de server maar binnen Visual Studio afgedwongen. Dit is een belangrijke eigenschap die direct impliceert dat een policy geïnstalleerd moet worden op het werkstation van de ontwikkelaar. Een policy is een normale .NET-assembly die een class implementeert die twee interfaces heeft. De eerste interface is noodzakelijk voor de definitiefase en geeft meta-informatie aan de omgeving over de policy en heet **IPolicyDefinition**. De tweede interface wordt gebruikt tijdens het evalueren van de policy en heet ook zeer toepasselijk **IPolicyEvaluation**. In afbeelding 2 zijn de interfacedefinities weergegeven.

Policy-definitie

De IPolicyDefinition-interface is belangrijk tijdens het definiëren van de gewenste policies voor een project. Als een projectmanager of lead developer policies wil definiëren, kan hij hiervoor in de Visual Studio Team Explorer de optie kiezen voor het configureren van source-control. Hierbij zal een dialoog worden getoond met daarin alle beschikbare policies die op de locale machine zijn geregistreerd. De lijst met beschikbare policies wordt verkregen uit de registry. (HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Hatteras\Checkin Policies) Als we een eigen gemaakte policy willen gebruiken, moeten we dus onze assembly registreren in de registry.

N.B: Als je een policy registreert in de registry, zorg er dan voor dat de keynaam exact de naam van de assembly is. Dit is de naam van de dll zonder de .dll-extensie

Op het moment dat de dialoog wordt getoond zullen de **Type-** en **Description**-properties worden aangeroepen op de IPolicyDefinition-interface om informatie te kunnen tonen aan de gebruiker over

```
[SoapDocumentMethod(Action = "http://Microsoft.VisualStudio.Bis/Notify",
                    RequestNamespace = "http://Microsoft.VisualStudio.Bis")]
[WebMethod]
public EventResult Notify(string eventXml)
{
   BuildController bc = new BuildController();

   bc.Url = "http://tfsdata:8080/TeamBuild/BuildController.asmx?op=StartBuild";
   bc.Credentials = System.Net.CredentialCache.DefaultCredentials;

   BuildStartParams bp = new BuildStartParams();
   bp.TfsServer = "http://tfsdata:8080";
   // Haal uit de eventXML de project naam op, dit is hier weggelaten
   bp.PortfolioProject = "Backup";
   bp.ConfigName = "DailyBuild";
   bp.BuildDirectory = "c:\\temp";
   bp.BuildType = "OnDemand";
   bp.BuildMachine = "tfsdata";

   bc.StartBuild(bp);

   EventResult retval = new EventResult();
   retval.ReturnCode = true;
   return retval;
}
```

Codevoorbeeld 4. De Eventnotification-implementatie

```
using Microsoft.VisualStudio.Hatteras.Client;

[Serializable]
 public class XmlDocumentationPolicy: IPolicyDefinition, IPolicyEvaluation
 {
  // IPolicyDefinition is hier niet weergegeven omdat dit niet meer is
  // dan het retouneren van de gewenste strings op de verschillende
  // methodes.

  // Deze methode voert de daadwerkelijke policy evaluatie uit.
  public PolicyFailure[] Evaluate()
  {
   // check the csproj file if xml documentation is turned on.
   ArrayList failures = IsXMLDocumentationTurnedOn();
   if (failures.Count == 0)
    return noFailures;
   else
   {
    PolicyFailure[] toReturn = new PolicyFailure[failures.Count];
      failures.CopyTo(toReturn);
    return toReturn;
   }
  }
  private ArrayList IsXMLDocumentationTurnedOn()
  {
   // Zoek uit of er een cspoj file wordt ingecheckt
   ArrayList csprojFiles = new ArrayList();
   ArrayList failures = new ArrayList();

    foreach(PendingChange change in
        pendingCheckin.PendingChanges.AllPendingChanges)
    {
     if (change.FileName.EndsWith("csproj"))
     {
      csprojFiles.Add(change.LocalItem);
     }
    }
     if (csprojFiles.Count >0)
     {
      // Laad de file in een XML document en zoek voor de node
      // DocumentationFile
      foreach (string csprojFile in csprojFiles)
      {
       XmlDocument csprojXml = new XmlDocument();
       csprojXml.Load(csprojFile);

       XmlNamespaceManager xmlNS = new
            XmlNamespaceManager(csprojXml.NameTable);
       // Omdat de MSBuild file een default namespace heeft
       // moeten we een namespace mapping maken, omdat XPath
       // geen default namespace kent. Als we deze mapping niet
       // aanmaken krijgen niets terug uit de SelectSingleNode.
       xmlNS.AddNamespace("a",Strings.vsDotNetDefaultNamespace);
       XmlNode n = csprojXml.SelectSingleNode("//a:
         DocumentationFile", xmlNS);
       if ((n == null) || (n.InnerText.Length== 0))
      {
       failures.Add(new
          PolicyFailure(string.Format(Strings.policyViolationMessage,
                 csprojFile), this));
      }
     }
    }
    return failures;
   }
   public void Initialize(IPendingCheckin pendingCheckin)
   {
    this.pendingCheckin = pendingCheckin;
    this.pendingCheckin.PendingChanges.CheckedPendingChangesChanged += new
        EventHandler(PendingChanges_CheckedPendingChangesChanged);
   }
  public void Dispose()
  {
   pendingCheckin.PendingChanges.CheckedPendingChangesChanged -= new
       EventHandler(PendingChanges_CheckedPendingChangesChanged);
   pendingCheckin = null;
  }
  private void PendingChanges_CheckedPendingChangesChanged(Object sender,
      EventArgs e)
  {
  OnPolicyStateChanged(Evaluate());
  }
  public event PolicyStateChangedHandler PolicyStateChanged;
  private void OnPolicyStateChanged(PolicyFailure[] failures)
  {
   if (PolicyStateChanged != null)
   {
    PolicyStateChanged(this, new PolicyStateChangedEventArgs(failures, this));
   }
  }
 public void Activate(PolicyFailure failure)
 {
  MessageBox.Show(Strings.policyFailureHelpText);
 }
 public void DisplayHelp(PolicyFailure failure)
 {
  MessageBox.Show(Strings.policyFailureHelpText,
    "XML Documentation Policy Help");
 }
 [NonSerialized]
 public static readonly PolicyFailure[] noFailures = new PolicyFailure[0];
 private IPendingCheckin pendingCheckin = null;
 #endregion
 }
}
```

Codevoorbeeld 5. Een implementatie van een policy

de policy. Als de policy wordt geselecteerd dan zal de **Edit**-methode worden aangeroepen op de IPolicyDefinition-interface, zodat de policy de optie kan geven aan de gebruiker om instellingen vast te leggen voor toekomstige policy-evaluatie. Deze configuratie-informatie wordt opgeslagen in de source-control-service door de implementerende class te serialiseren in de server. Bij de evaluatie van de policy zal de class worden gedeserialiseerd en kan met de eerder aangegeven configuratie worden gewerkt. Na de aanroep van de **Edit**-methode zal de property **InstallationInstructions** worden aangeroepen. Deze property moet een string terug geven die vervolgens wordt gebruikt om de eindgebruiker informatie te verschaffen over hoe hij een policy moet installeren als deze niet is geïnstalleerd op de locale machine.

Policy-evaluatie

De **IPolicyEvaluation**-interface wordt gebruikt tijdens de check-in-procedure. De interface heeft naast een set methoden een **PolicyStateChanged-event** waar Visual Studio een abonnement op zal nemen. Als de policy van status verandert en je dit wilt aangeven aan de Visual-Studio-omgeving, dan dient dit te worden gedaan door het afvuren van het event. Het is namelijk mogelijk dat je een policy schrijft die de veranderingen in files bijhoudt door middel van bijvoorbeeld een FileSystemWatcher. Zodra de policystatus dan is veranderd, kan dit met behulp van dit event worden doorgegeven aan de Visual Studio IDE, zodat deze de user interface hierop kan aanpassen.
De Policy wordt geactiveerd door de Pending-check-in-dialog. Zodra het Check-in- of policy-icoon voor de eerste keer wordt geklikt, zal de Initialize-methode worden aangeroepen. Dit is het moment waarop we informatie ontvangen over de uitstaande check-in en kunnen we een abonnement nemen op een event dat door Visual Studio wordt afgevuurd zodra er iets verandert in de lijst met geselecteerde items voor de uitstaande check-in. Dit event heet **CheckedPendingChangesChanged**. De informatie over de pending changes bevat onder andere een lijst met bestanden die worden ingecheckt. Als bijvoorbeeld de gebruiker een file meer of minder aanvinkt in de lijst met check-in items, zal het **CheckedPendingChangesChanged**-event worden afgevuurd en kunnen we opnieuw een evaluatie van de policy uitvoeren. Voor het scenario van het valideren van C#-projectfiles, selecteren we uit de lijst met pending changes de .csproj-files en onderzoeken we deze op de aanwezigheid van een XML-element met de naam **DocumentationFile**. We kunnen de csproj- file openen met een XML-document, omdat vanaf Visual Studio 2005 alle projectfiles voortaan een MSBuild-file zijn. Deze zijn altijd in een XML-formaat en dat maakt het zoeken in deze files zeer eenvoudig. Indien we de **DocumentationFile**-node vinden en deze een waarde heeft, dan geeft de policy aan dat er geen overtreding is. In alle andere gevallen zal worden gerapporteerd dat de XML-documentatie niet is aangezet.

In codevoorbeeld 5 is een voorbeeldimplementatie te zien van de policy die controleert of in een C#-projectfile de sourcecode-documentatiegeneratie aan staat. In het voorbeeld wordt de implementatie van **IPolicyDefinition** niet weergegeven, omdat dit niet

veel meer is dan het retourneren van strings die de juiste waarde hebben.

Tips voor het bouwen van custom policies

Een paar tips die je kunnen helpen bij het bouwen van je eigen policies:

- Om een policy te kunnen debuggen, kun je Visual Studio als debugproces aangeven. Zodra je dan de policy hebt geactiveerd, kan je met het pending-check-in-window de policy-evaluatie starten en wordt gestopt bij een eerder geplaatst breakpoint in de code.
- Zodra je de implementatie van de policy verandert waardoor de serialisatie-stream van het object verandert, moet je de policy eerst deactiveren voor het project en daarna weer activeren. Op die manier forceer je dat een nieuwe instantie van de policy wordt geserialiseerd en de serialisatiedata in de server worden vervangen. Doe je dit niet, dan zul je een internal exception zien die je niet kunt debuggen, omdat dit al bij objectcreatie misgaat.
- De assembly die de interfaces **IPolicyEvaluation** en **IPolicyDefintion** bevat heeft de naam: **Microsoft.visualStudio.Hatteras.Client.dll** en kan in de eerder genoemde folder worden gevonden.

Aanpassingen en uitbreidingen

Visual Studio Team System is ontworpen met de mogelijkheid om te kunnen uitbreiden. Er wordt op diverse plaatsen in het product de mogelijkheid geboden zelf aanpassingen of uitbreidingen te maken. Naast de mogelijkheden die hier zijn beschreven, zijn er ook nog vele andere mogelijkheden voor aanpassingen. Denk daarbij aan het aanpassen van de methodologietemplates, work-item-definities, datawarehouse-uitbreidingen, test-typen, Team-Portal-aanpassingen, custom reports, enzovoort. Voor al deze zaken heeft Microsoft ook een Visual Studio Team System Extensibility-kit uitgebracht met documentatie en voorbeelden hoe je dit allemaal kunt doen. Ik kan je aanbevelen deze kit eerst op te halen als je zelf aan de slag wilt met het aanpassen van Visual Studio Team System. Je kunt deze Extensibility-kit terug vinden op de volgende locatie: www.vsipdev.com/downloads.

Marcel de Vries is werkzaam bij Info Support in Veenendaal. Hij is als .NET-architect mede verantwoordelijk voor innovatie in het Professional Development Center, waar de software ontwikkelstraat Endeavour wordt ontwikkeld. Vanuit zijn rol in het PDC is hij ook de trekker van het Whidbey Technical Adoption Program (TAP). Vanuit het TAP-programma is Marcel al sinds juni 2004 actief aan het werk met Visual Studio Team System.

Nuttige internetadressen:

http://blogs.infosupport.com/marcelv
http://blogs.msdn.com/robcaron/default.aspx
http://lab.msdn.microsoft.com/teamsystem/default.aspx
http://forums.microsoft.com/msdn/default.aspx?ForumGroupID=5

NOTEN

[1] De informatie in dit artikel is gebaseerd op VSTS Bèta 2. De definitieve API's kunnen nog veranderen in de uiteindelijke versie.

[2] De afkorting BIS in de namespace is afkomstig uit de eerder gebruikte code naam voor Team System. Dit was Burton. BIS staat voor Buron Integration Services. Al deze code namen zullen voor de definitieve versie verdwijnen, en zullen dus aanpassingen in de gegeven code voorbeelden tot gevolg hebben

[3] NDOC is een open source product dat XML-documentatie kan omzetten naar documentatie die er uit ziet als de MSDN help. Ga voor meer informatie naar: http://ndoc.sourceforge.net