Oslo, het nieuwe modelleer-platform van Microsoft

Felienne Hermans

Tijdens de laatste Professional Developers Conference (PDC) in Los Angeles kon de wereld voor het eerst kennismaken met Oslo, het nieuwe modelleer-platform van Mircrosoft. Dit platform bestaat uit de modelleertaal M, de visualisatietool Quadrant en een SQL-database waarin modellen worden opgeslagen genaamd Repository. In dit artikel nemen we het eerste element, de programmeertaal M, verder onder de loep.

De programmeertaal M bestaat uit drie verschillende talen: MGraph, MGrammar en MSchema. We zullen ze één voor één beschrijven te beginnen met MSchema. Dit is een taal om modellen te beschrijven. Een schema om een boek met de eigenschappen 'titel', 'auteur' en 'datum van uitgifte' te beschrijven ziet er in MSchema uit zoals onderstaand codevoorbeeld. Het schrijven met MSchema heeft overigens veel overeenkomsten met het maken van een XSD, DTD of XML-schema.

```
module Examples.Boeken {
  type Boek
  {
    Id: Integer64 = AutoNumber();
    Titel : Text;
    Auteur : Text;
    UitgaveDatum : Date;
  } where identity Id;
  Bibliotheek: Boek*;
};
```

Uit dit schema kan Oslo direct SQL-code genereren om een bijbehorende tabel in de database aan te maken. Je doet dit door in Intellipad in het M-menu Reach SQL Preview te kiezen. Het M-menu verschijnt wanneer je een .m bestand opent.

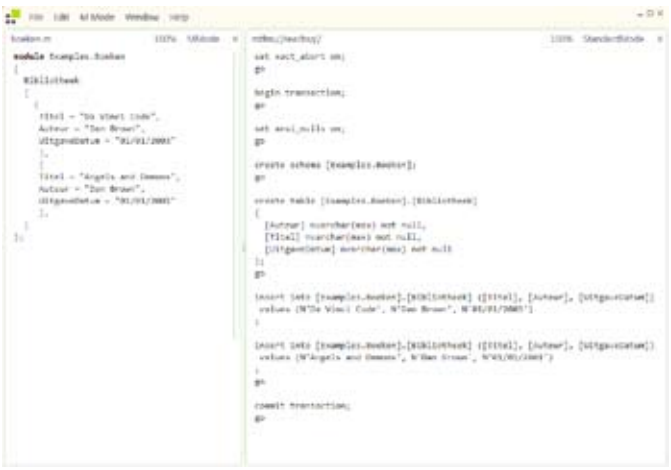
MGraph is een taal die gebruikt kan worden om data te beschrijven. Als we de analogie met XML doorvoeren, is dit het schrijven van het XML-bestand zelf. Data over boeken die voldoet aan het hierboven beschreven stukje MSchema, ziet er zo uit:

```
module Examples.Boeken {
  Bibliotheek
  {
    {
     Titel = "Da Vinci Code",
     Auteur = "Dan Brown",
     UitgaveDatum = "01/01/2003"
     },
     {
     Titel = "Angels and Demons",
     Auteur = "Dan Brown",
     UitgaveDatum = "01/01/2001"
     },
  }


};
```

De derde taal in M is MGrammar. Met MGrammar kun je een definitie van je eigen formele taal schrijven, die 'live' in Intellipad geparst kan worden.Een taal in MGrammar moet altijd een commando 'Syntax Main' bevatten, die aangeeft welke regels er geparst

FIGUUR 1: HET MSCHEMA VOOR BOEK IN SQL-PREVIEWMODUS

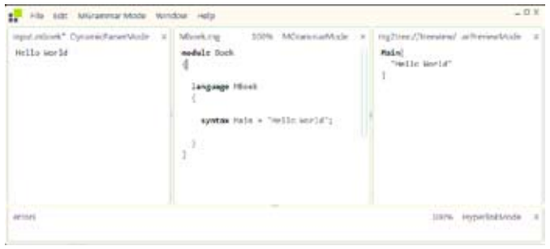


FIGUUR 2: HET MGRAPH VOOR BOEK IN SQL-PREVIEWMODUS

Intellipad

Intellipad is de teksteditor die bij Oslo SDK geleverd wordt. Zorg ervoor dat je de 'samples enabled' versie gebruikt, dit is de versie die ook in alle verdere voorbeelden gebruikt is. Om de live-parsemodus te gebruiken, open je eerst het bestand dat geparst moet worden (de linkerkant). Vervolgens open je een MGrammar-bestand met Ctrl-Shift-t.

kunnen worden. De regel 'Syntax Main = "Hello World";' geeft aan dat de taal alleen maar mag bestaan uit Hello World.



FIGUUR 3: HELLO WORLD IN LIVE-PARSE MODUS. LINKS DE INVOER, IN HET MIDDEN DE BESCHRIJVING VAN DE TAAL, RECHTS DE GEPARSTE DATA

MGrammar bestaat verder nog uit het 'Token' commando, waarmee we een aantal karakters kunnen groeperen.

```
module Boek {
  language MBoek
  {
    token Char = 'a'..'z'|'A'..'Z';
    token Name = Char+;

    syntax Main = "Hello " Name;
  }
}
```

De parsetree

Als met bovenstaande uitgebreidere Hello-taal, de tekst 'Hello Bill' geparst wordt, ziet dat er zo uit:

```
mg2tree://treeview/  100%  MGrammarPreviewMode
Main[
  "Hello ",
  "Bill"
```

FIGUUR 4: GEPARSTE BOOM BEHORENDE BIJ 'HELLO BILL'

Omdat de naam het enige variabele element in deze taal is,willen we de boom zo maken dat alleen die informatie terug te zien is. Daarvoor breiden we de syntaxregel Main uit met een '=>'.

```
syntax Main = "Hello " n:Name => n;
```

De geparste boom bestaat nu alleen uit de naam.'Bill' in dit geval. Als deze syntaxregel onderdeel uit zou maken van een grotere taal, is het handig als deze node in de boom gelabeld wordt. Op die manier is het bij het verwerken van de boom duidelijk wat deze informatie voorstelt. Hiertoe breiden we de syntaxregel nog wat verder uit.

```
syntax Main = "Hello " n:Name => Name[n];
```

De boom bestaat nu uit één node, namelijk

```
Name[
  "Bill"
]
```

Dit is een korte inleiding op de syntax van MGrammar. De volledige beschrijving van de taal is te vinden op http://msdn.microsoft.com/en-us/library/dd129869.aspx. Alle presentaties van de PDC zijn te vinden op Channel 9 (http://channel9.msdn.com/pdc2008/).

Domeinspecifieke talen

Met MGrammar kun je iedere taal maken, je kunt denken aan je eigen programmeertaal. Maar uit de presentaties die gegeven zijn op de PDC, blijkt dat Microsoft met Oslo programmeurs wil stimuleren om hun eigen domeinspecifieke taal (DSL) te bouwen. Een DSL is een taal die gericht is op een specifiek, klein domein, zoals bijvoorbeeld 'verzekeringen' of 'databases'. In die zin kun je een DSL zien als het tegenovergestelde van een programmeertaal, die juist geschikt is voor ieder domein. Voorbeelden van DSL's zijn SQL en HTML. Een simpele DSL voor het domein 'boeken' zou er bijvoorbeeld zo uit kunnen zien:

FIGUUR 5: EEN TAAL OM INFORMATIE OVER BOEKEN OP TE SCHRIJVEN

Parse Errors

Wat gebeurt er wanneer we een bestand proberen te parsen dat niet voldoet aan onze taal? Oslo genereert automatisch foutmeldingen als er wat mis gaat; met beknopte informatie over waarom het parsen niet lukt.

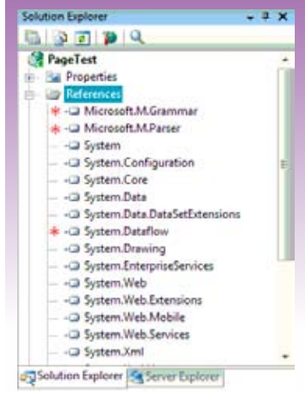
FIGUUR 6: ALS HET PARSEN MISLUKT, TOONT INTELLIPAD FOUTINFORMATIE

Runtime Interpretatie

Het dynamisch parsen van je eigen taal is indrukwekkend, maar wat kan een ontwikkelaar in de praktijk met een domeinspecifieke taal? Oslo's antwoord hierop is runtime interpretatie. In Visual Studio kun je de hele geparste boom rechts in figuur 5 opvragen, doorlopen en acties uitvoeren op basis van de waarden die je tegenkomt. Dat klinkt ingewikkeld, maar is eenvoudiger dan je zou denken.

Oslo in VS 2008

Om in Visual Studio gebruik te kunnen maken van de mogelijkheden van Oslo, moet je referenties naar een aantal dll's toevvoegen. Deze dll's zijn te vinden in de map Bin van de SDK.



BENODIGDE REFERENCES

MGrammar-bestanden compileren

MGrammar compileren

In de rest van dit artikel wordt gesproken over mgx-bestanden. Dit zijn gecompileerde MGrammar-bestanden. Het compileren van een .mg-bestand, doe je met executable mg.exe, die in de Oslo SDK zit. Omdat deze versie van Oslo nog pre-alpha is, moet dat via de commandline.

Als de benodigde referenties toegevoegd zijn (zie kader Oslo in VS 2008), kunnen we de dynamische parser aanroepen met het volgende stukje code:

```
// Load image and instantiate a corresponding dynamic parser
DynamicParser language = MGrammarCompiler.LoadParserFromMgx
(imageFileName, languageName);
```

LoadParserFromMgx heeft twee argumenten imageFileName, de eerste is het mgx-bestand (zie kader MGrammar compileren). In een .mg-bestand kunnen meerdere talen voorkomen. Door middel van het tweede argument geef je aan welke taal er gebruikt wordt door de parser. Vervolgens kan ieder invoerbestand geparst worden door de methode ParseObject van de DynamicParser.

```
// Process the input using the grammar and get the root node of
the output graph
object rootNode = language.ParseObject(input, ErrorReporter.
Standard);
```

Ook deze methode wordt aangeroepen met twee argumenten.

Het eerste is het pad van het invoerbestand, het tweede is een ErrorReporter. Het laatste dat nodig is, voordat de boom verwerkt kan worden, is een GraphBuilder.

```
// Create the GraphBuilder
GraphBuilder ParseTree = new GraphBuilder();
```

Vervolgens kan informatie over de rootnode (en over iedere andere node) opgevraagd worden via de GraphBuilder. Een aantal handige functies zijn:

- GetLabel(node) geeft het label van de node terug. Dat is de tekst die voor de rechte haken staat in de boom. Het label van de rootnode in het voorbeeld is 'Boek'
- GetSequenceCount(node) geeft het aantal kinderen terug dat deze node heeft. 'Boek' heeft drie kinderen, namelijk de auteur, de titel en de uitgiftedatum. In de syntax van de taal is dit terug te zien in ' Boek[Name[n],Auteur[a], Datum[d]]'
- GetSequenceElements(node) geeft een verzameling van alle kinderen terug. Met een for each kunnen we operaties op alle kinderen uitvoeren.
- GetSequenceElementAt(node,i) geeft het kind op de i-de plaats, waarbij met 0 begonnen wordt met tellen.

De GraphBuilder

Het uitlezen van de boom via de GraphBuilder voelt nog een beetje onhandig aan. Hopelijk zal je in de toekomst met een commando als Boek.Auteur de informatie kunnen opvragen.

Een simpele recursieve functie om de hele boom naar de consolete schrijven, ziet er zo uit:

```
static void EnumerateOutputGraph(GraphBuilder builder, object
node, int level)
{
  // Display label for current node
  Console.WriteLine("Node with Label {0} on Level {1}", builder.
  GetLabel(node), level);

  // Enumerate all child values
  foreach (object childNode in builder.GetSequenceElements-
  (node))
  {
    // If the child node is a string, it is matched input so
    display
    // its value and skip to next value since it can't have
    children
    if (childNode is string)
    {
      string note = childNode.ToString();
      Console.WriteLine("Matched Input {0} on Level {1}",
      note,(level+1));
      continue;
    }
    // Otherwise, the value is a node for a syntax rule, so
    // enumerate its children
    EnumerateOutputGraph(builder, childNode, level+1);
  }
}
```

Als deze functie op de volgende aangeroepen wordt, zien we op de console het volgende resultaat (zie figuur 7).

```
// Recursively navigate the output graph using GraphBuilder
EnumerateOutputGraph(new GraphBuilder(), rootNode, 0);
```

```
Node with Label Boek on Level 0
Node with Label Name on Level 1
Matched Input Da Vinci Code on Level 2
Node with Label Auteur on Level 1
Matched Input Dan Brown on Level 2
Node with Label Datum on Level 1
Matched Input 12-03-2003 on Level 2
```

FIGUUR 7: TEKSTUELE WEERGAVE VAN DE BOOM UIT FIGUUR 5

MWebApplicatie

MWebApplicatie is een taal om webapplicaties te beschrijven, gemaakt in MGrammar. Deze taal is een simpel voorbeeld van een DSL die door een ontwikkelaar ontwikkeld en gebruikt zou kunnen worden. De taal bestaat uit elementen om webpagina's te beschrijven; de eerste regel bestaat uit configuratieparameters voor de webapplicatie. Daarna volgen een of meerdere regels die pagina's uit de webapplicatie beschrijven.

FIGUUR 8: MWEBAPPLICATIE IN INTELLIPAD

Nu gaan we net als hierboven de boom doorlopen. Het gemak van het maken van een runtime is het feit dat het at designtime precies bekend is hoe de boom opgebouwd is. In plaats van de gehele boom recursief te doorlopen, kunnen we nu die informatie uit de boom halen die nodig is om, in dit geval, een webapplicatie te draaien. Op deze manier bijvoorbeeld wordt de Connection-String opgebouwd:

```
object AppNode = language.ParseObject(input, Error-
Reporter.Standard);
GraphBuilder builder = new GraphBuilder();

string ServerName = builder.GetSequenceElementAt
(AppNode, 1).ToString();
string DataBaseName = builder.GetSequenceElementAt
(AppNode, 2).ToString();
string ConnectionString = "Data Source=" + ServerName
+ ";Initial Catalog=" + DataBaseName + ";";
```

Omdat de syntax van de taal bepaalt dat er het tweede kind van de Application-node de naam van de server is, kan dat meteen in een string opgeslagen worden. Er hoeft niet meer gecontroleerd te worden of de naam leeg is, want als dat het geval was geweest, was er tijdens het parsen al een fout opgetreden.
Vervolgens wordt de webpagina dynamisch opgebouwd op basis van het type van de pagina. Als een pagina van het type 'View' is,moet een GridView worden getoond die de inhoud van de tabel weergeeft die als argument is meegegeven. In het voorbeeldinvoerbestand is de eerste pagina van het type View met argument 'TestTable'.

```
//Traverse the tree to find the PageInformation
object PagesNode = builder.GetSequenceElementAt(AppNode,5);
object PageNode = builder.GetSequenceElementAt
(PagesNode, 0);
string PageName = builder.GetSequenceElementAt
(PageNode, 0).ToString();
object PageType = builder.GetSequenceElementAt
(PageNode, 1);
string PageTypeName = builder.GetLabel(PageType).
ToString();

 if (PageTypeName == "View")
{
   form1.Controls.Clear();
   object QueryTable = builder.GetSequence-
   ElementAt(PageType, 0);
   GridView View = new GridView();
   SqlDataSource ViewSource = new SqlDataSource
   (ConnectionString,"Select * from " + QueryTable.
   ToString());
   View.DataSource = ViewSource;
   View.DataBind();
   form1.Controls.Add(View);
}
```

Deze runtime kan dus beschrijvingen in een zelfgemaakte taal lezen en omzetten in een webapplicatie. Deze taal zou je bijvoorbeeld kunnen aanbieden aan een systeembeheerder die weet wat voor databases er in een bedrijf aanwezig zijn, maar die geen .NET-kennis heeft.

De kritische noot

Dit nieuwe platform van Microsoft roept ook vragen op.

- MSchema biedt de mogelijkheid om tabellen en instanties in de database te zetten, maar hoe zouden deze weer gewijzigd of verwijderd kunnen worden?
- Wat is de relatie tussen enerzijds MGraph en MSchema en anderzijds MGrammar?
- Hoe verhoudt Oslo zich tot DSL Tools? Immers, ook DSL Tools is bedoeld om domeinspecifieke talen mee te ontwikkelen. Maar in tegenstelling tot Oslo dat bedoeld lijkt te zijn voor runtime interpretatie, is DSL Tools gericht op het genereren van code. Het is natuurlijk mogelijk om een runtime te maken die code genereert, maar dat lijkt niet de richting te zijn waar Microsoft naartoe wil. Over wat ze precies voor de toekomst van Oslo zien, is nog veel onduidelijkheid.

Quadrant en de repository

Op de PDC werd Oslo door Doug Purdy bestempeld als prepre-alpha, dus niet alle onderdelen zijn stabiel, of zelfs beschikbaar. De visualisatietool Quadrant waar we eerder naar verwezen is geen onderdeel van de Oslo SDK en over de mogelijkheden is nog niet zoveel bekend. Het is in ieder geval
mogelijk om modellen geschreven in M op allerlei manieren te visualiseren en waarschijnlijk zal het in de toekomst ook mogelijk zijn modellen in Quadrant te maken.

Het PerPlex-project

Eind 2008 is Microsoft samen met Avanade en Technische Universiteit Delft het PerPlex-project om DSL's in de praktijk te onderzoeken. Binnen dit project worden nieuwe technologieën op het gebied van modelgedreven ontwerpen, zoals bijvoorbeeld Oslo, bestudeerd en worden nieuwe technieken ontwikkeld. Vervolgens kunnen deze technieken in de praktijk worden getest en vergeleken met de huidige methodes die gebruikt worden bij Avanade en haar klanten. Hierbij werken onderzoekers nauw samen met Avanade.

Links

http://msdn.microsoft.com/en-us/library/dd129869.aspx
http://geekswithblogs.net/cyoung/archive/2009/01/05/128369.aspx
http://social.msdn.microsoft.com/Forums/en-US/oslo/thread/3dc54671-89b8-4cc3-a42c-8ecf7e078dff/
http://douglaspurdy.com/2008/09/06/what-is-oslo/
http://channel9.msdn.com/pdc2008/
http://msdn.microsoft.com/nl-nl/library/dd185457(en-us,VS.85).aspx

Felienne Hermans, heeft technische informatica gestudeerd aan de TU Eindhoven en is nu werkzaam als promovendus aan de TU Delft, binnen het bovengenoemde PerPlex-project.