Microsoft .NET is de snelst opkomende ontwikkelarchitectuur van dit moment en van de komende jaren. Onderzoeksbureau Gartner schat dat in 2005 zo'n veertig procent van alle systeemontwikkelprojecten wordt uitgevoerd met .NET. De Smart .NET Express is een initiatief dat bedoeld is om kennis en ervaring op het gebied van .NET, Smart en UML optimaal te benutten en te verspreiden. Dit artikel werpt een blik op een aantal van de gekozen pragmatische oplossingen.

achtergrond

Herbruikbare patronen met .NET

Bundeling van kennis en ervaring

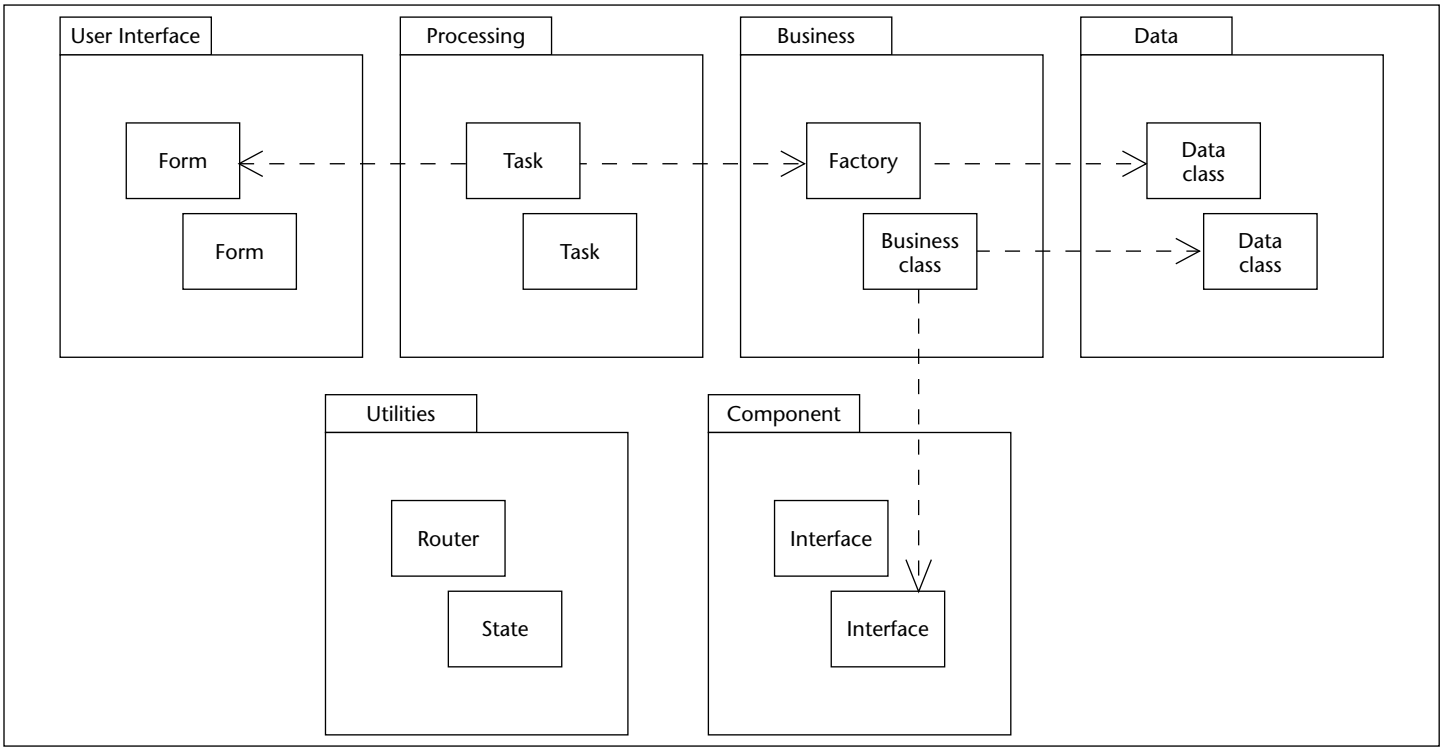
De allereerste .NET-projecten in Nederland zijn uitgevoerd volgens de agile systeemontwikkelmethode Smart. Medewerkers werken hierbij nauw samen in live projecten, die worden uitgevoerd volgens Smart, met als doel methode en technologie zo snel mogelijk onder de knie te krijgen. De Smart .NET Express beantwoordt hiervoor klantvragen in realistische settings. Ondertussen creëren de teams oplossingen voor problemen die bij volgende projecten onmiddellijk bruikbaar zijn.

MODELLENTECHNIEKEN De agile systeemontwikkelmethode kent een aantal fasen. Een project start met het uitbrengen van een proposal, waarna een haalbaarheidsstudie wordt uitgevoerd. Al tijdens deze fase wordt gebruikt gemaakt van modelleertechnieken. Zo worden al tijdens de proposal high level use cases opgesteld. Aan de hand van deze use cases wordt een goede schatting verkregen van de omvang van het project. Een go/no-go aan het eind van de haalbaarheidsstudie, worden projecten vervolgd in korte iteraties van drie tot zes weken. Tijdens deze iteraties wordt allereerst vastgesteld welke use cases worden uitgewerkt tijdens de iteraties. Vervolgens wordt in een dagelijks terugkerende cyclus gewerkt aan het ontwerp, de realisatie en het testen van de use cases. Een team in optima forma kan iedere dag één of meerdere use case compleet uitwerken.

> Bijna wekelijks worden vanuit de projecten nieuwe onderdelen aangedragen voor dit Ecosystem

Om een dergelijke snelheid te bereiken is het van belang de modellering in dienst te stellen van de te realiseren oplossing. Iedere use case wordt zo ontworpen in een activity diagram en eventueel een sequence diagram. Het activity diagram dient als basis voor de testscenario's en testgevallen. Het sequence diagram transformeert het ontwerp naar de gekozen architectuur. Smart maakt hiervoor gebruik van een n-tier referentiearchitectuur. Deze architectuur kent eenvoudige lagen als user interface, processing, business, utilities en data, en is uitbreidbaar voor component based development. Iedere use case kent hierin dezelfde doorsnede. De use case wordt geïmplementeerd door een task die de user interface aanstuurt en gegevens heen en weer transporteert uit business classes en factories. Deze architectuur is weergegeven in afbeelding 1. Tijdens eerdere projecten is deze referentiearchitectuur uitgewerkt voor het ontwikkelen van webapplicatie met het .NET framework.

ONTHOUDEN VAN STATE Iedere task implementeert het stappenplan van een enkele use case. Deze tasks functioneren onafhankelijk van elkaar. Overdracht van informatie (zoals geselecteerde ID's) vindt derhalve plaats via een derde, onafhankelijke klasse. Ontwikkelaars die bekend zijn met het web hanteren hierbij al gauw het *Session* object (zowel in Java als in .NET) dat hiervoor bij uitstek geschikt is. Het leeft



FIGUUR 1. Referentiearchitectuur

immers net zo lang als de sessie duurt. Wanneer echter delen van de code zowel via een webapplicatie als via Windows beschikbaar zijn wordt dit lastig. Het Session object is in een Windows applicatie immers niet beschikbaar. Er zijn twee eenvoudige state managers ontwikkeld, één voor het web en één voor Windows. Deze *state managers* zijn nu verantwoordelijk voor het tijdelijk opslaan van informatie en zijn beschikbaar voor alle tasks, zoals in codevoorbeeld 1.

```
public class WebState: IState  // implementa-
tie voor het web
{
   public object GetValue(string name)
   {
      return HttpContext.Current.Session[id];
   }

   ...
}

public class WinState: IState  // implementa-
tie voor windows
{
   private Hashtable hashtable = new
Hashtable();
   public object GetValue(string name)
   {
      return hashtable[id];
   }

   ...
}
```

Codevoorbeeld 1

Beide state managers implementeren dezelfde interface *IState*. Onderstaand codevoorbeeld toont de definitie van deze interface; het *contract* waaraan elke state manager voldoet. Volgens dit contract is iedere state manager in staat informatie op te halen via *GetValue()*, opslaan via *SetValue()* en verwijderen middels *DeleteValue()*. Ook kan worden getest of bepaalde informatie al aanwezig is in de state manager, via *ContainsValue()*. Zie hiervoor codevoorbeeld 2

De beide klassen *WebState* en *WinState* geven aan dat ze de interface implementeren. De klasse *WebState* implementeert de methoden van *IState* door gebruik te maken van het Session object. De klasse *WinState* implementeert de methoden door met een hashtable te werken.

Applicaties kunnen eenvoudig gebruikmaken van een platformonafhankelijke state manager doordat een object in de applicatie valideert op welk platform de

```
public interface IState  // interface voor
state managers
{
   object GetValue(string name);
   void SetValue(string name, object value);
   void DeleteValue(string name);
   bool ContainsValue(string name);
}
```

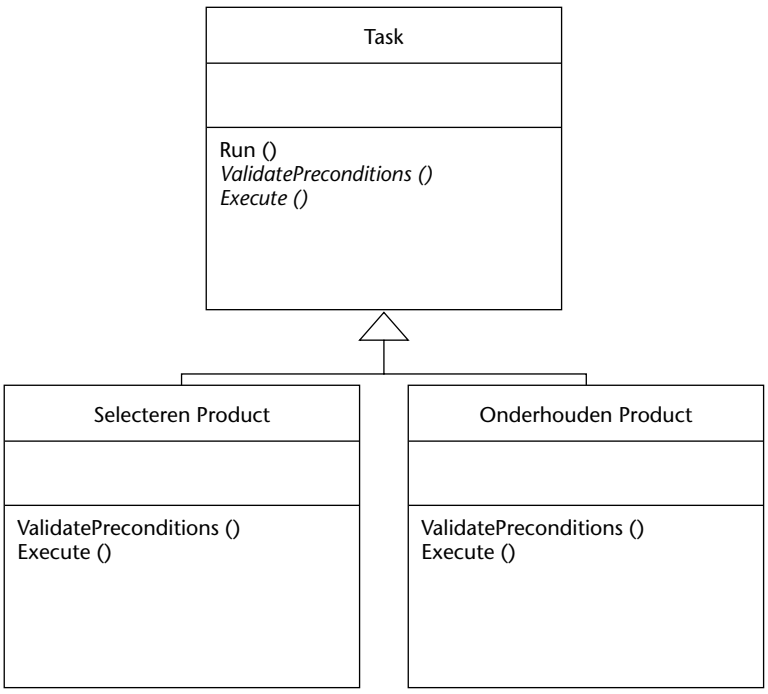
Codevoorbeeld 2

applicatie draait. Als dit Windows is - hier geldt immers *HttpContext.Current = null* - dan wordt de instantie van de klasse *WinState* geraadpleegd. In het andere geval *WebState*. Het object dat hiervoor zorg draagt wordt de *state factory* genoemd, analoog aan het design pattern *abstract factory*, zoals te zien is in codevoorbeeld 3.

ROUTING IN WEBAPPLICATIES Met dit soort pragmatische oplossingen treedt al vlug een versnelling op in projecten. Het is zaak dergelijke herbruikbare oplossingen in projecten te verzamelen. Het hanteren van een gemeenschappelijke architectuur is hiervoor een belangrijke voorwaarde. De Smart .NET Express kent teams die kleine tot middelgrote .NET-projecten snel en effectief (leren) uitvoeren, op basis van eigen oplossingen en die uit vorige projecten. Doorlopend stappen nieuwe deelnemers op, zoals projectmedewerkers, maar ook medewerkers van klanten. Zo worden ze snel en effectief voorbereid op het vervullen van rollen in Smart-projecten, zoals coach, (lead) developer of tester.

Een belangrijke uitdaging in veel webapplicaties is de samenwerking tussen de afzonderlijke webpagina's. Normaliter roept een webpagina de URL aan van een volgende te benaderen webpagina. Hiermee is een directe afhankelijkheid tussen beide pagina's gecreëerd. Deze situatie is uiteraard ongewenst. Een voorbeeld van het combineren van de referentiearchitectuur van Smart en .NET is de wijze van samenwerking tussen de tasks (en forms) in de applicatie. Een *router* zorgt hierbij voor de broodnodige onafhankelijkheid.

Een task implementeert rechtstreeks het stappenplan



FIGUUR 2. Smart's task pattern

```
public static IState GetState()
{
   ...

   if (HttpContext.Current == null)  // is
dit een webapplicatie?
   {
      winstate.GetValue(...);  // nee,
retourneer de windows state manager
   }
   else
   {
      webstate.GetValue(...);  // ja, retour-
neer de web state manager
   }

   ...
}
```

Codevoorbeeld 3

van een enkele use case. Tasks maken hiervoor regelmatig gebruik van andere tasks (en dus use cases). Een centraal object dat *Router* wordt genoemd, is verantwoordelijk voor het schakelen tussen de verschillende tasks. Een task kan aan *Router* de opdracht geven om een andere taak te starten, of om terug te gaan naar de vorige task, zonder de implementatie van deze task te kennen. Om dit mogelijk te maken erven alle tasks van een gemeenschappelijke voorouder, zoals in afbeelding 2 is weergegeven.

Hier wordt gebruik gemaakt van het template method design pattern. De methode *Run()* roept eerst de methode *ValidatePreconditions()* aan, en indien de precondities waar zijn de methode *Execute()*. Deze laatste methode implementeert het scenario van de use case. Het object Router, dat geen van de specifieke tasks kent, slechts de gemeenschappelijke voorouder, roept van de nieuwe tasks alleen de methode Run() aan. Wanneer een task zijn werkzaamheden afsluit, wordt aan de router doorgegeven hoe deze werkzaamheden zijn afgesloten, oftewel hoe de postcondities van de use case zijn waargemaakt. Deze postcondities worden door de aanroepende task gebruikt om de eigen verantwoordelijkheden te realiseren. In codevoorbeeld 4 wordt de methode *Execute()* van de task *tOnderhoudenProduct* getoond.

Wanneer de methode Execute() voor de eerste maal wordt uitgevoerd is nog geen product geselecteerd. Nu wordt door Router een nieuwe task uitgevoerd (in de tak default). Nadat de uitvoering van deze nieuwe task gereed is, wordt opnieuw de methode Execute() aangeroepen voor de klasse tOnderhoudenProduct. Als de voorlaatste task hierbij SelecterenProduct was, wordt het geselecteerde record uit de database opgehaald en

```
public class tOnderhoudenProduct
{
    public bcProduct b;
    ...
    public void Execute()
    {
        switch (Router.PreviousTask)
        {
            case Task.SelecterenProduct:
            {
                if (Router.PreviousState =
StopStates.Selected)
                {
                    b =
fProducten.Get(State.GetValue(States.Product)
                    ShowPage();
                    break;
                }
                else
                {
                    Router.EndState =
StopStates.Cancelled;
                    Router.GoBack()
                }
                default:
                {

Router.DoTask(Tasks.SelecterenProduct)
                    break;
                }
            }
        }
    }
    ...
}
```

Codevoorbeeld 4

wordt de bijbehorende webpagina getoond. Dat wil zeggen, als de voorgaande taak succesvol afgesloten is. Er geldt dan dat PreviousState gelijk is aan Selected. Daarbij valt op dat, wanneer de task SelecterenProduct negatief is beëindigd - nu is PreviousState gelijk aan Cancelled – ook het onderhouden van een product niet wordt uitgevoerd. Het object Router krijgt opdracht een stap terug te nemen. Het is deze task niet bekend welke andere task nu wordt uitgevoerd; dat weet alleen Router. Zo zijn alle tasks onafhankelijk geworden van elkaars uitvoering. Dit vergroot de mate van hergebruik in applicaties. Dit patroon wordt het task pattern genoemd.

Ecosysteem Generieke oplossingen zoals *State* en *Router* worden bij de uitvoering van projecten toegevoegd aan het Smart .NET Ecosystem, de snel groeiende verzameling componenten, gereedschappen, richtlijnen en artikelen die als basis wordt gebruikt voor het inrichten van agile .NET-ontwikkelstraten. Nieuwe onderdelen voor dit Ecosystem worden vrijwel wekelijks aangedragen door projecten en de Smart .NET Express. Belangrijk uitgangspunt voor dit Ecosystem is dat het organisaties niet in een keurslijf dwingt, maar veel ruimte laat voor creativiteit en eigen initiatief. De gemeenschappelijke referentiearchitectuur zorgt voor de inpasbaarheid van nieuwe componenten. Teams in de Smart .NET Express kunnen zo snel en accuraat use cases realiseren, mits ze beschikken over een ervaren (Smart) coach, een ruimte en enkele computers.

Creativiteit Wanneer organisaties op zoek gaan naar potentieel hergebruik over projecten in eenzelfde ontwikkelomgeving is het dan ook van harte aan te bevelen om in kort tijdsbestek zo'n referentiearchitectuur te definiëren. Daarbij is de hierboven genoemde ruimte voor eigen creativiteit van enorm belang, omdat ontwikkelaars nu kunnen participeren in het invullen van het Ecosysteem van de organisatie zonder dat de ontwikkelaars hierbij belemmerd worden door te veel voorschriften en regels. Sowieso is het schier onmogelijk om van tevoren te definiëren wat men in projecten zal tegenkomen. Er geldt dat er geen one-size-fits-all is, zowel in de uitvoering van projecten als het verzamelen van herbruikbare componenten.

> Wanneer een task zijn werkzaamheden afsluit, wordt aan de router doorgegeven hoe de postcondities zijn waargemaakt

Referenties

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, "*Design patterns*", Addison Wesley 1995.

Sander Hoogendoorn, "*Use case drempelvrees*", maandblad Informatie, mei 2001.

Sander Hoogendoorn, "*Pragmatisch modelleren met UML*", Addison Wesley 2003.

Sander Hoogendoorn, Partner Ordina
(e-mail: sander.hoogendoorn@ordina.nl)
Jurgen Appelo, Senior Technical Consultant Ordina Public
(e-mail: jurgen.appelo@ordina.nl)