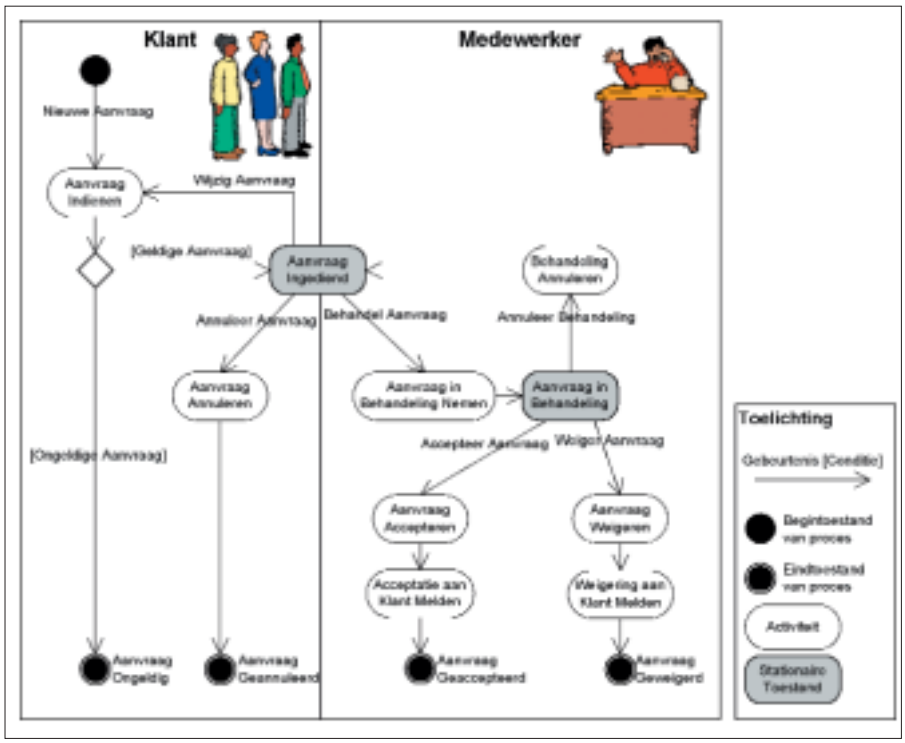
Gerke Geurts, Edward Bakker en Adrie Geelhoed
zijn werkzaam als softwarearchitect bij CMG Nederland.

Procescomponenten

Een verdeel- en heersstrategie voor de businesslaag

Een goede applicatiearchitectuur stelt een organisatie in staat haar processen te veranderen zonder ingrijpende softwareaanpassingen. Bij een applicatiearchitectuur die volgens Microsoft-richtlijnen in drie lagen is opgedeeld (presentatie, business logica en data access), kan dit worden bereikt door binnen de businesslaag onderscheid te maken tussen componenten die activiteiten aansturen (proces- of workflowcomponenten) en componenten die de activiteiten daadwerkelijk uitvoeren (businesscomponenten). In dit artikel leggen de auteurs allereerst uit waarom je de businesslaag van een applicatie zou opsplitsen in procescomponenten en businesscomponenten. Vervolgens beschrijven ze hoe je procescomponenten kunt modelleren en bouwen met behulp van Visual Studio .NET.

De businesslaag ondersteunt de uitvoering van bedrijfsprocessen. Een bedrijfsproces bestaat uit een geordende verzameling activiteiten met begin, einde en duidelijke gedefinieerde inputs en outputs. Het levert een resultaat met waarde voor interne of externe klanten. Het verloop van een bedrijfsproces is afhankelijk van gebeurtenissen in de buitenwereld of van andere processen. Een procescomponent draagt zorg voor de uitvoering van een geautomatiseerd (deel van) een bedrijfsproces.



Afbeelding 1. Voorbeeld bedrijfsproces voor beëindigen effectenrekening

Een procescomponent lijkt op een LEGO® robotcontroller. Zoals de robotcontroller informatie verzamelt van sensoren en vervolgens commando's stuurt naar actuatoren (motoren en schakelaars, enzovoort), verzamelt een procescomponent informatie van businesscomponenten en stuurt hij commando's terug naar deze componenten. De robot heeft een flexibele architectuur met een hoge mate van hergebruik: voor een verandering in het gedrag van de robot is de herprogrammering van de controller voldoende en zijn geen aanpassingen nodig van de sensoren en actuatoren. Procescomponenten bieden binnen de businesslaag van een 3-lagenarchitectuur een vergelijkbare flexibiliteit en hergebruik van businesslogica. Die flexibiliteit krijg je alleen wanneer de businesscomponenten zelf zo min mogelijk beslissingen nemen. Een robotmotor die

zijn eigen gang gaat, maakt het een willekeurige controller onmogelijk om het gewenste gedrag te leveren.

De scheiding van de businesslaag in procescomponenten en businesscomponenten schept de voorwaarden om sneller in te spelen op veranderingen in de business, door:

- Gevolgen van veranderende of nieuwe bedrijfsprocessen te beperken tot het aanpassen of maken van procescomponenten.
- Grote herbruikbaarheid van businesscomponenten binnen bestaande en toekomstige procescomponenten. Dit geldt zowel voor businesscomponenten binnen als buiten de organisatie, bijvoorbeeld in de vorm van webservices.

De uitvoering van bedrijfsprocessen door procescomponenten biedt naast flexibiliteit andere grote voordelen:

- Inzicht in de huidige toestand van bedrijfsprocessen.
- Inzicht in de doorlooptijd en bottlenecks van bedrijfsprocessen, dat het fundament legt voor continue verbetering van deze processen.

Procescomponenten en businesscomponenten

Een applicatiearchitectuur volgens Microsoft-richtlijnen deelt een applicatie op in een presentatielaag, een businesslaag en een data-acceslaag. De voornaamste verantwoordelijkheid van de presentatielaag en de data-acceslaag is het beschermen van de applicatie tegen technologische veranderingen. De presentatielaag isoleert de rest van de applicatie van de protocollen en berichtformaten die nodig zijn voor communicatie met eindgebruikers of andere applicaties. Op vergelijkbare wijze biedt de data-acceslaag aan de rest van de applicatie een consistente interface voor het opslaan en opvragen van gegevens.

In tegenstelling tot de andere lagen heeft de businesslaag geen technisch, maar een functioneel karakter. De businesscomponenten in de businesslaag ondersteunen een bedrijfsproces door het beheren en genereren van bedrijfsinformatie ten behoeve van de activiteiten binnen het proces:

- Ophalen van bedrijfsgegevens.
- Uitvoeren van complexe berekeningen.
- Opslaan van gegevens die voortkomen uit activiteiten.

De procescomponenten dragen zorg voor de aansturing van het bedrijfsproces:

- Uitvoeren van complete activiteiten.
- Bepalen van de volgorde van de uitvoering van activiteiten.
- Monitoren van de uitvoering van activiteiten.

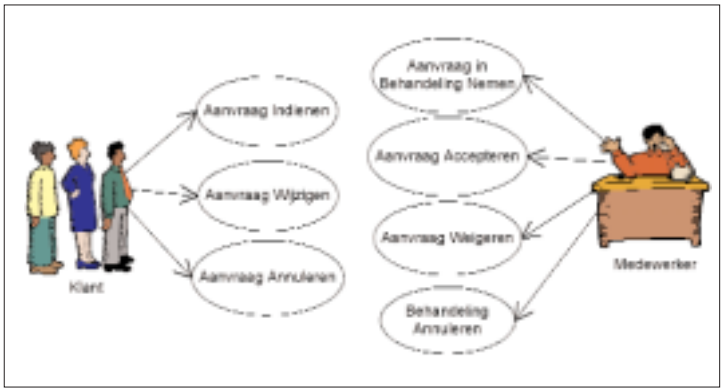
Om te illustreren hoe je de businesslaag kunt opdelen in proces- en businesscomponenten zullen we gebruik maken van een voorbeeldproces. De NEB (Nederlandse Effecten Broker) heeft, toen dotcoms nog floreerden en de beursindices tot in de hemel groeiden, een groot aantal particuliere beleggers aangetrokken door te stunten met lage tarieven. Het beursklimaat is echter omgeslagen en menig teleurgestelde, ja zelfs gedupeerde NEB-klant wil zijn/haar effectenrekening sluiten. Hoog tijd dus om NEB's businessproces voor het beëindigen van een effectenrekening onder de loep te nemen!

Een NEB-klant kan een aanvraag indienen om zijn effectenrekening te sluiten. Als de aanvraag fouten bevat, eindigt het proces vroegtijdig. Correcte aanvragen worden opgeslagen met de status "Aanvraag Ingediend" totdat een medewerker de aanvraag in behandeling neemt. Een aanvraag die nog niet in behandeling is genomen, kan altijd worden gewijzigd door een klant. Een medewerker kan een in behandeling zijnde aanvraag accepteren of weigeren. Eventueel kan de medewerker de behandeling staken wanneer hij/zij nog geen beslissing kan nemen over de aanvraag. In dat geval krijgt de aanvraag opnieuw de status "Aanvraag Ingediend".

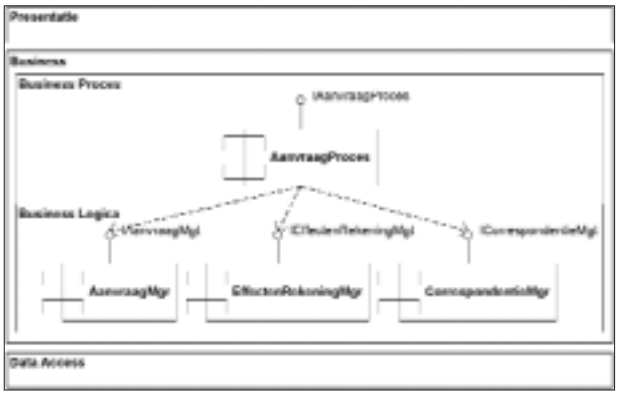
Procesanalyse

We moeten weten hoe een bedrijfsproces in elkaar steekt, voordat we het kunnen implementeren in code. We zullen gebruik maken van Visio 2002 (onderdeel van Visual Studio .NET Enterprise Architect) om ons voorbeeldproces te modelleren in UML. Afbeelding 1 toont een UML activity-diagram dat de volgende aspecten beschrijft van het voorbeeldproces:

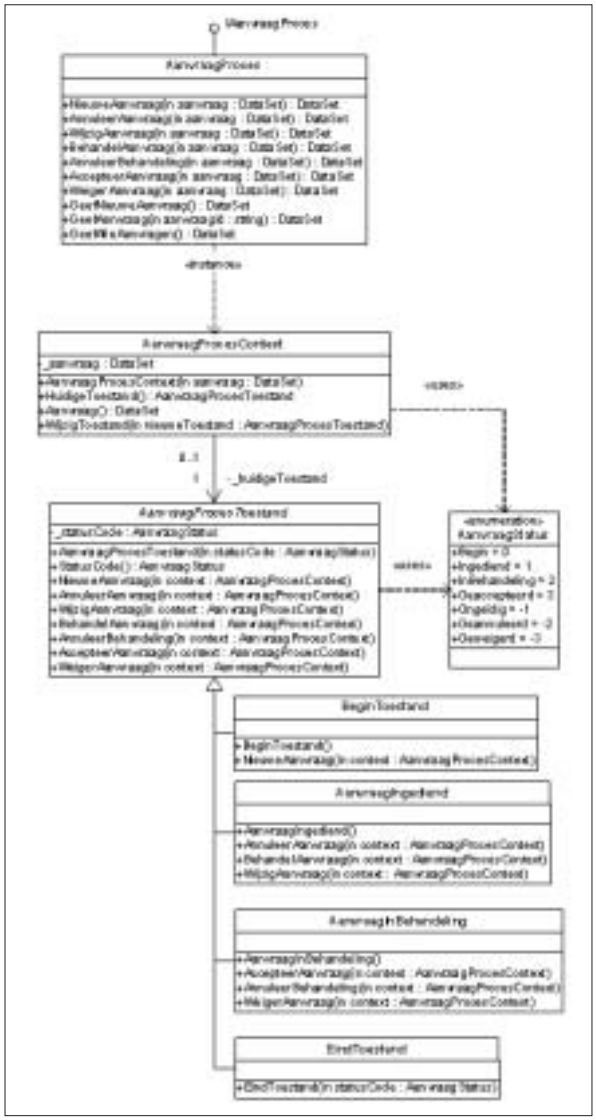
- *Stationaire toestanden* waarin het proces kan verkeren. In een stationaire toestand wacht het proces op een gebeurtenis, zoals een opdracht van de gebruikersinterface of het verstrijken van een timer.
- *Gebeurtenissen* die een proces ondersteunen in een gegeven stationaire situatie.
- *Activiteiten* die een proces uitvoeren in reactie op een gebeurtenis. Een activiteit bestaat uit één of meer aanroepen van businesscomponenten.
- *Beslissingen* die een proces nemen over de volgende uit te voeren stappen.
- *Actoren* die kunnen deelnemen aan het proces.
- *Autorisaties* van actoren voor het verrichten van activiteiten.



Afbeelding 2. Use case model voor beëindiging effectenrekening



Afbeelding 3. Voorbeeld applicatiearchitectuur



Afbeelding 4. Ontwerp van procescomponent met toepassing van het State design pattern

Het activity-diagram geeft een overzicht van de activiteiten binnen het proces en de volgorde waarin activiteiten kunnen worden uitgevoerd. Wat nog ontbreekt, is een inhoudelijke beschrijving van deze activiteiten. Use cases zijn hiervoor goed geschikt, waarbij meer benaderingen mogelijk zijn. Je kunt één use case schrijven per activiteit of één use case per gebeurtenis die kan voorkomen als het proces in stationaire toestand verkeert. Onze voorkeur gaat uit naar de laatstgenoemde aanpak. Een use case beschrijft dan doorgaans alle activiteiten die één actor binnen één proces, in reactie op één gebeurtenis, uitvoert om het proces van een stationaire toestand over te brengen naar een nieuwe stationaire toestand (of eindtoestand). Deze aanpak komt overeen met 'normaal' gebruik van use cases voor de beschrijving van de interactie tussen een systeem en één actor op één plaats en één tijdstip. Afbeelding 2 toont de use cases die relevant zijn voor het voorbeeldproces. De uitwerking van deze use cases valt buiten het kader van dit artikel.

```
public class AanvraagProces : MarshalByRefObject, IAanvraagProces {
   #region Actie methodes
   public DataSet NieuweAanvraag(DataSet aanvraag) {
     AanvraagProcesContext context = new AanvraagProcesContext(aanvraag);
       context.HuidigeToestand.NieuweAanvraag(context);
       return context.Aanvraag;
    }

    /* Overige actie methodes weggelaten... */
    #endregion

    #region Query methodes
    public DataSet GeefNieuweAanvraag() {
        IAanvraagMgt aanvraagMgr = (IAanvraagMgt) new AanvraagMgr();
        return aanvraagMgr.GeefNieuweAanvraag();
    }

    /* Overige query methodes weggelaten... */
    #endregion
}
```

Afbeelding 5. Code AanvraagProces

Ontwerp procescomponent

De NEB besluit om voor het bedrijfsproces "Beëindiging effectenrekeningen" een .NET-applicatie te ontwikkelen met een 3-lagenarchitectuur (zie afbeelding 3). De businesslaag bestaat uit één procescomponent die gebruik maakt van diverse businesscomponenten, waarvan de analyse en het ontwerp buiten het kader van dit artikel vallen.

De AanvraagProces-component implementeert het bedrijfsproces voor het beëindigen van effectenrekeningen. De interface van deze component bevat methodes voor alle gebeurtenissen die in het activity-diagram voor het bedrijfsproces zijn gemodelleerd. Deze methodes hebben dezelfde naam als de gebeurtenis en implementeren de activiteiten die moeten worden uitgevoerd als reactie op de gebeurtenis. Behalve methodes die reageren op gebeurtenissen, bevat de interface ook methodes voor het opvragen van gegevens, voor zover nodig voor het realiseren van de use cases. De procescomponent delegeert aanroepen van methodes voor het opvragen van gegevens direct naar de businesscomponenten die de gevraagde gegevens beheren. De interface van de procescomponent fungeert dus als een façade die de presentatielaag van alle benodigde businessfunctionaliteit voorziet. Het gedrag van de methodes die reageren op gebeurtenissen is afhankelijk van de huidige toestand van het bedrijfsproces. Het is mogelijk om iedere methode te programmeren als een geneste if..then of select..case structuur, maar deze aanpak levert al snel spaghetticode op. Aangezien het .NET Framework en alle Microsoft .NET-talen objectoriëntatie ondersteunen, kunnen we het objectgeoriënteerde 'State' design-pattern toepassen als een alternatief met meer perspectief op onderhoudbare code.

```
internal class AanvraagProcesContext {
    private DataSet _aanvraag;
    private AanvraagProcesToestand _huidigeToestand;

    public AanvraagProcesToestand HuidigeToestand { get { return
_huidigeToestand; }}
    public DataSet Aanvraag { get { return _aanvraag; }}

    public AanvraagProcesContext(DataSet aanvraag): base() {
        AanvraagStatus statusCode = (AanvraagStatus)
aanvraag.Tables["Aanvraag"].Rows[0]["Status"];
        _huidigeToestand = GeefToestandVoorStatusCode(statusCode);
        _aanvraag = aanvraag;
    }
    public void WijzigToestand(AanvraagProcesToestand nieuweToestand) {
        AanvraagStatus statusCode = nieuweToestand.StatusCode;
        IAanvraagMgt aanvraagMgr = (IAanvraagMgt) new AanvraagMgr();

        _aanvraag.Tables["Aanvraag"].Rows[0]["Status"] = statusCode;
        _huidigeToestand = nieuweToestand;
        if (nieuweToestand is EindToestand)
            _aanvraag = aanvraagMgr.VerwijderAanvraag(_aanvraag);
        else
            _aanvraag = aanvraagMgr.BewaarAanvraag(_aanvraag);
    }
    private AanvraagProcesToestand GeefToestandVoorStatusCode
(AanvraagStatus statusCode) {
        switch (statusCode) {
            case AanvraagStatus.Begin: return new BeginToestand();
            case AanvraagStatus.Ingediend: return new Aanvraag
Ingediend();
            case AanvraagStatus.InBehandeling: return new Aanvraag
InBehandeling();
            default: return new EindToestand(statusCode);
        }
    }
}
```

Afbeelding 6. AanvraagProcesContext class

```
internal abstract class AanvraagProcesToestand {
    private AanvraagStatus _statusCode;

    public AanvraagProcesToestand(AanvraagStatus statusCode): base() {
        _statusCode = statusCode;
    }
    public AanvraagStatus StatusCode {
        get { return _statusCode; }
    }
    public virtual void NieuweAanvraag(AanvraagProcesContext context) {
        throw new System.InvalidOperationException();
    }

    /* overige actie methodes weggelaten... */
}
```

Afbeelding 7. Code AanvraagProcesToestand

Het 'State' design-pattern introduceert in het ontwerp van de AanvraagProces-component een abstracte AanvraagProcesToestand-class met publieke methodes voor alle gebeurtenissen in het bedrijfsproces (zie afbeelding 4). Voor iedere stationaire toestand in het bedrijfsproces wordt het toestandafhankelijke gedrag geïmplementeerd in een class die overerft van deze abstracte class. Gedurende de uitvoering van een actiemethode beheert de AanvraagProcessContext-class alle gegevens van een specifiek aanvraagproces, namelijk de huidige procestoestand en de aanvraaggegevens.

Implementatie proces-component

De publieke interface van de AanvraagProces-component bestaat uit de IAanvraagProces-interface die wordt geïmplementeerd door de AanvraagProces-class (zie afbeelding 5). Alle actiemethodes van AanvraagProces hebben dezelfde structuur:

```
    public DataSet Nieuwe
Aanvraag(DataSet aanvraag) {
        AanvraagProcesContext
 context = new AanvraagProces
Context(aanvraag);
        context.HuidigeToe
stand.NieuweAanvraag(context);
        return context.Aanvraag;
    }
```

Een actiemethode instantieert een AanvraagProcesContext-class, delegeert vervolgens de aanroep naar de toestand-class die de huidige procestoestand representeert en retourneert ten slotte de bijgewerkte procesgegevens. De uitkomst van query-methodes van AanvraagProces zijn niet toestandafhankelijk en delegeren aanroepen direct naar een geschikte businesscomponent. De AanvraagProcesContext-class (zie afbeelding 6) beheert de huidige toestand van een bedrijfsproces. In de constructor initialiseert deze class op basis van een statuscode in de procesgegevens de juiste toestand-class.

De WijzigToestand-methode stelt toestand-classes in staat een toestandverandering te bewerkstelligen:

```
    public void
WijzigToestand(AanvraagProce
sToestand nieuweToestand) {
        AanvraagStatus statusCo
de = nieuweToestand.StatusCode;
        IAanvraagMgt aanvraagMgr
 = (IAanvraagMgt) new Aan
vraagMgr();

        _aanvraag.Tables["Aan
vraag"].Rows[0]["Status"] =
statusCode;
        _huidigeToestand =
nieuweToestand;
        if (nieuweToestand is
EindToestand)
            _aanvraag =
aanvraagMgr.VerwijderAanvraag
(_aanvraag);
        else
            _aanvraag = aan
vraagMgr.BewaarAanvraag(_aan
vraag);
    }
```

Deze methode is ook verantwoordelijk voor het persisteren van procesgegevens wanneer de nieuwe status geen eindstatus is, of voor het verwijderen van de procesgegevens wanneer het proces een eindstatus heeft bereikt. Alle toestand-classes overerven van de abstracte base class ProcesToestandClass (zie afbeelding 7). Het standaardgedrag van actiemethodes in deze class bestaat uit het genereren van een exceptie, die aangeeft dat de actie niet is toegestaan. Een afgeleide toestand-class herimplementeert alleen die actiemethodes die zijn toegestaan in een de procestoestand waarvoor de class het gedrag implementeert (zie afbeelding 8).

Opdelen heeft voordelen

Het opdelen van de businesslaag in proces- en businesscomponenten heeft grote voordelen. We hebben gezien hoe je procescomponenten kunt implementeren met behulp van het 'State' design-pattern. Daarnaast bestaan andere mogelijkheden om procescomponenten te implementeren, bijvoorbeeld Microsoft .NET serverproducten als BizTalk Orchestration en Exchange Workflow. We hebben in dit artikel echter laten zien, dat het .NET Framework en Visual Studio .NET uitstekende mogelijkheden bieden voor zowel procesanalyse als het ontwerp en de bouw van procescomponenten. Hoewel we geen aandacht hebben besteed aan belangrijke aspecten als autorisatie en transactiemanagement, biedt het .NET Framework ook in dit opzicht goede ondersteuning.

```
internal class AanvraagInBehandeling: AanvraagProcesToestand {
    public AanvraagInBehandeling(): base(AanvraagStatus.InBehandeling) {}
    public override void AccepteerAanvraag(AanvraagProcesContext context) {
        string rekeningId = (string) context.Aanvraag.Tables["
Aanvraag"].Rows[0]["RekeningNo"];
        IEffectenRekeningMgt rekeningMgt = (IEffectenRekeningMgt)
 new EffectenRekeningMgr();
        rekeningMgt.BeeindigRekening(rekeningId);
        ICorrespondentieMgt correspondentieMgt = (ICorresponden
tieMgt) new CorrespondentieMgr();
        correspondentieMgt.MeldAanvraagAcceptatie(context.Aanvraag);
        context.WijzigToestand(new EindToestand(AanvraagStatus.
Geaccepteerd));
    }
    public override void AnnuleerBehandeling(AanvraagProcesContext
context) {
        context.WijzigToestand(new AanvraagIngediend());
    }
    public override void WeigerAanvraag(AanvraagProcesContext context) {
        ICorrespondentieMgt correspondentieMgt = (ICorrespondentie
Mgt) new CorrespondentieMgr();
        correspondentieMgt.MeldAanvraagWeigering(context.Aanvraag);
       context.WijzigToestand(new EindToestand(AanvraagStatus.Geweigerd));
    }
}
```

Afbeelding 8. Code AanvraagInBehandeling

Nuttige internetadressen

- http://msdn.microsoft.com/architecture
- http://msdn.microsoft.com/library/en-us/dnea/html/EaAppConProcesses.asp
- http://msdn.microsoft.com/msdnmag/issues/01/07/patterns/patterns.asp
- http://www.umlcomponents.com