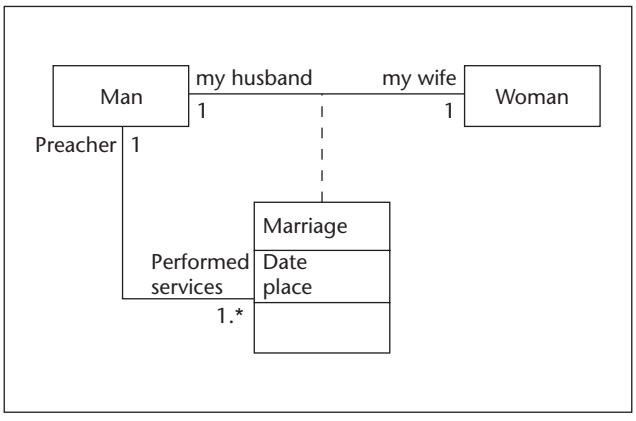
In het vorige artikel in deze serie hebben we het concept en de toepassing van associaties in UML uiteengezet. Dit artikel neemt die kennis een stap verder en verklaart de UML-associatieklasse en de implementatie daarvan. De implementatievoorbeelden worden opnieuw geschreven in Java, maar ze zijn gemakkelijk te vertalen in een andere programmeertaal.

De kracht van associaties

Associatieklassen in UML

De codevoorbeelden in dit artikel werden allemaal volledig gegenereerd door het UML/OCL-tool Octopus, die kan worden gedownload van http://www.klasse.nl/octopus/index.html. De codevoorbeelden zelf zijn ook beschikbaar via de website van dit blad (www.release.nl).

BETEKENIS VAN ASSOCIATIEKLASSEN Hoewel uit het vorige artikel al is gebleken dat het implementeren van een gewone associatie al behoorlijk gecompliceerd is, heeft UML nog meer voor ons in petto in de vorm van een associatieklasse. De vorige keer vergeleken we een normale associatie, zonder een daaraan verbonden associatieklasse, met een huwelijk. Ieder paar objecten dat wordt gelinkt via een associatie is 'getrouwd'. Deze vergelijking kan ook worden toegepast op een associatieklasse. We gaan dus het concept van associaties verkennen met gebruik van het model dat getoond wordt in Figuur 1.



FIGUUR 1. Monogaam huwelijk als associatieklasse.

Het model laat zien dat de *Man* instanties getrouwd kunnen zijn met de *Woman* instanties, en deze relatie is een instantie van een andere klasse genaamd *Marriage*. Dit is wat een associatieklasse echt betekent: het is een upgrade van een normale associatie naar een klasse die kan worden geïnstantieerd. De relatie zelf kan worden behandeld als een object.

Een reden voor het upgraden van de associatie naar een associatieklasse zou kunnen zijn dat je wenst de informatie te behouden voor de associatie zelf maar niet voor één van beide partners. Wanneer je bijvoorbeeld de datum en plaats van het huwelijk wilt vastleggen, zijn noch de *Man* noch de *Woman* klasse een goede plek om die informatie op te slaan. Deze gegevens kunnen het beste worden opgeslagen als attributen in de *Marriage* klasse. Iedere instantie van deze klasse heeft dan zijn eigen waarden voor deze attributen.

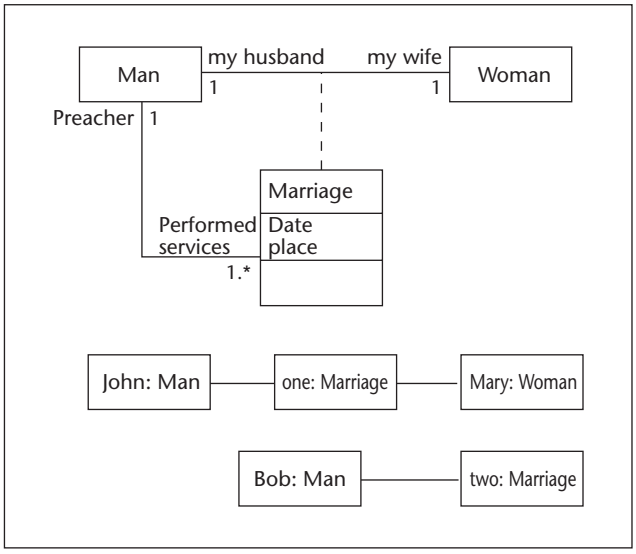
De associatieklasse is een complete klasse. Naast het feit dat de associatieklasse kan worden geïnstantieerd, kan deze zijn eigen operaties, attributen, en associaties bezitten. De *Marriage*-klasse bijvoorbeeld kan een associatie hebben met de kerk waarin de ceremonie wordt gehouden, of met de persoon die huwelijksdienst leidt. Figuur 1 toont het laatste. Het model voorziet niet in een gender-neutrale *Person* klasse, dus hebben we arbitrair gekozen om van deze functionaris een man te maken.

ABACUS-REGELS VOOR ASSOCIATIEKLASSEN In ons vorige artikel hebben we een aantal karakteristieken van gewone associaties uiteengezet. Deze karakteristieken noemen we de ABACUS-regels. Deze regels houden allemaal stand wanneer een associatie wordt

ge-upgrade naar een associatieklasse. We zullen de regels kort doornemen om te zien hoe de toepassing ervan verschilt van die van normale associaties.

- *Awareness*: Beide objecten zijn zich bewust dat de relatie tussen hen bestaat. Ze weten van elkaars bestaan, maar ze kennen ook de associatieklasse-instantie die hun relatie representeert.
- *Boolean existence*: Als de partners overeenkomen de relatie te beëindigen, wordt deze volledig ontbonden. Wanneer het een associatieklasse betreft, betekent dit dat óf de associatieklasse instantie bestaat en referenties bevat naar beide partners, óf dat de instantie in het geheel niet bestaat. Het is niet mogelijk dat een associatieklasse instantie een referentie bevat naar slechts één van de partners.
- *Agreement*: Beide objecten moeten instemmen met de relatie, net als bij de gewone associatie.
- *Cascaded deletion*: Wanneer één van de partners wordt verwijderd, wordt de link eveneens ontbonden. Bij gebruik van een associatieklasse betekent dit dat de instantie daarvan met al het andere wordt verwijderd.
- *USe of role names*: Een object zou aan zijn partner kunnen refereren door de rolnaam van de associatie te gebruiken: 'my husband' of 'my wife'. Maar het object mag ook refereren aan de instantie die de relatie vertegenwoordigt. Omdat de UML je niet in staat stelt om je eigen namen daarvoor te definiëren, wordt de naam van de associatieklasse voor dit doel gebruikt.

EEN OP EEN LINKS Een belangrijk gegeven is dat een instantie van een associatieklasse altijd één instantie van de klasse aan het ene einde verbindt met één instantie van de klasse aan het andere einde. Ongeacht de multipliciteiten aan beide uiteinden, vertegenwoordigt een associatieklasse instantie een één-op-één link (zie Figuur 2).



FIGUUR 2. Polygaam huwlijk

In Figuur 2 is Mary getrouwd met zowel John als Bob. Beide huwelijken zouden op verschillende dagen, op verschillende plaatsen en met verschillende predikanten kunnen plaatsvinden. Daarom moeten er twee instanties van de associatieklassen zijn die de twee huwelijken vertegenwoordigen, ieder met zijn eigen waarden voor de attributen en associaties. In het geval een van de twee huwelijken zou worden ontbonden, zou alleen de instantie die het mislukte huwelijk vertegenwoordigt, moeten worden verwijderd.

De meeste associatieklasse-instanties kunnen uniek geïdentificeerd worden door de instanties van de twee klassen die zij kunnen verbinden. Het *one* huwelijk in het voorbeeld is de unieke link tussen John en Mary, en het *two* huwelijk heeft specifiek betrekking op Bob en Mary. Er kunnen niet twee instanties van *Marriage* zijn die dezelfde twee objecten verbinden. Een uitzondering daarop is de situatie waarin beide uiteinden worden gemarkeerd als <bag> of <sequence>.

Dit betekent dat het type van de associatie-einde een verzameling is die hetzelfde element meer dan één keer zou mogen vasthouden. In dat geval zou er meer dan één instantie van *Marriage* kunnen zijn die John en Mary verbindt. Als je in dit geval specifieke associatieklasse instanties zou moeten identificeren, moet je een attribuut toevoegen aan de *Marriage* klasse die een unieke waarde heeft voor alle *Marriage* instanties. Als er geen expliciete noodzaak is voor unieke identificatie, zal een normale object-identiteit volstaan.

IMPLEMENTATIE ASSOCIATIEKLASSEN Nu we weten wat een associatieklasse betekent en hoe deze instantie wordt gekarakteriseerd, kunnen we kijken naar hoe deze wordt geïmplementeerd. Centraal in de implementatie is het gegeven dat je de associatie waaraan een associatieklasse verbonden is op dezelfde manier wilt gebruiken als een normale associatie. Zo wil je in de één op veel of veel op veel situatie een echtgenoot of een *Woman*-instantie kunnen toevoegen door het aanroepen van de operatie *addToHusbands*, terwijl je in de een op een situatie de operatie *setHusband* zou willen gebruiken. Het toevoegen van een echtgenoot aan een *Woman*-instantie impliceert het aanmaken van een instantie van de associatieklasse *Marriage*. Onze implementatie verzekert dat deze instanties alleen worden aangemaakt wanneer één van de *addToHusbands* of *setHusband* operaties wordt aangeroepen. Het creëren van een instantie van een associatieklasse op een andere manier zou inconsistenties in het systeem veroorzaken.

In het vorige artikel onderscheidden we vier configuraties of associatie-uiteinden: één op één, één op veel,

veel op veel en de eenzijdig navigeerbare associatie. De eerste drie configuraties komen ook voor in combinatie met een associatieklasse. Eenzijdig navigeerbare associaties zijn niet nuttig wanneer daaraan een associatieklasse is verbonden, daarom zullen we deze configuratie niet bespreken.

ASSOCIATIEKLASSE De implementatie van de associatieklasse zelf is voor alle drie configuraties hetzelfde. De implementatie zou twee velden moeten bevatten die pointers zijn naar de twee objecten die verbonden zijn via de associatieklasse-instantie. Het type van het eerste is de klasse aan het ene uiteinde en het type van het andere veld is de klasse aan het andere uiteinde. Ieder veld zou een normale get-operatie moeten kunnen uitvoeren, maar er zou geen set-operatie moeten zijn. De velden krijgen hun waarde bij het aanmaken van de associatieklasse instantie, zoals blijkt uit de volgende code:

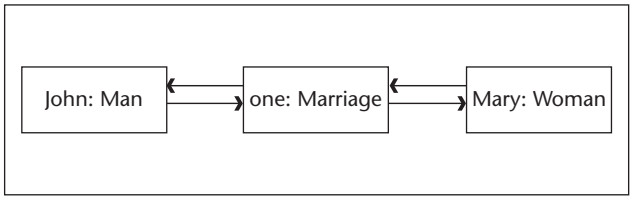
```
public class Marriage {
      ...
      private Man f_husband = null;
      private Woman f_wife = null;

      public Marriage(Man a, Woman b) {
            if ( a != null && b != null ) {
// can't relate an object to null
               this.f_husband = a;
               a.z_internalAddToMarriage(this);
               this.f_wife = b;
               b.z_internalAddToMarriage(this);
            }
      }

      public Man getMyHusband() {
            return f_myHusband;
      }

      public Woman getMyWife() {
            return f_myWife;
      }
      ...
}
```

De constructor van de *Marriage*-klasse zorgt voor het opzetten van de link tussen de twee partners door het aanroepen van de speciale *z_internalAddToMarriage* operaties van beide. Deze operatie zou alleen moeten worden aangeroepen vanaf de constructor van de *Marriage*-klasse, wat z'n obscure naam verklaart. Intern wordt de link tussen de twee partners geïmplementeerd als een van het *Woman*-object naar het *Marriage*-object, die op zijn beurt een link vasthoudt naar het *Man*-object, en vice versa. In Figuur 3, die de implementatie toont, vertegenwoordigen de pijlen de pointers tussen de instanties.



FIGUUR 3. Implementatie van de één op één associatie klasse.

PARTNERKLASSEN IN ÉÉN OP ÉÉN SITUATIE Beide klassen aan de uiteinden van de associatie kunnen op vrijwel dezelfde wijze worden geïmplementeerd in de één op één associatie. Iedere klasse zou een veld moeten hebben, waarvan het type de associatieklasse moet zijn. Er moet een get-operatie voor dit veld (*getMarriage*) zijn, maar geen set-operatie. In plaats daarvan zouden de *setMyWife* en *setMyHusband* operaties moeten worden gebruikt. Dit zijn tevens de belangrijkste operaties, omdat zij het nieuwe partnerobject bewust maken van de link. Laten we eens kijken naar de code voor de *Man*-klasse:

```
public class Man {
      ...
      private Marriage f_marriage = null;

      public void setMyWife(Woman element) {
            if ( this.f_marriage != null ) {
                  // this man is already mar-
ried, remove the old marriage
             ((Marriage)this.f_marriage).clean();
             }
             if ( element != null ) {
                  if ( element.getMarriage() !=
null ) {
                        // the new wife is
already married, remove the old marriage
                        ((Marriage)element.get
                          Marriage()).clean();
                  }
                  // create the new marriage
                  this.f_marriage = new
                  Marriage(this, element);
            } else {
                  // cannot marry a null
object, so the marriage is set to null
                  this.f_marriage = null;
            }
      }
      ...
}
```

Net als in het geval van de normale associatie, check je eerst of dit *Man*-object al getrouwd is. Als dat zo is, verwijder je het oude huwelijk ten gunste van het nieuwe. Dit wordt gedaan door het aanroepen van de *clean*-operatie van de *Marriage*-klasse:

```
public void clean() {
      f_myHusband.z_internalRemoveFrom
                        Marriage(this);
      f_myHusband = null;
      f_myWife.z_internalRemoveFrom
                        Marriage(this);
      f_myWife = null;
}
```

Deze operatie verwijdert alle pijlen uit Figuur 3, die deze implementatie toont. De *z_internalRemoveFrom-Marriage* operaties zetten de pointer naar de huwelijksinstantie op de waarde null. De garbage collector voert het feitelijke verwijderen van de *Marriage* -instantie uit. Merk op dat deze implementatie volgens de ABACUS-regels verloopt; wanneer de relatie wordt ontbonden, wordt de associatieklasse-instantie eveneens ontbonden. Wanneer andere objecten aan de associatieklasse-instantie refereren, zouden deze referenties eveneens opgeruimd moeten worden, anders kan de garbage collector de huwelijksinstantie niet verwijderen. Deze situatie wordt niet afgehandeld in de getoonde code. Het verwijderen van een instantie terwijl er nog steeds referenties naar die instantie zijn is daarentegen geen goed idee. Daarom zou je moeten zorgen dat deze referenties expliciet vóór het verwijderen van de associatie worden opgeruimd.

Vervolgens onderzoeken we of het nieuwe *wife*-object al getrouwd is. Als dat zo is, wordt haar oude huwelijk verwijderd. Tenslotte kun je een nieuwe *Marriage*-instantie creëren. Dit is de enige regel die verschillend is voor de *setMyHusband* en *setMyWife* operaties. De *Marriage* constructor neemt als parameters eerst een *Man*- en daarna een *Woman*-instantie. Om die reden moet je de volgorde van de feitelijke parameters veranderen. De aanroep van de constructor in *setMyHusband* is:

```
this.f_marriage = new Marriage(element, this);
```

Iedere *Man*-instantie moet in staat zijn de rolnaam te gebruiken om te refereren aan de *Woman*-instantie waaraan deze gerelateerd is. Omdat de implementatie geen directe pointer levert voor de *Woman*-instantie, moeten we de *getMyWife* operatie implementeren met gebruik van de referentie in de *Marriage*-instantie:

```
public Woman getMyWife() {
      if ( this.f_marriage != null ) {
            return this.f_marriage.getMyWife();
      } else {
            return null;
      }
}
```

PARTNERKLASSEN IN EEN OP EEN SITUATIE Bij het implementeren van de klassen aan de uiteinden van de associatie in de één op één situatie, moet je differentiëren tussen het *many* uiteinde en het *one* uiteinde. Aan het *many* uiteinde zien de zaken er min of meer uit zoals in de één op één situatie, maar aan het *one* uiteinde ziet het er anders uit. We kijken eerst naar het many uiteinde. De *setMyWife* operatie in de *Man* klasse wordt als volgt geïmplementeerd.

```
public void setMyWife(Woman element) {
      if ( this.f_marriage != null ) {
            // this man is already mar-
ried, remove the old marriage
            ((Marriage)this.f_marriage).
                              clean();
      }
      if ( element != null ) {
            // create the new marriage
            this.f_marriage = new
            Marriage(this, element);
      } else {
            this.f_marriage = null;
      }
}
```

Nog steeds moet gecheckt worden of deze man al getrouwd is, maar je hoeft niet meer uit te zoeken of de nieuwe echtgenote getrouwd is. Zij mag meerdere keren getrouwd zijn. Aan het *many* einde verschilt de implementatie van de *Woman* klasse in grotere mate. Allereerst is het type dat refereert aan de associatieklasse niet langer *Marriage*, maar *java.util.Set*. Het bevat een verzameling van *Marriage*-instanties. Ten tweede zullen bij de implementatie van de *Woman* class, net als in het normale geval, de volgende operaties moeten worden uitgevoerd:

- een *set*-operatie met een verzameling als parameter: stelt de verzameling echtgenoten vast als de gegeven verzameling.
- een *add*-operatie met een enkel object als parameter: voegt een enkele echtgenoot toe.
- een *add*-operatie met een verzameling als parameter: voegt alle elementen in de parameter-verzameling toe aan de set echtgenoten.

Lees verder op pagina 49

Vervolg van pagina 6

- een *remove*-operatie met een enkel object als parameter: verwijdert een enkele echtgenoot.
- een *remove*-operatie met een verzameling als parameter: verwijdert alle elementen in de parameter-verzameling uit de set echtgenoten.
- een *clear*-operatie die alle geassocieerde objecten verwijdert: laat de set echtgenoten leeg achter.

Als we wat nauwkeuriger kijken naar de *add* en *remove* operaties die een enkelvoudig object als parameter pakken, kunnen we constateren dat deze nogal gecompliceerd zijn. Ze worden als volgt geïmplementeerd:

```
   public void addToMyHusbands(Man element) {
     if ( element != null ) {
       if ( element.getMarriage() != null )
{
          ((Marriage)element.getMarriage()).
                                    clean();
               }
               new Marriage(element, this);
          }
     }

   public void removeFromMyHusbands(Man
                             element) {
     if ( element != null ) {
       if ( element.getMarriage() != null )
{
          ((Marriage)element.getMarriage()).
                                    clean();
               }
          }
     }
```

In de *addToMyHusbands* operatie wordt een bestaand huwelijk van de nieuwe echtgenoot gedeletet. In de *removeFromMyHusbands* operatie wordt het bestaande huwelijk van de verwijderde echtgenoot - degene met deze *Woman* instantie - verwijderd. Je gebruikt gewoon de bestaande *clean*-operatie in *Marriage*. Blijf echter wel alert op het naleven van de ABACUS-regels.

Een andere relevante operatie is degene waarmee een *Woman*-object in staat wordt gesteld om de rolnaam *myHusbands* te gebruiken om te refereren aan de set van *Man*-instanties waarmee deze is verbonden. Dat wordt als volgt geïmplementeerd:

```
     public Set getMyHusbands() {
          Set /*(Man)*/ result = new

HashSet( /*Man*/);
     Iterator it = this.f_marriage.iterator();
        while ( it.hasNext() ) {
          Marriage elem = (Marriage)
                          it.next();
             result.add( elem.
             getMyHusbands() );
          }
          return result;
     }
```

Net als in de één op één situatie, gebruik je het *f_marriage* veld om de waarden te verkrijgen die zich in de resultaat-set zouden moeten bevinden. Dat veld is van het type *java.util.Set*. Het is daarom noodzakelijk om de elementen te analyseren en de waarde van de *getMyHusbands*-operatie te verkrijgen voor ieder element. Merk op, dat de *getMyHusbands* operatie in de *Marriage*-klasse

De velden krijgen hun waarde bij het aanmaken van de associatieklasseinstantie

een enkel *Man*-object genereert, hoewel de naam iets anders suggereert. De naam is afgeleid van de rolnaam aan het *Man* uiteinde: *myHusbands*.

Partnerklassen in veel op veel situatie

Zoals verwacht lijkt de implementatie van de klassen aan de uiteinden van de associatie in de veel op veel situatie sterk op de implementatie van de *Woman* klasse in de één op veel situatie. En vanwege de symmetrie kunnen beide klassen op dezelfde wijze geïmplementeerd worden. Beide klassen hebben een veld dat een set *Marriage*-instanties bevat, en een aantal operaties die zorgdragen voor de implementatie van het toevoegen en verwijderen van partner-instanties. We kijken nu eerst naar de *addToMyHusbands* operatie in de *Woman*-klasse.

```
     public void addToMyHusbands(Man element)
{
         boolean isPresent = false;
           // ensure that the new husband is
not allready present
         Iterator it = f_marriage.iterator();
           while ( it.hasNext() && !isPresent
) {
             Marriage elem = (Marriage)
                             it.next();

               if ( elem.getMyHusbands() ==
```

```
                        element ) {
                    isPresent = true;
                }
            }
            if ( !isPresent ) {
                f_marriage.add(new
                  Marriage(element, this));
            }
        }
```

De meeste code in deze operatie dient om er zeker van te zijn dat de verzameling echtgenoten een set is en niet een bag of sequence. Daarom analyseren we de set *Marriage*-instanties van deze vrouw. Alleen wanneer de nieuwe echtgenoot niet aanwezig is in de set echtgenoten wordt een nieuwe *Marriage*-instantie aangemaakt.

Ook hier moeten we dat dus eerst vaststellen, voordat een element uit de set echtgenoten verwijderd wordt. Daarom moeten we eveneens de set *Marriage*-instanties van deze vrouw analyseren in de *removeFromMyHusbands* operatie. Wanneer je het corresponderende element hebt gevonden, kun je het verwijderen door de *clean*-operatie in de *Marriage*-klasse te gebruiken.

```
public void removeFromMyHusbands(Man
                          element) {
    Marriage foundElem = null;
    Iterator it = f_marriage.
                    iterator();
    while ( it.hasNext() ) {
        Marriage elem = (Marriage)
                        it.next();
        if ( elem.getMyHusbands() ==
                        element ) {
            foundElem = elem;
        }
    }
    if ( foundElem != null ) {
        ((Marriage)foundElem).clean();
          this.f_marriage.
          remove(foundElem);
    }
}
```

TEN SLOTTE Het bouwen van een goede implementatie is geen triviale zaak. Er valt veel te zeggen voor het genereren van code uit een UML-model. Je hoeft je alleen maar de situatie voor te stellen waarbij de modelleerder plotseling beslist om een associatie te upgraden naar een associatieklasse. Dat zou een ramp zijn voor de programmeur; hij zou weer helemaal opnieuw moeten beginnen. Net als bij normale associaties, kunnen de associatieklassen allemaal op dezelfde manier geïmplementeerd worden. Slechts de rolnamen behoeven aanpassing. De enige conclusie luidt dan ook: er is dringend behoefte aan betere codegeneratoren!

MIND TEASERS We besluiten net als de vorige keer met een paar puzzels waardoor je associatieklassen beter kunt leren begrijpen.

- Probeer een situatie te modelleren waarbij een persoon voor een bank werkt, en zich dus in een werkgever-werknemer relatie bevindt, terwijl die persoon tegelijkertijd cliënt is van de bank. Deze persoon heeft een hypotheek bij deze bank. De bank-cliënt relatie zou dus informatie moeten bevatten over bijvoorbeeld de hoogte van de lening en de maandelijkse aflossing. De werkgever-werknemer relatie bevat informatie over bijvoorbeeld salaris, startdatum van het dienstverband en een functieomschrijving.
- Een associatieklasse wordt vaak vergeleken met een linktabel in een database. De records in de linktabel houden de sleutels van de records in de andere tabellen waarmee deze verbonden is, maar de linktabel kan tevens eigen attributen bevatten. Onderzoek eens of je het eens bent met deze vergelijking. Zijn er verschillen of niet?
- Er zijn ook mensen die vinden dat álle associaties vergelijkbaar zijn met linktabellen. Een consequentie hiervan is dat je ervoor zou kunnen kiezen om normale associaties te implementeren (zonder associatieklassen) op dezelfde wijze als een associatie mét associatieklasse. Wat zijn de voor- en nadelen van deze benadering?

Referenties

De Octopus-tool kan gedownload worden van octopus.sourceforge.net. Naast het genereren van alle code die nodig is om de associaties te implementeren, kan dit tool een user interface prototype genereren en een XML-storage faciliteit vanaf een UML-model. Dit maakt het mogelijk om de applicatie te draaien en instanties van de klassen in het model te creëren.

Jos Warmer is partner bij Ordina. Anneke Kleppe is onderzoeker aan de Universiteit Twente.