Projectbeheer en geavanceerde processen in softwareontwikkeling.

Ontwikkeling volgens MDA

Professoren: Jan Wirix

Kristof Coninx

19 mei 2012

Inhoudsopgave

Lijst van figuren

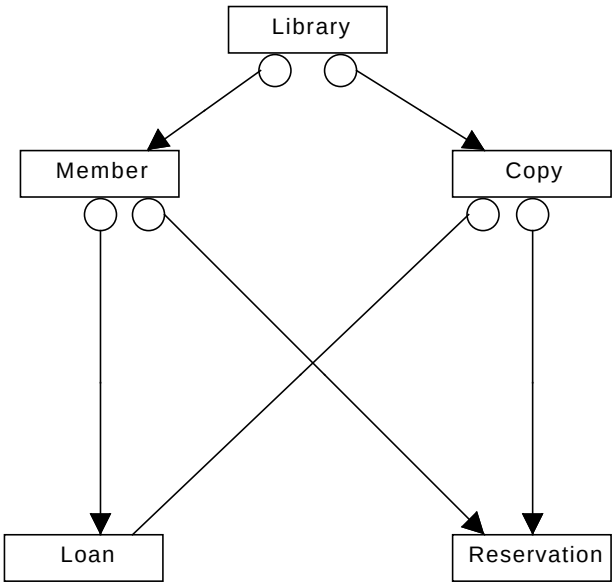
1 Merode Deliverables

In deze sectie worden de merode deliverables besproken waarvan gewerkt wordt om tot een oplossing te komen binnen het versata framework. Deze deliverables zijn voornamelijk geïnspireerd door het voorbeeld in hoofdstuk 7 van de merode documentatie. Hier en daar zijn er echter wel enkele aanpassingen en/of uitbreidingen doorgevoerd.

Zo is er ook een library object toegevoegd om de member en copy klassen in een context te plaatsen. Verder werd ook de object event tabel uitgebreid met events voor het aanmaken en vernietigen van library objecten. Tenslotte werd ook de life cycle van het copy object wat uitgebreid om een moeilijker scenario te illustreren bij de omzetting van deze modellen binnen het versata framework.

1.1 Existence Dependency Graph

Dit EDG-diagram is hetzelfde als wat er in de opgave getoond werd, met als toevoeging de library-component. Deze component zal verder ook invloed hebben op de OET zoals besproken in de 1.2.



Figuur 1: EDG for the library

1.2 Object Event Table

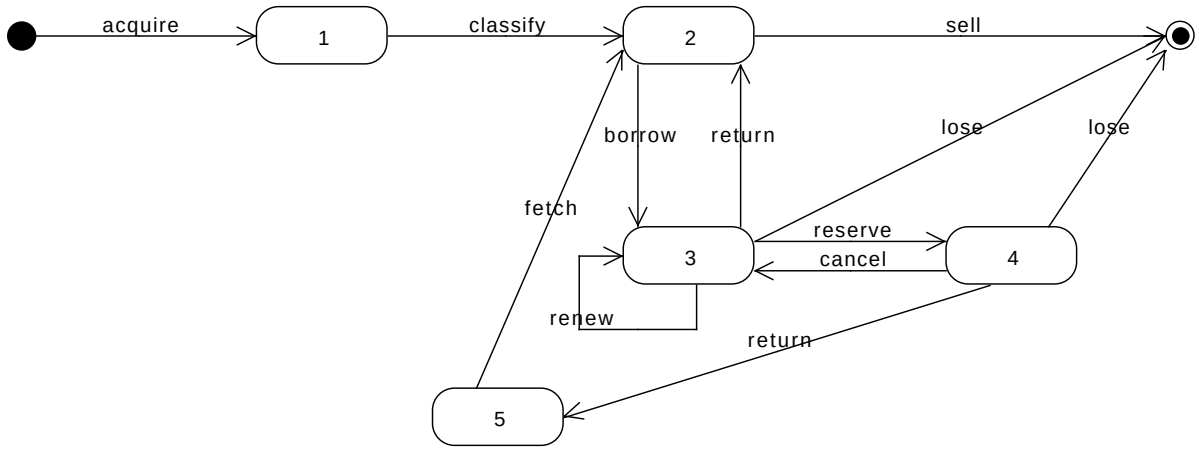
In deze sectie wordt de Object event table besproken. Per event wordt er in deze tabel weergegeven welke types van acties er worden uitgevoerd op verschillende objecten. (Deze acties zijn met de letters C, M en E letters afgekort weergegeven voor Create, modify en End.)

Tabel 1

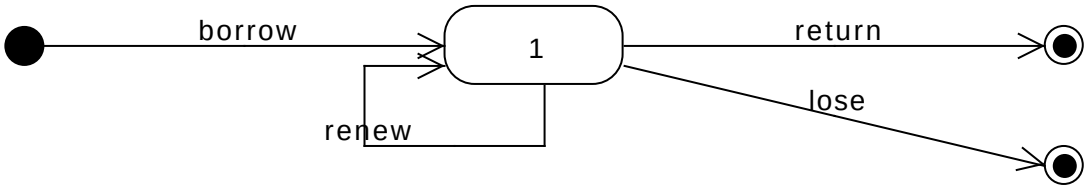
	Library	Member	Copy	Loan	Reservation
create	C				
enter	M	C			
leave	M	E			
acquire	M		C		
classify			M		
borrow		M	M	C	
renew		M	M	M	
return		M	M	E	
sell	M		E		
reserve		M	M		C
cancel		M	M		E
fetch		M	M	C	E
lose		M	E	E	
destroy	E				

1.3 Life cycles

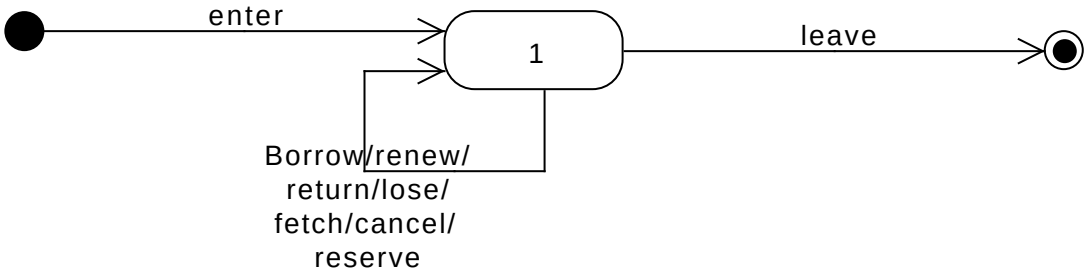
In deze sectie worden de lifecycles besproken voor de objecten in de voorgaande Merode modellen. Deze life cycles worden voorgesteld in toestandsdiagramma en bepalen de toestanden waarin een bepaald object kan geraken door middel van het afvuren van bepaalde events op deze objecten.



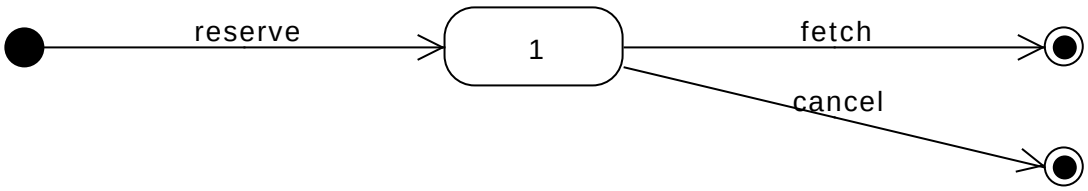
Figuur 2: Lifecycle for Copy



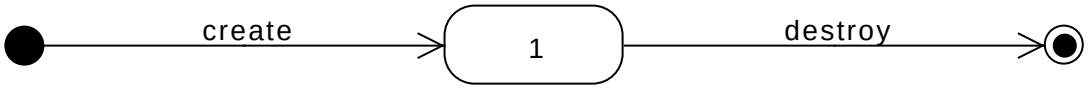
Figuur 3: Lifecycle for Loan



Figuur 4: Lifecycle for Member



Figuur 5: Lifecycle for Reservation



Figuur 6: Lifecycle for Reservation

1.4 Precondities

In deze sectie worden enkele precondities besproken waaraan bepaalde events moeten voldoen alvorens te vuren.

Allereerst worden er voor alle events constraints opgesteld gebaseerd op de lifecycles waarin hun doelobjecten voorkomen. Voor elke event wordt er een constraint bepaald voor het nagaan of het doelobject zich in de juiste toestand bevindt.

Een volgende conditie die uitgewerkt wordt als preconditie, is het feit dat een lid maar maximum 5 boeken mag ontlenen. Alsook de conditie dat een boek enkel kan uitgeleend worden wanneer het nog niet is ontleend wordt, wordt uitgedrukt als een preconditie.

In deze context wordt ook de regel toegevoegd dat een boek niet uitgeleend mag worden wanneer er nog een openstaande reservatie voor bestaat (die nog niet gefetched is).

Betreffende de bibliotheekobjecten zelf werd er bepaald dat alle boeken verkocht moeten worden, en alle leden het pand verlaten moeten hebben alvorens een bibliotheek te vernietigen.

Een boek kan ook niet verkocht worden wanneer er nog openstaande reservaties zijn voor dit boek, of wanneer dit boek nog uitgeleend is aan een lid.

Tenslotte kan een lid ook enkel de bibliotheek betreden wanneer deze bibliotheek effectief boeken heeft verworven.

Hoe deze precondities uiteindelijk worden afgedwongen wordt beschreven in sectie 2.4

1.5 Acties

In deze sectie worden bepaalde acties besproken die gepaard gaan met het afvuren van bepaalde events.

Allereerst worden hier alle acties opgesteld gebaseerd op de lifecycles waarin hun doelobjecten voorkomen. Voor elke event die een object in een andere toestand brengt, moet er als actie de regel toegevoegd worden die deze toestandsverandering uitvoert.

Verder moet er ook nog afgedwongen worden dat bij de events die een creatie-eigenschap hebben, de juiste business objecten moeten aangemaakt worden. Dit wordt verder besproken in sectie 2.5

2 Transformatie van MERODE naar VERSATA

In deze sectie wordt systematisch de mechanische omzetting van de Merode modellen uit sectie 1 in een oplossing binnen het Versata framework besproken.

2.1 EDG

De omzetting van het EDG diagram naar een oplossing binnen versata wordt besproken in sectie 3

2.2 OET

2.2.1 Business objects

Bij het omzetten van de merode modellen naar een oplossing binnen het versata framework, vertrekken we van de object-event-table. Deze tabel zal gebruikt worden om de versata objecten te construeren. Voor elke kolom van de tabel wordt er een versata business object aangemaakt. Deze objecten zullen als het ware de data containers zijn voor de domein elementen die ze voorstellen. Deze objecten worden ook voorzien van attributen die relevant kunnen zijn voor deze context. Deze attributen worden niet expliciet in de merode documentatie vermeld, maar kunnen zelf nog toegevoegd worden waar nodig.

Voor een mechanische transformatie dient er echter wel nog een soort van referentie document voorzien te worden als input voor het omzettingsproces. Dit referentiedocument zal voor elk business object de informatie velden bevatten die gegenereerd moeten worden.

Verder worden de objecten die aangemaakt worden in versata studio ook nog voorzien van:

- Primary key: Unieke primaire sleutel voor elk object.
- Foreign key: Verwijzing naar de primaire sleutels van elk object waar dit object een relatie mee heeft.
- Afleidende velden: Attributen die het aantal relaties tellen of een som maken. Deze objecten dienen te worden aangemaakt wanneer nodig in een voorheen besproken constraint.
- ObjectState: Een teller die bijhoudt in welke toestand een object zich bevindt.

2.2.2 Event objects

Voor elke rij in de OET (events) worden event object aangemaakt in versata studio. Deze objecten verschillen volgens versata niet per definitie van business objecten, maar worden in onze context wel op een andere manier gebruikt. Voor elke mogelijke event worden er dus event objecten aangemaakt. Bij deze objectevents kunnen immers constraints en acties worden gespecifieerd die de beperkingen en handelingen die gepaard gaan bij een event, voorstellen.

Een bepaald subtype van deze event objecten, namelijk de create events zorgen echter ook voor de aanmaak van hun parent-object (wanneer natuurlijk aan alle precondities is voldaan). Voor het voorzien van deze functionaliteit is het nodig dat de gegenereerde java code moet worden aangepast voor deze event objecten. Zo moet er een operatie *beforeInsert()* voorzien worden. Deze methode zal opgeroepen worden voordat de effectieve insert operatie gebeurt van het event object. In deze methode worden dan de logica voorzien om het parent object (waarvoor deze create event geldt) te creëren, te instantiëren en te voorzien van de noodzakelijke informatie. Dit laatste gebeurt door de setters in het parent object op te roepen met als parameter de waarde van het locale event object. Het is mogelijk om dit allemaal mechanisch te doen omdat de velden in het event object aangemaakt worden op basis van de velden van het parent object. De getters and setters van deze

velden komen dan echter ook overeen omdat ze door het volgen van hetzelfde process gegenereerd worden.

Verder hebben deze event objecten de volgende eigenschappen(/velden).

- Primary key: Unieke primaire sleutel voor elk event object.
- Foreign key: Voor elk object waarop deze event een handeling uitvoert.
- Attributes: Voor een create event object, een attribuut veld voor elk te isntantiëren veld in het aan te maken parent object.

Het is mogelijk om nog meer informatie bij te houden in deze event objecten zoals tijdstip van uitvoeren of de initiator van het event. Dit is echter niet strikt noodzakelijk en kan mogelijk gezien worden als administratieve keuze die gemaakt moet worden.

2.3 Life cycles

De life cycles worden in de vorm van state machine diagrams geleverd als input voor het transformatie proces. Dit is een belangrijk document voor dit process aangezien de toestandsovergangen van de business objecten hieruit afgeleid kunnen worden. Elke toestand wordt zo voorgesteld als een getal en er is een veld voorzien in de business objecten om dit getal bij te houden. Dit getal dient niet aangepast te worden door gebruikers zelf. Het aanpassen van de *objectState* gebeurt door het afvuren van events (i.e. het creëren van event objects). Deze event objecten zullen bij creatie dus de toestandsvariabelen van de parent objecten manipuleren. Wanneer en hoe dit gebeurt, wordt uitgelegd in secties 2.4 en 2.5.

2.4 Precondities

Er zijn een aantal precondities die gewoon afgeleid kunnen worden uit de geleverde merode documenten. Meer bepaald de events die gebruikt worden om een business object van toestand te doen veranderen, kunnen enkel afgevuurd worden wanneer het parent object zich in de juiste toestand bevindt. Afhankelijk van de life cycle documentatie uit sectie 2.3, kan mechanisch afgeleid worden welke toestanden geldig zijn voor een object om een bepaalde event uit te voeren. Op basis van deze informatie worden dan precondities gegenereerd voor de event objects in kwestie om af te dwingen dat zo een event enkel kan voorkomen wanneer het parent object zich in de juiste toestand bevindt.

Verder kan het zijn dat er nog enkele constraints moeten gelegd worden op business objecten. Deze constrains zijn dan echter wel context afhankelijk en zijn moeilijker automatisch te genereren zonder de nodige informatie. Om dit probleem op te lossen is het mogelijk om nog een extra referentie document te voorzien naast de gebruikelijke Merode documentatie. In dit referentiedocument kunnen dan bijvoorbeeld constraints in een meer algemene taal uitgedrukt worden voor bepaalde business objecten. Zo kan bijvoorbeeld OCL gebruikt worden als zo een algemene taal.

2.5 Acties

Net zoals in sectie 2.4 kunnen er al enkele acties afgeleid worden uit de geleverde merode documenten. Bij het overgaan van de ene toestand van een business object naar een andere, ten gevolge van een bepaalde event, dient deze manipulatie van de toestandsvariabele voorgesteld te worden door een een feitelijke actie. Deze actie kan voor elk event object geconstrueerd worden als een manipulatie van de toestandsvariabele van het parent object door middel van het gebruik van de setter van de objectState variabele. De concrete doel waarde van die variabele valt af te leiden uit het life cycle diagram.

Verder zijn er niet echt acties die afgeleid kunnen worden uit de merode documentatie. Indien er via een mechanische transformatie nog extra acties moeten gespecifieerd worden, dienen deze

voorzien te worden in een referentie document op een manier die gelijkaardig is aan de methode besproken in sectie 2.4.

3 Transformatie van EDG naar een UI

Het mechanisch transformeren van merode documentatie naar een gebruikersinterface is iets moeilijker gebleken als het transformeren van de services zelf. Merode zelf, bespreekt voornamelijk services, los van de gebruikersinterface. Om echter toch een poging te wagen is er gekozen om te vertrekken van de Existence Dependency Graph. Deze graph geeft een rudimentair beeld van de flow die ingewerkt kan worden in een gebruikersinterface. Om een mechanische transformatie te bespreken nemen we hier expliciet het versata framework, waar in voorgaande secties het bespreken van een algemene transformatie nog mogelijk was.

Om te beginnen moet er altijd een startpagina voorzien worden voor de html applicatie die de gebruikersinterface zal voorstellen. Naast deze startpagina moet er ook een FramesetPage voorzien worden voor opmaak doeleinden. De startpagina zal entries hebben voor elk root object dat voorkomt in de geleverde merode documentatie (en meer specifiek, de EDG's). Voor het huidige voorbeeld is dit enkel van toepassing op het library object.

Verder moet er voor elk business object in de versata repository een pagina aangemaakt worden waarop de informatie velden getoond worden. Naast deze informatie velden moeten er ook verwijzingen naar child objecten getoond worden op deze pagina. In de EDG's worden deze relaties getoond als existence dependency relaties. Op basis van deze relaties is het dus mogelijk om de master-detail pagina's in versata studio op te stellen. Voor elke ED moet er dus zo een verwijzing geplaatst worden.

Deze manier van werken kan men ook doortrekken tot de event objecten. Zo kan men voor elk event object zo een informatie pagina opmaken en de pagina van het parent object invullen met verwijzingen naar deze event pagina's. Men kan deze events ook een andere opmaak geven, indien gewenst, om de gebruikersinterface wat interactiever en visueel meer stimulerend te maken.

Bij deze voorstelling zijn er wel nog enkel problemen die opgelost dienen te worden voordat een volledige mechanische transformatie van merode naar een applicatie (inclusief gebruikersinterface) mogelijk is. Wanneer men de events afhankelijk maakt van hun parent objecten, moet er rekening gehouden worden met het feit dat voor creatie events er nog geen parent object bestaat om een detail view voor te openen. Het is op die manier dus niet onmiddellijk mogelijk om een creatie event aan te maken zonder eerst ook een passend parent object aan te maken, ook al is de java code in de service implementatie hier wel voor aanwezig. Dit probleem is in de huidige versie van de voorbeeld oplossing nog niet opgelost.

4 Toepasbaarheid van MDA

4.1 Lessen

4.2 Overige noodzakelijkheden

4.3 Conclusies