Genereren van operationele specificaties aan de hand van UML diagrammen

Bachelor scriptie

Auteur:	Mark Zandstra (0413429)
Begeleider:	Patrick van Bommel
Tweede beoordeelaar:	Erik Barendsen
Cursus (EC):	IBI009 (9 EC)

Abstract

Het genereren van operationele specificaties aan de hand van UML klassen diagrammen kan een methode zijn om foutgevoeligheid te verminderen en een programmeur werk uit handen te nemen. In het artikel 'Generating operation specifications from UML class diagrams: A model transformation appraoch' wordt een techniek besproken die beweert dit te kunnen doen. De analyse van de techniek geeft inderdaad als resultaat dat de techniek 85% van de operaties die een programmeur zelf zou toevoegen genereerd. Hierbij houdt het rekening met constraints en worden er geen operaties gegenereerd die niet nuttig zijn. De functionaliteit laat dus weinig te wensen over behalve dat toegangsfuncties ontbreken. De naamgeving voldoet echter niet aan standaarden wat problemen kan veroorzaken in de leesbaarheid op het moment dat je de resterende 15% aan functionaliteit wil toevoegen of in de toekomst onderhoudt wil doen. De gebreken in de techniek zijn echter op te lossen met enkele naamswijzigingen, de toevoeging van toegangsfuncties en wat userinput in het generatieproces. Met deze wijzigingen houdt je daarna een nuttige techniek over die de programmeur van een hoop werk kan ontlasten.

Inhoudsopgave

1 Inleiding 5
1.1 De analyse 5
1.2 Lopend voorbeeld 5
2 Genereren van operationele specificaties 7
2.1 Concepten 7
2.2 Operaties 8
2.3 Stappen van de techniek 9
2.3.1 Identificatie van operaties 9
2.3.2 Specificeren van inhoud van operaties 9
2.3.3 Specificeren van signatuur van operaties 10
2.4 Toepassing techniek op lopend voorbeeld 10
2.4.1 Klasse 10
2.4.2 Attribuut 11
2.4.3 Associatie 13
2.4.4 Generalisatie 17
2.5 Formalisatie van methode als model-to-model transformatie 19
3 Evaluatie van de techniek 20
3.1 Ongebruikelijke naamgeving 20
3.1.1 Create en delete 20
3.1.2 Update 20
3.1.3 Associaties 20
3.1.4 Generalisatie 21
3.2 Compleetheid en correctheid 21
3.2.1 Gebrek aan toegangsfuncties 21
3.2.2 Hogere multipliciteit attributen 21
3.2.3 Meer dan enkel basisfunctionaliteit 21
3.2.4 Evaluatie in artikel zelf 22
4 Mogelijke uitbreidingen 23
4.1 Naamswijzigingen in algoritme 23
4.1.1 Constructor en delete 23
4.1.2 Set voor attribuut 24
4.1.3 Generalisatie 24
4.2 User input voor associaties 25
4.3 Toevoegen van toegangsfuncties 28

4.3.1 Get voor attribuut 28
4.3.2 Get voor collecties die deelnemen aan associatie 28
5 Conclusions 30
5.1 Volledigheid en correctheid techniek 30
5.2 Wijzigingen 30
5.3 Mogelijke toekomstige uitbreidingen 31
5.4 Algemene bevindingen 31
6 Literatuur 32

1 Inleiding

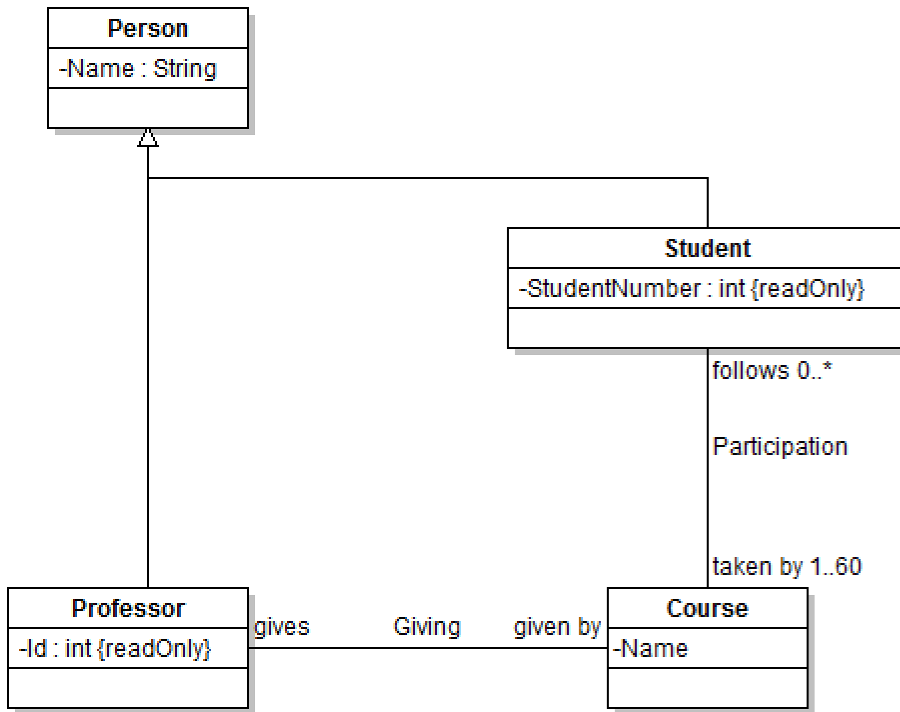
In dit onderzoek zal een analyse worden gedaan van een transformatie techniek voor het toevoegen van operationele specificaties aan UML diagrammen zoals beschreven in het paper: Generating operation specifications from UML class diagrams: A model transformation approach. Het doel van de techniek is het automatisch genereren van operationele specification door vanuit het bestaande UML diagram af te leiden welke basis operaties zeker nodig zijn en deze toe te voegen aan het diagram zodat deze vanuit het diagram later direct in de code gegenereerd kunnen worden en hiermee de hoeveelheid programmeerwerk reduceren. Het werk wat gereduceerd wordt bestaat meestal uit zeer eenvoudige stappen maar zijn wel normaliter tijdrovend en foutgevoelig vanwege het hoge copy-paste gehalte. Het genereren van basis operaties kan dus de kwaliteit verhogen en de programmeertijd verkorten.

1.1 De analyse

De analyse in dit onderzoek zal bestaan uit drie onderdelen overeenkomend met de hoofdstukindeling van het onderzoek. Het eerste onderdeel zal bestaan uit een analyse van de werking van de generatie en de stappen hierbinnen. Dit om een goed begrip te kweken van de werking van de techniek. Als tweede onderdeel worden de voor- en na-delen van de techniek geëvalueerd hierbij zal gekeken worden naar hoe volledig hetgeen is wat door de techniek wordt toegevoegd of er geen onnodige operaties worden gegenereerd en of de gegenereerde operaties de consistenties van het systeem instand houden. Naast deze consistentie wordt gekeken naar de bruikbaarheid van de gegenereerde operaties en het voldoen aan standaarden. Als laatste onderdeel binnen dit onderzoek wordt er gekeken naar mogelijke uitbreidingen of verbeteringen van de techniek om zo de hoeveelheid code die gegeneerd kan worden te vergroten of de kwaliteit hiervan te verbeteren.

1.2 Lopend voorbeeld

Voor de analyse zal het diagram op de volgende pagina dienen als voorbeeld voor de analyse van de techniek. Dit diagram zal in zijn geheel of gedeeltes ervan gebruikt worden ter illustratie van de werking en de voor en na-delen van de techniek en mogelijke veranderingen en verbeteringen.



Figuur 1 : Het lopende voorbeeld

2 Genereren van operationele specificaties

In dit hoofdstuk zal de techniek voor het toevoegen van operationele specificaties aan UML diagrammen geanalyseerd worden. Dit zal gebeuren op basis van een aantal aspecten zoals de concepten van UML die herkent worden door de techniek. De operaties die de techniek toevoegt en de stappen waarin dit gebeurd. Dit zal daarna verduidelijkt worden door de techniek toe te passen op een serie voorbeelden. Tot slot wordt nog even stil gestaan bij hoe de techniek geformaliseerd is.

2.1 Concepten

UML bevat veel concepten en de techniek kan met vele hiervan overweg. Hieronder vind u een opsomming van de concepten waarmee de generatie techniek overweg kan. Hoe deze concepten effect hebben op het al dan wel genereren van operaties zal duidelijk worden bij de voorbeelden.

- Klasses
    - Concrete
    - Abstracte
- Attributen
    - Read-only
    - Derived
    - isNotNull
- Binaire associaties
    - Maximum multipliciteit
    - Minimum multipliciteit
    - Derived
    - Navigability
    - Changeability
        - Unrestricted
        - Add only
        - Remove only
        - Readonly
- Generalisaties
    - Disjoint
    - Complete

De techniek heeft op het moment echter ook nog een aantal beperkingen. Dit houdt in dat de techniek op het moment nog niet overweg kan met de volgende concepten:

- Attributen
    - Maximum multipliciteit van 1
- Associaties
    - Subsetting
    - Redefinition
    - Uniqueness

2.2 Operaties

De techniek voegt zoals eerder besproken operaties toe aan de bestaande UML diagrammen waarop de techniek wordt uitgevoert. Hieronder wordt een opsomming gegeven van de operaties die voor de verschillende concepten kunnen worden toegevoegd.

- Klasse
  - Create
  - Delete
- Attribuut
  - Update
- Associatie
  - Create
  - Delete
  - Specialize
  - Generalize

In de volgende tabel wordt een overzicht gegeven van welke actie wanneer gegenereerd wordt.

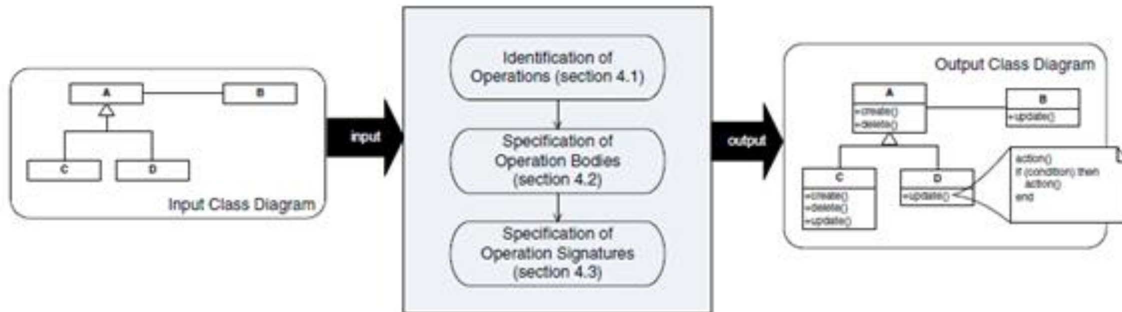
Element	Properties	Events	Comments
A class *Cl*	isConstant(Cl) or isAbstract(Cl)	∅	
	isPermanent(Cl)	iCl	
	Other	iCl,dCl	
An attribute *at* of *Cl*	changeability(at;Cl)=changeable and not isDerived(at;Cl)	uatCl	
	changeability(at;Cl)=readOnly and not isDerived(at;Cl)	uatCl	Only possible just after a new *Cl* object is created
	isDerived(a;Cl)	∅	
An association $As(p_1:Cl_1,p_2:Cl_2)$	changeability(p_1;As) = changeable and changeability(p_2;As) = changeable and not isDerived(As)	iAs,dAs	
	changeability(p_1;As) = readOnly and changeability(p_2;As) = changeable and not isDerived(As) (similarly to p_1 changeable and p_2 readOnly)	iAs,dAs	iAs only admitted immediately after the creation of a Cl_2 object. dAs only after the removal of Cl_2 object
	isDerived(As)	∅	
A generalization set $Gens(Cl_p;Cl_c)$	Always	sCl_pCl_c, gCl_cCl_p	
An association class $Ac(p_1:Cl_1,p_2:Cl_2)$	- iAsCl can be applied when both *iCl* (considering the properties of the class facet of *Ac*) and *iAs* (considering its association properties) events could be applied. - dAsCl can be applied when both *dCl* and *dAs* could be applied		

Tabel 1 : Acties die door techniek worden toegevoegd

Hoe dit daadwerkelijk gedaan wordt en wanneer zal bij de voorbeelden in sectie 2.4 duidelijk worden.

2.3 Stappen van de techniek

De techniek bestaat uit drie stappen die als resultaat hebben dat er na uitvoering hiervan operationele specificaties aan het bestaande UML diagram zijn toegevoegd. De drie stappen worden in de volgende secties kort beschreven. In de volgende sectie volgt een serie van toepassingen van de techniek op een aantal voorbeeld diagrammen van de elementaire basis waaraan langzaam steeds dingen worden toegevoegd om op deze wijze de werking van de techniek goed te illustreren. Figuur 2 geeft een een overzicht van de globale werking van de techniek.



Figuur 2: Overzicht van stappen van techniek

2.3.1 Identificatie van operaties

De identificatie van de toe te voegen vormt de eerste stap van de techniek. Tijdens deze stap wordt er gekeken welke operaties relevant zijn om te worden toegevoegd. Een opsomming van deze operaties is al gegeven in sectie 2.2. De genoemde operaties worden echter niet zomaar allemaal toegevoegd maar er wordt bewust gekeken naar welke operaties relevant zijn voor welke situatie. Een create operatie wordt bijvoorbeeld niet toegevoegd aan een abstracte klasse aangezien die niet geinstantieerd kan worden een zelfde soort redenering wordt toegepast voor het niet toevoegen van een update operatie voor read-only elementen.

2.3.2 Specificeren van inhoud van operaties

Tijdens stap 2 wordt er een inhoud gespecificeerd voor de body van de in stap 1 toegevoegde operaties. Hierbij wordt de actie behorende bij operatie toegevoegd. Tevens wordt echter gekeken welke acties nog meer nodig zijn om het geheel van het systeem consistent te houden. Hiervoor moet gedacht worden aan zorgen dat verplichten attributen een waarde toegekend krijgen en toevoegen van checks met betrekking tot multipliciteit van associaties zodat bijvoorbeeld niet het laatste element van een verzameling die niet leeg mag zijn wordt weggegooid.

2.3.3 Specificeren van signatuur van operaties

De laatste van de 3 stappen is het specificeren van de signatuur van de operaties. Hiervoor gelden de volgende regels.

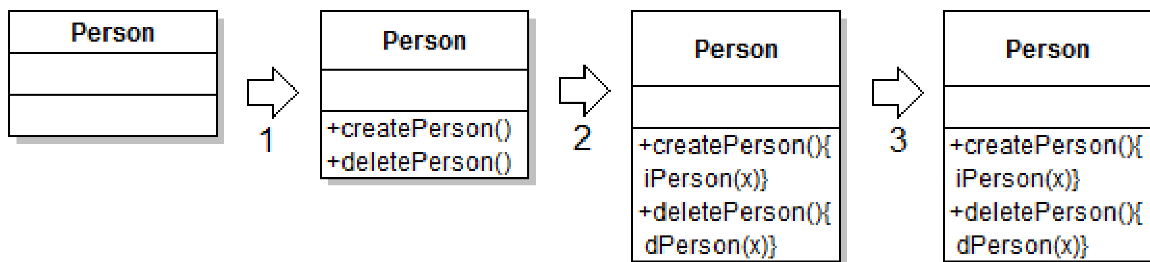
1. Objecten voor de iCl functie zijn geen parameters maar worden tijdens de operatie aangemaakt.
2. Parameters worden niet dubbel aangemaakt als ze in meerdere acties gebruikt worden.
3. Voor de x wordt altijd de implicite zelf gebruikt en wordt dus geen parameter gegenereerd.
4. Variabelen die in de klasse aanwezig zijn worden niet toegevoegd als parameter.

2.4 Toepassing techniek op lopend voorbeeld

In deze sectie zal de techniek worden toegepast op alle concepten die de techniek beslaat. Voor deze concepten zullen subsecties van het lopende voorbeeld gebruikt worden hierop zullen de stappen worden uitgevoerd en beschreven worden wat er gebeurd. Soms op enige punten aangepast om het voorbeeld goed uit te kunnen werken.

2.4.1 Klasse

Bij klasse maakt de techniek onderscheid tussen normale en abstracte klassen. Een abstracte klasse bevat op conceptuele basis geen functionele aspecten. Aan een abstracte klasse wordt door de techniek dus niks gewijzigd. Voor de normale klasse worden de volgende stappen uitgevoerd. In stap 1 worden een create en delete operation gegenereerd. In stap 2 worden bij de create en delete functies respectievelijk de iCL en dCL acties toegevoegd. In stap 3 zal niks gedaan worden aangezien er de gecreerde functies geen parameters vereisen.



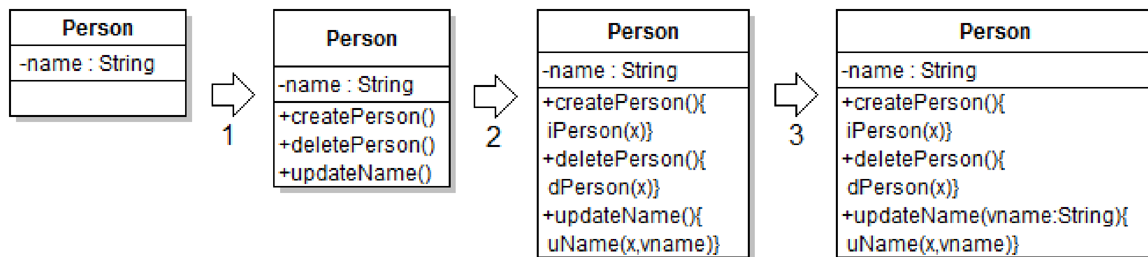
Figuur 3 : Toepassing op klasse

2.4.2 Attribuut

In deze sectie zullen alle variaties die mogelijk zijn voor een attribuut besproken worden. Dit betreffen een standaard attribuut zonder restricties, een attribuut met de toevoeging NOT NULL en een read-only attribuut. Voor het afgeleide attribuut worden geen functies aangemaakt dus deze zal niet verder besproken worden.

2.4.2.1 Standaard

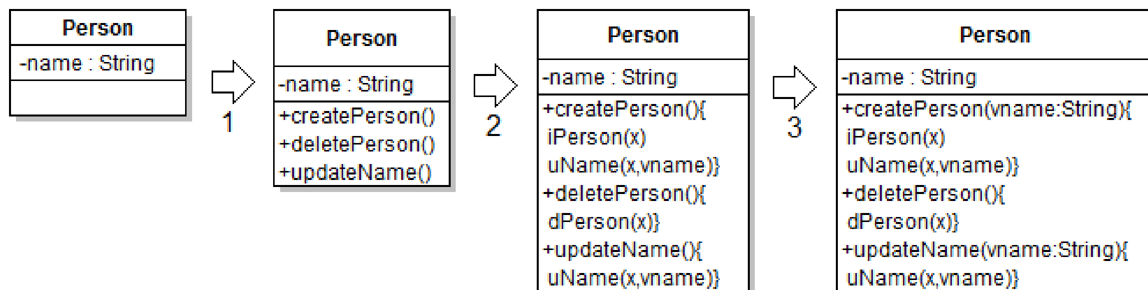
Zodra de klasse een attribuut bevat zal er in stap 1 naast de create en delete operaties een update operatie worden toegevoegd voor het attribuut. In stap 2 zal er een body worden gespecificeerd voor de update operatie van het attribuut. Bij stap 3 zal aan de update operatie de parameter voor het attribuut worden toegevoegd zoals deze gebruikt is in de body van de update operatie zoals gespecificeerd in stap 2.



Figuur 4 : Toepassing op standaard attribuut

2.4.2.3 Attribuut NOT NULL

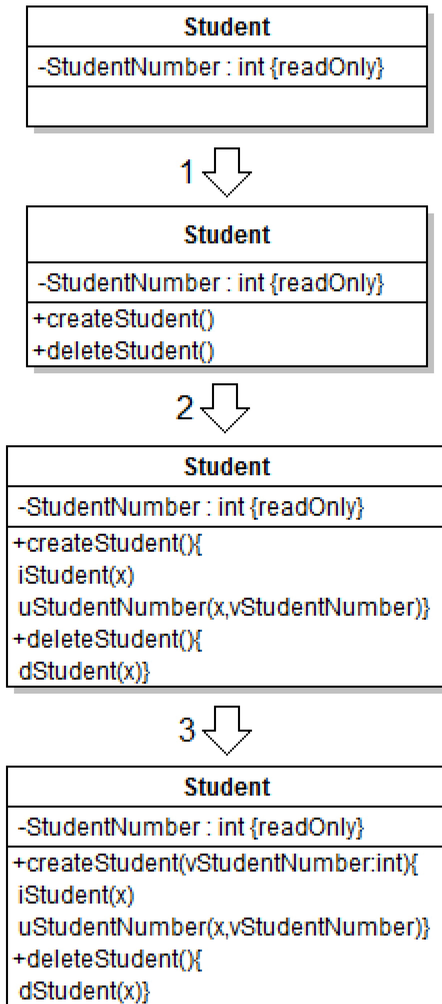
Als aan het attribuut de voorwaarde wordt gesteld dat deze niet NULL mag zijn worden nogsteeds dezelfde functies gegeneerd als bij het standaard voorbeeld. Bij stap 2 moet echter in de create operatie een toekenning gedaan worden aan het attribuut omdat deze een waarde moet hebben. In stap 3 zal er naast de parameter voor de update operatie ook een parameter aan de signatuur van de create operatie worden toegevoegd.



Figuur 5 : Toepassing op attribuut NOT NULL

2.4.2.2 Read-only attribuut

Als een attribuut de read-only eigenschap heeft zal er in stap 1 geen update operatie voor het attribuut worden teogevoegd aangezien het attribuut in de toekomst niet aangepast mag worden. In stap 2 zal wel toekenning worden gedaan aan het attribuut in de body van de create operatie en bij stap 3 zal een parameter worden toegevoegd aan de signatuur van de create operatie voor de in stap 2 gegenereerde toekenning.



Figuur 6 : Toepassing op read-only attribuut

2.4.3 Associatie

Voor een associatie worden operaties gegenereerd in het geval dat de associatie niet is afgeleid en navigeerbaar is vanaf die klasse. Oftewel voor een Student die een adres heeft maar dat adres wordt overgeorven van zijn superklasse Person dan worden voor die associatie niet opnieuw operaties aangemaakt. Het aspect navigeerbaar houdt in dat de de ene klasse de andere mag benaderen is dit niet het geval dan worden er ook geen operaties gegenereerd.

2.4.3.1 Unrestricted

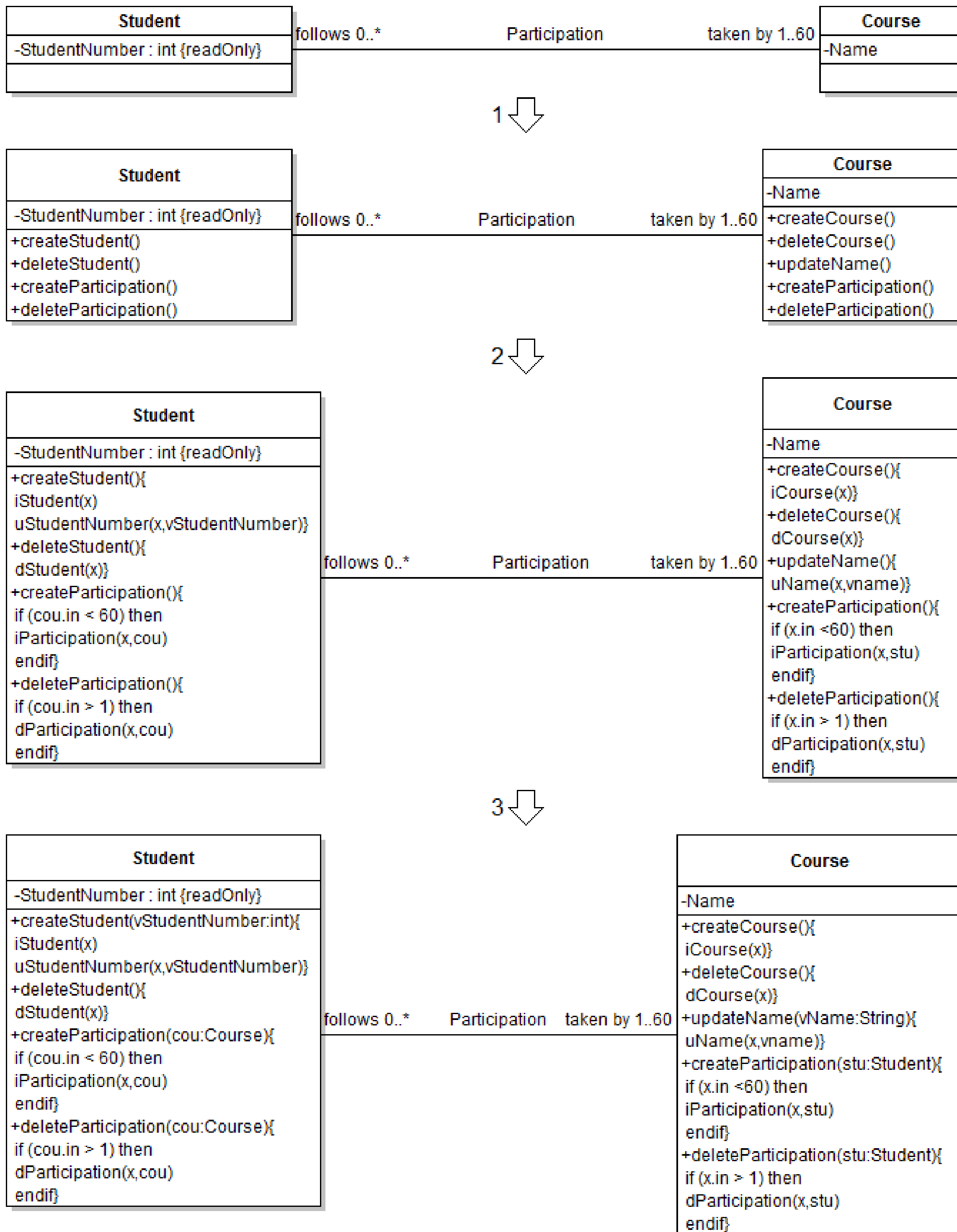
De simpelste vorm waar dus operaties voor worden gegeneerd zijn associaties die aan bovengenoemde voorwaarden voldoen en verder geen restricties hebben. Voor een navigeerbare klasse zonder restricties worden alle twee de mogelijke operaties voor associaties gegenereerd. Dit houdt in dat er in stap 1 een create en delete operatie gemaakt worden worden.

In stap 2 wordt er gekeken verder gekeken naar de multipliciteit constraints op de associatie. Indien de minimale multipliciteit groter is dan 0 dan wordt er een check in zowel de update als de create functie toegevoegd of het aantal elementen wat deelneemt aan de associatie niet onder de minimum multipliciteit komt.

Voor de maximum multipliciteit geldt een soort gelijke voorwaarde. Hiervoor wordt een check ingebouwd in de create en update functie indien de maximum multipliciteit kleiner dan oneindig is om na te gaan of dit maximum niet overschreden wordt.

In stap 3 worden in de signatuur van de create en delete operaties de objecten meegegeven waarmee de associaties moet worden aangegaan.

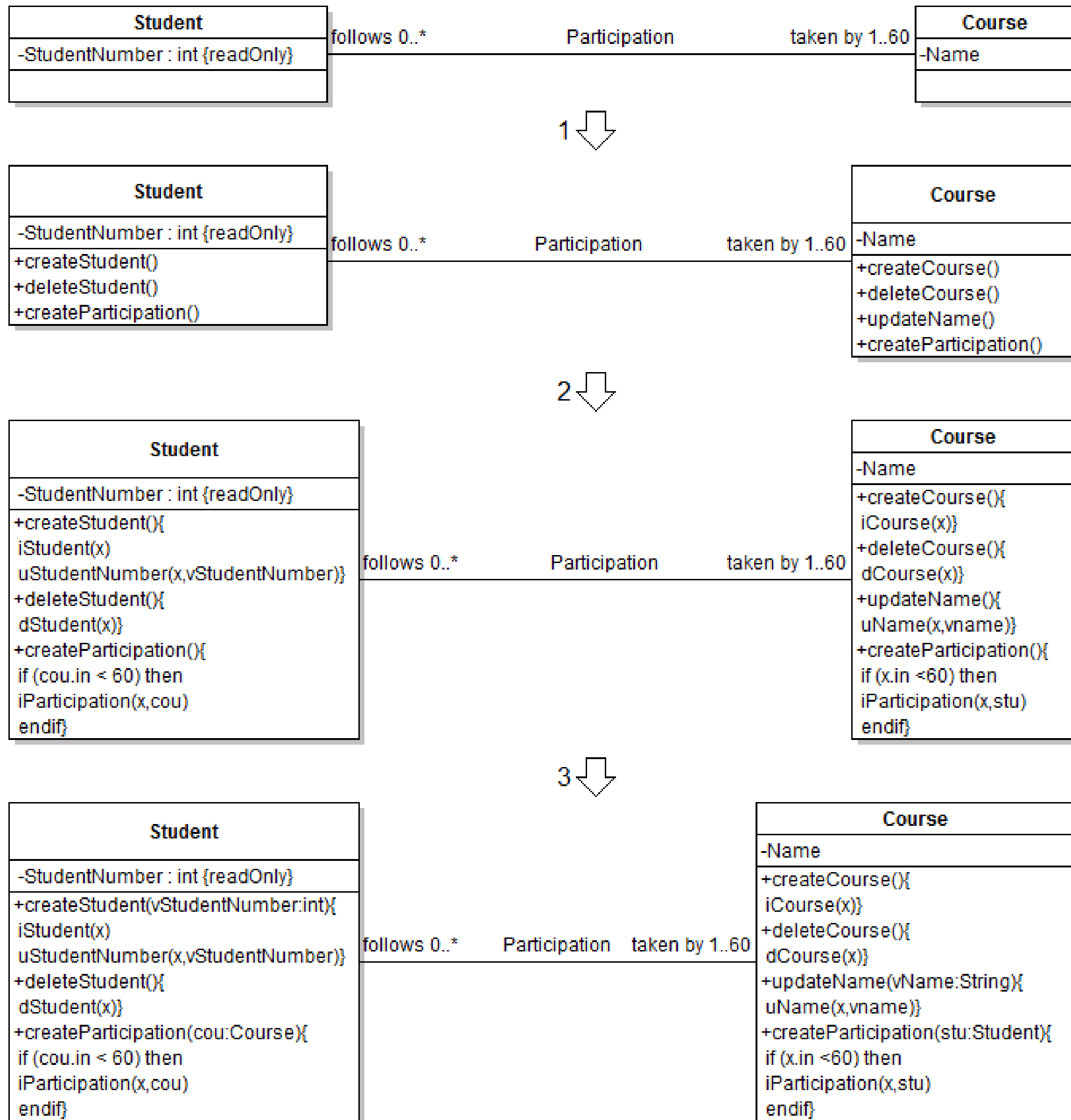
In het voorbeeld op de volgende pagina zijn de stappen uitgewerkt zoals hierboven besproken. In dit voorbeeld is uitgegaan van multipliciteits constraints. Voor het geval dat deze niet aanwezig zijn wordt alleen de bij de operatie horende acties toegevoegd en niet de hele multipliciteits checks.



Figuur 7 : Toepassing op unrestricted associatie

2.4.3.2 Add only

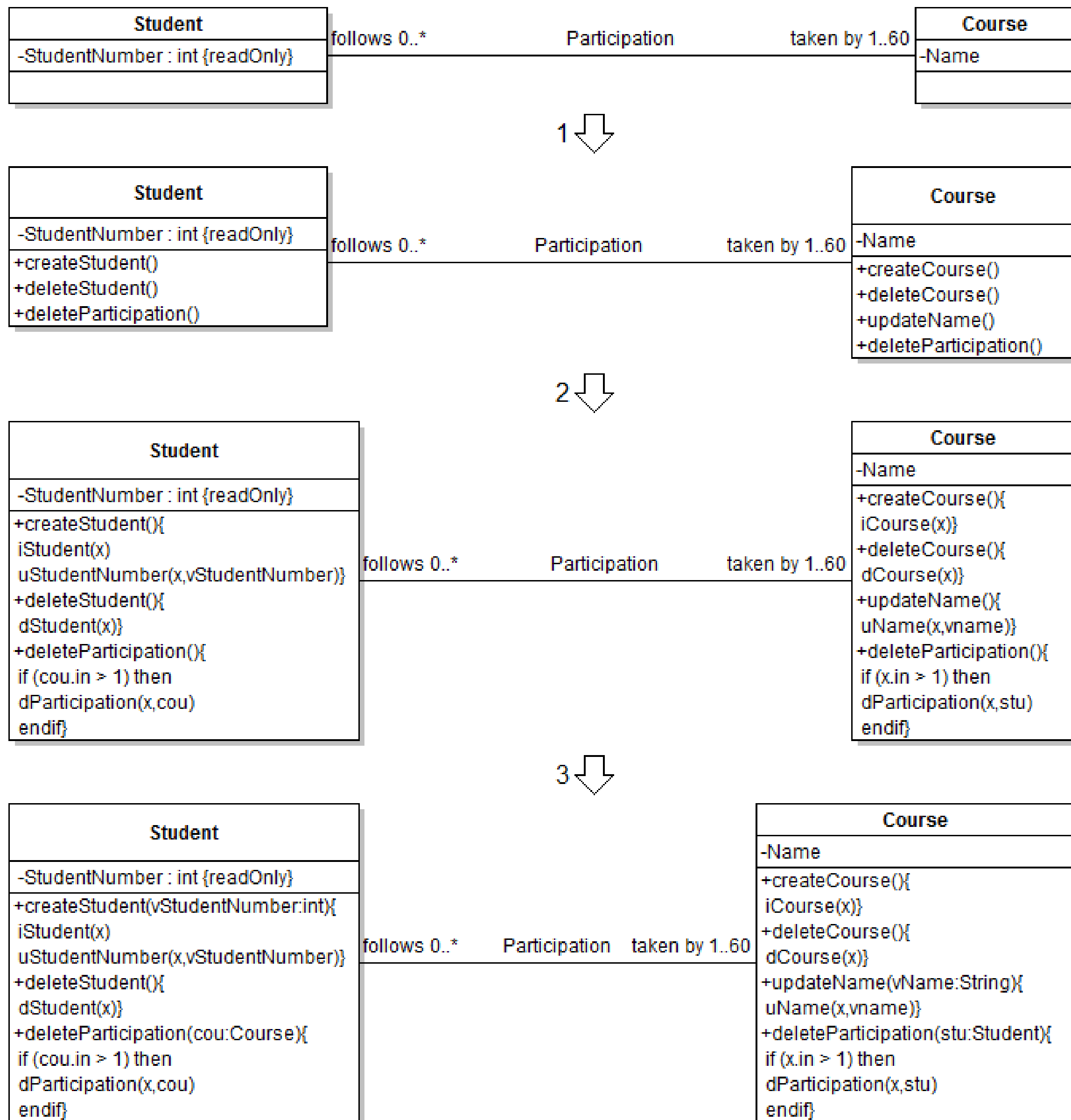
Voor het geval dat aan een associaties changeability add only is dan wordt de enkel de create functie gegenereerd. Verder wordt ook gekeken naar de check met betrekking tot de maximum multipliciteit zoals bij het vorige voorbeeld.



Figuur 8 : Toepassing op add-only associatie

2.4.3.3 Remove only

Voor de remove only associatie geldt weer ongeveer hetzelfde alleen nu wordt alleen de delete operatie toegevoegd. En indien noodzakelijk wordt er natuurlijk weer rekening gehouden met de multipliciteit constraints.



Figuur 9 : Toepassing op remove-only associatie

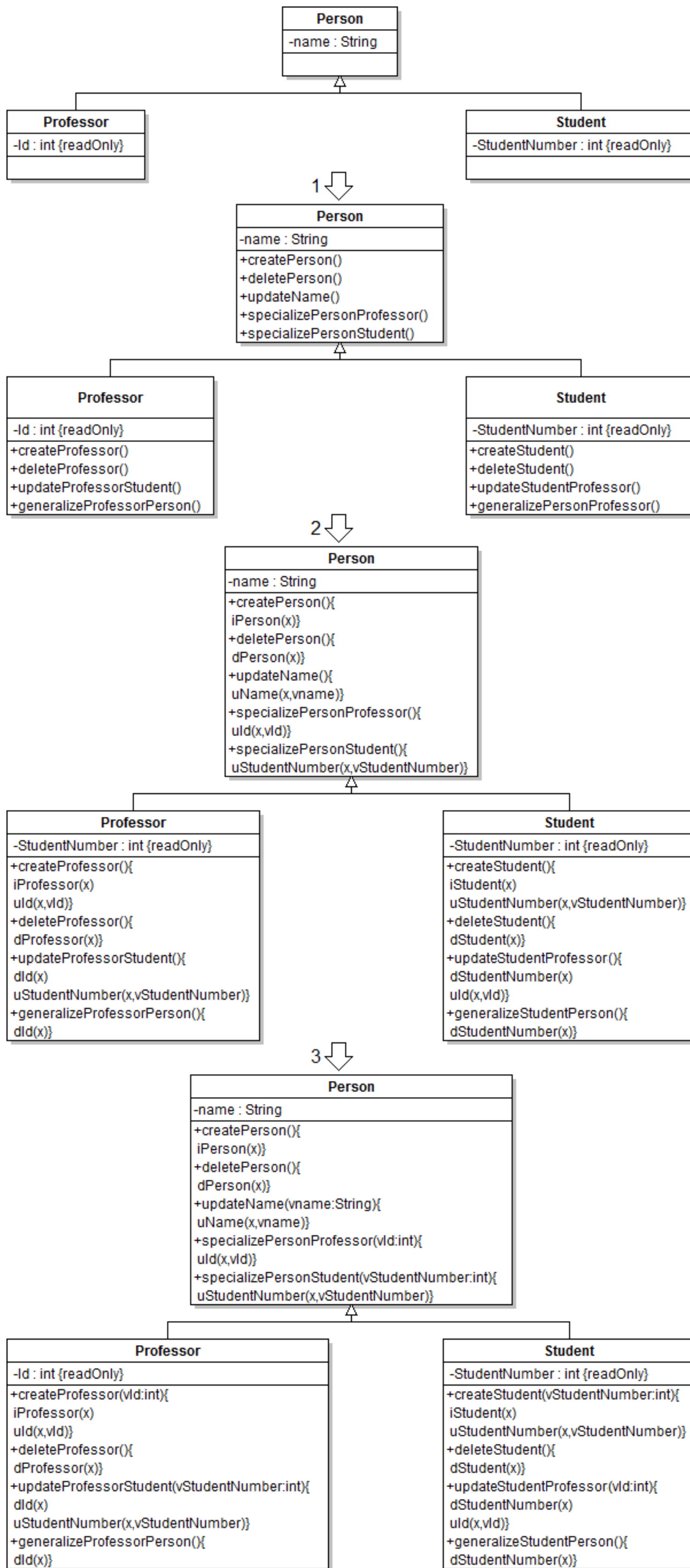
2.4.4 Generalisatie

Voor een generalisatie worden operaties gegenereerd om een klasse van het ene subtype naar het andere subtype te converteren in de vorm van updateHuidigeKlasseToekomstigeKlasse(). Teven om van een supertype naar een subtype te specifiseren en andersom om van een subtype naar supertype te generaliseren. Hiervoor worden de respectievelijk de operaties specializeHuidigeKlasseToekomstigeKlasse() en generalizeHuidigeKlasseToekomstigeKlasse() gegenereerd. Deze laatste functies worden echter alleen aangemaakt indien de generalisatie niet Disjoint of Covering is. Voor deze functies moeten vaak wel een aantal extra dingen gebeuren om te zorgen dat ze over de juiste attributen beschikken en geen associaties aangaan die niet mogelijk zijn. Het toevoegen van deze operaties is stap 1.

Voor de specialisatie zijn deze extra stappen vrij eenvoudig. Hierbij is het nodig om de attributen van de subklasse toe te voegen en de verplichte associaties van de subklasse aan te gaan. Andersom voor de generalisatie moeten associaties die niet mogelijk zijn voor het supertype worden verwijderd evenals de attributen die specifiek waren voor het subtype. Voor de update van het ene subtype naar het andere subtype worden eigenlijk de stappen van de generalisatie en de specilisatie achterelkaar uitgevoerd. Oftewel eerst het verwijderen van alle associaties en attributen van het ene subtype en daarna het toevoegen van de verplichte associatie en attributen van het andere subtype. Deze stappen voor het behoud van de correctheid is stap 2.

In stap 3 worden de de argumenten die bij stap 2 nodig waren toegevoegd aan de signatuur van de operatie.

In het geval van dit stukje van het voorbeeld zijn er geen verplichte associaties dus in het diagram op de volgende pagina zal je alleen zien dat de attributen toegevoegd en verwijderd worden. In het voorbeeld zijn de specialize en generalize operaties toegevoegd omdat de de generalisatie niet disjoint of covering is.



Figuur 10 : Toepassing op generalisatie

2.5 Formalisatie van methode als model-to-model transformatie

De transformaties die door de techniek worden uitgevoert zijn gedefineerd als model-to-model transformatie in de taal Atlas Transformation Language (ATL). De daadwerkelijke definities van deze formalisatie zullen binnen dit onderzoek buiten beschouwing gelaten worden aangezien dit onderzoek zich richt op de mogelijkheden en tekortkomingen van de techniek en mogelijke manieren om de techniek te verbeteren en niet op de daadwerkelijke implementatie van de techniek.

3 Evaluatie van de techniek

In dit hoofdstuk zullen de voor- en na-delen van de techniek besproken worden. Dit zal bekeken worden in de volgende onderdelen, namelijk de logica van hetgeen gegeneerd wordt, de compleetheid ervan en of alle constraints in acht worden genomen.

3.1 Ongebruikelijke naamgeving

Als informatici zijn we gewend om gebruik te maken van standaarden. Hierdoor is code voor iedereen snel en makkelijk leesbaar. Het vereenvoudigd het uitwisselen, de herbruikbaarheid en het aanroepen van functies van stukken code die door anderen geschreven zijn. Dus ook als voor het genereren van code geldt dat het aan te raden is om te zorgen dat deze voldoet aan standaarden die we gewend zijn. Dit is een punt waarop de besproken techniek nogal wat te wensen overlaat. In deze sectie zal besproken worden wat voor ongebruikelijke naamgeving er gebruikt wordt. In het volgende hoofdstuk komen we terug op hoe deze problemen op te lossen zijn.

3.1.1 Create en delete

Voor een klasse wordt een create[classname] functie aangemaakt. In feite is dit een functie die een instantie van de klasse aanmaakt. Normaal gesproken verwachten we hier een constructor voor en zelfs eventueel meerdere met verschillende parameters.

Hiernaast wordt ook een delete functie gegenereerd welke normaal gesproken niet echt voor komt voor een klasse. Zodra alle links verwijderd zijn wordt het object verwijderd door de garbagecollection inplaats van dat ervan uitgegaan wordt dat een programmeur altijd controleert of hij de laatste verwijzing heeft en in dat geval een deletefunctie aanroept. Wat mij betreft is dus de generatie van een delete functie overbodig voor een klasse. Dit is echter een keuze waarvoor je eventueel wel zou kunnen gaan.

3.1.2 Update

Voor het veranderen van de waarde van een attribuut voegt de techniek een update functie toe. Deze doet hetzelfde als wat we gewend zijn de set functie. Het betreft dus opnieuw een correcte functie die je normaal gesproken onder een andere naam verwacht.

3.1.3 Associaties

De naamgeving bij associaties is een stuk vervelender. Hier wordt namelijk voor beide kanten van de associatie dezelfde naam gegenereerd die afgeleid wordt van het label wat aan de associatie gegeven wordt. Dit label is echter lang niet altijd ingevuld. Iets wat voor de uitvoering van de techniek dus wel gedaan moet worden omdat de techniek anders faalt. Dit is natuurlijk nog maar een klein probleen, omdat ervoor gezorgt zou kunnen worden dat voor de uitvoering een label op de associatie is gezet.

Er is echter lang niet altijd een heel logisch label voor een associatie te verzinnen. Dit omdat een associatie eerder aangeduid wordt met een beschrijving in plaats van een enkele naam. In geval van het voorbeeld is de Student a participant of a course en is de Course taken by a Student. Het label wat gekozen is voor het voorbeeld is Participation. Dit levert in beide klassen de naam createParticipation() en deleteParticipation() op. Als functie naam in de Course komt dit nog enigzins logisch over maar je zou eerder iets verwachten als addParticipant(). Vanuit de Student klasse gezien is het echter nog vreemde omdat je hier eerder een addCourse() of addAsParticipantToCourse() zou verwachten. Dit is een standaard probleem met automatisch genereren van operaties gebaseerd op labels in een diagram. Een techniek/algoritme heeft en zal waarschijnlijk ook nooit het inzicht krijgen om veelzeggende namen te genereren op een manier dat mensen dat kunnen omdat het niet de betekenis en bedoelde functionaliteit kent. In het volgende hoofdstuk wordt gekeken naar hoe dit probleem enigzins op te lossen is.

3.1.4 Generalisatie

Opzich is de naamgeving van de operaties behorende bij de generalisatie niet echt verkeerd alleen een aantal lidwoorden zou het nog iets verduidelijken. De exacte verandering die naar het nog verder verduidelijken worden in hoofdstuk 4 uitgewerkt.

3.2 Compleetheid en correctheid

In deze sectie zal worden gekeken naar hoe volledig het aantal gegenereerde operaties zijn. Wat er ontbreek ten opzichte van wat een programmeur zelf zou maken. Tevens wordt er gekeken naar de constraints waarmee de techniek rekening houdt.

3.2.1 Gebrek aan toegangsfuncties

Bij het toevoegen van operationele functies zou je verwachten dat toegangsfuncties voor het opvragen van attributen of navigeerbare associaties ook gegenereerd worden. Deze ontbreken echter zo zijn er geen get-functies voor de attributen en is het ook niet mogelijk om op te vragen welke elementen deelnemen aan een associatie.

3.2.2 Hogere multipliciteit attributen

De techniek zoals hij nou is is niet in staat om om te gaan met attributen met hogere multipliciteit. Reeksen van bijvoorbeeld meerdere resultaten voor een student zijn dus dingen waarmee de techniek niet overweg kan. Dit is iets waar de bedenkers zelf nog mee bezig zijn en wat dus in de toekomst waarschijnlijk wel komt.

3.2.3 Meer dan enkel basisfunctionaliteit

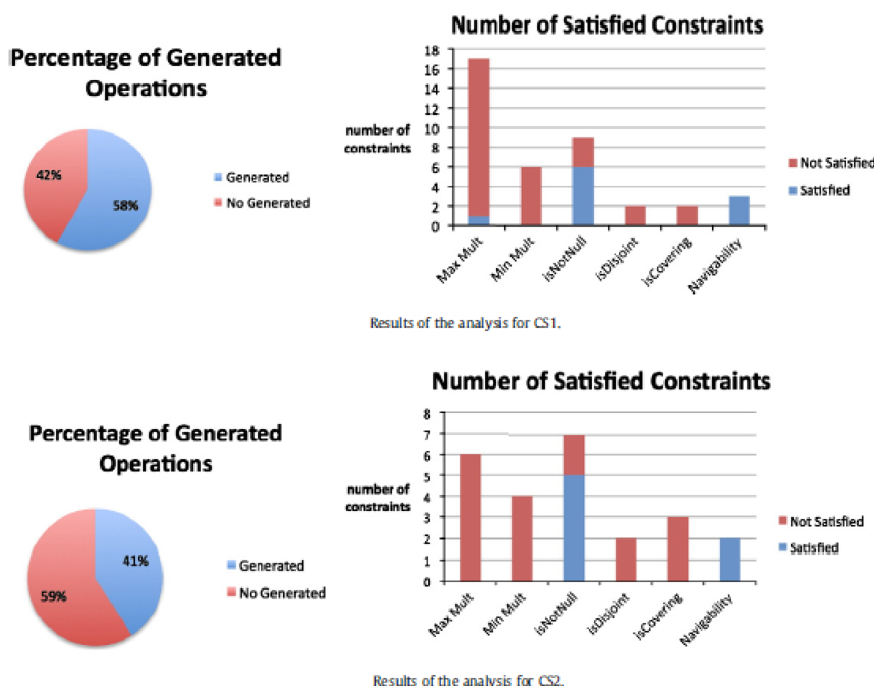
De techniek geeft zelf ook aan dat er voornamelijk basisfunctionaliteit wordt toegevoegd en dat dat het voornaamste doel is. Hierin is de techniek redelijk compleet alleen zou het natuurlijk mooi zijn als er in de toekomst met wat kleine stukjes userinput ook complexere functionaliteit toegevoegd zou kunnen worden met het behoudt van de correctheid die

geautomatiseerde generatie toevoegd. Dit is echter iets wat in toekomstig onderzoek verder bekenen zou kunnen worden.

3.2.4 Evaluatie in artikel zelf

In het artikel wordt een een evaluatie gegeven van de compleetheid van het aantal gegenereerde operaties. In deze evaluatie wordt gekeken naar hoe de techniek presteert op een bestaat systeem voor webwinkels (osCommerce). Dit systeem waarop getest wordt beslaat 12 klassen, 7 associaties, een generalizatie set en 43 attributen. Voor dit systeem wordt door de techniek 85% van de operaties gegenereerd die de programmeurs van de software zelf ook bedacht hadden. Dit vergelijken ze ook met de prestaties van IBM Retional architect en Poseidon die ongeveer hetzelfde zouden moeten doen maar die komen niet veel verder dan enkel getter en setter fucntionaliteit. De getters die vreemd genoeg in deze techniek juist ontbraken. Maar de techniek genereerd dus een groot percentage van de functionaliteit die ontwerpers zelf ook bedenken. Dat wat niet gegenereerd wordt betreft voornamelijk de niet basisfunctionaliteit zoals in 3.2.3 ook al beschreven.

In het paper worden het aantal gegenereerd operaties en de constraints waarmee de techniek rekening houdt vergeleken met hoe een groep studenten presteert als die dezelfde input ontvangen. Dit is uitgevoerd op twee input schemas en hieruit blijkt dat de studenten slechts 42% en 59% van de operaties maken die de techniek genereerd en dat de enig constraint waarmee de studenten echt rekening houden de navigeerbaarheid en NOT NULL zijn. Hierbij wordt echter wel de kantekening gemaakt dat professionele programmeurs beter zullen scoren. Deze zullen in de buurt van de techniek komen en dit zelfs evenaren maar ook voor hun zal het inhouden dat er wel een grote reductie in de hoeveelheid werk te behalen valt.



Figuur 11 : Overzicht van resultaten van groep studenten ten opzichte van techniek

4 Mogelijke uitbreidingen

In dit hoofdstuk worden een aantal punten voorgesteld om de nadelen die in het vorige hoofdstuk besproken zijn weg te nemen of te minimaliseren.

4.1 Naamswijzigingen in algoritme

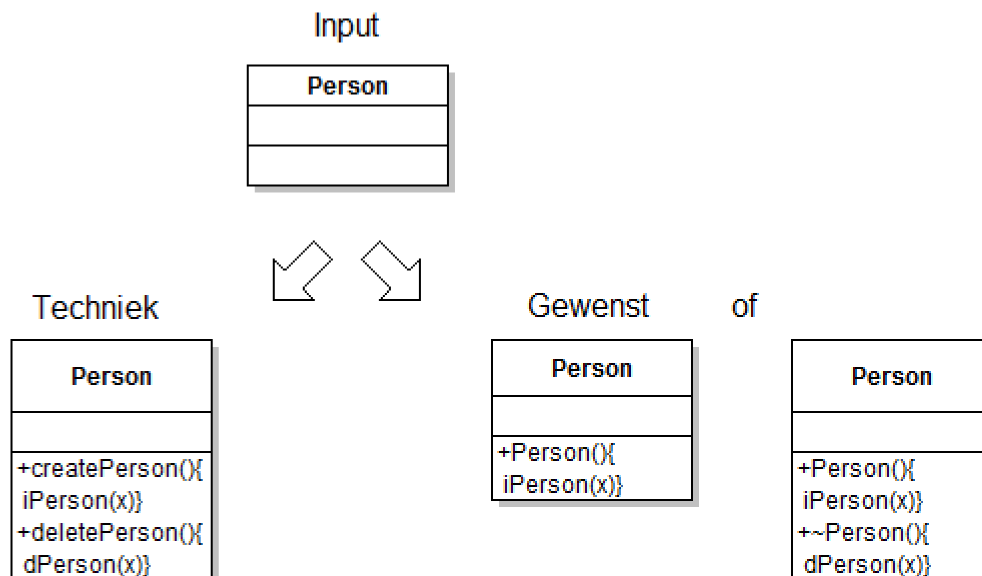
In hoofdstuk 3 zijn een aantal punten van onlogisch naamgeving behandeld.. Een aantal hiervan kunnen opgelost worden door simpelweg wat naamswijzigingen in het algoritme aan te brengen zodat de namen van de te generenen functies meer voldoen aan standaarden.

4.1.1 Constructor en delete

Zoals in paragraaf 3.1.1 besproken genereerd de techniek een createfunctie die eigenlijk niks anders doet dan als constructor dienen voor nieuwe instanties van die klasse. Het eerste voorstel ter verbetering is dan ook om de toevoeging create voor deze functie te laten vervallen zodat deze er na die tijd uitziet zoals we van een constructor gewenst zijn.

In dezelfde paragraaf van het vorige hoofdstuk wordt besproken hoe de delete functie overbodig is met de garbagecollection van moderne object georienteerde programmeertalen. Soms wordt deze nog wel gebruikt maar meestal wordt gebruik gemaakt van standaard functionaliteit van de programmeertaal. Je zou het dus als keuze kunnen laten genereren maar dan wel als deconstructor met ~Klassenaam in plaats van deleteKlassenaam.

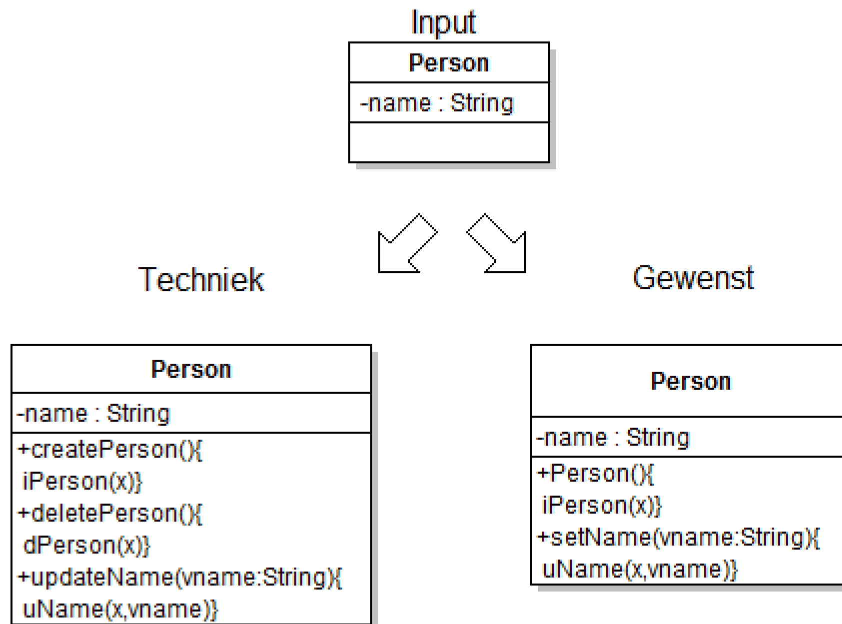
In onderstaand schema wordt weergeven hoe de voorgestelde verandering is ten opzichte van wat door de techniek gegenereerd wordt.



Figuur 12 : Vergelijking gegenereerd en gewenst resultaat voor constructor en delete

4.1.2 Set voor attribuut

Update was zoals aangegeven in paragraaf 3.1.2 niet de gebruikelijke naam die we verwachten voor de waarde toekenning aan een attribuut. De oplossing hiervoor betreft ook een simpele naamswijziging en verder geen wijzigingen in het algoritme zelf. De wijziging in de naam is de vervanging van update door set. In onderstaand schema is een overzicht te vinden van de voorgestelde wijziging.



Figuur 13 : Vergelijking gegenereerd en gewenst resultaat voor Set

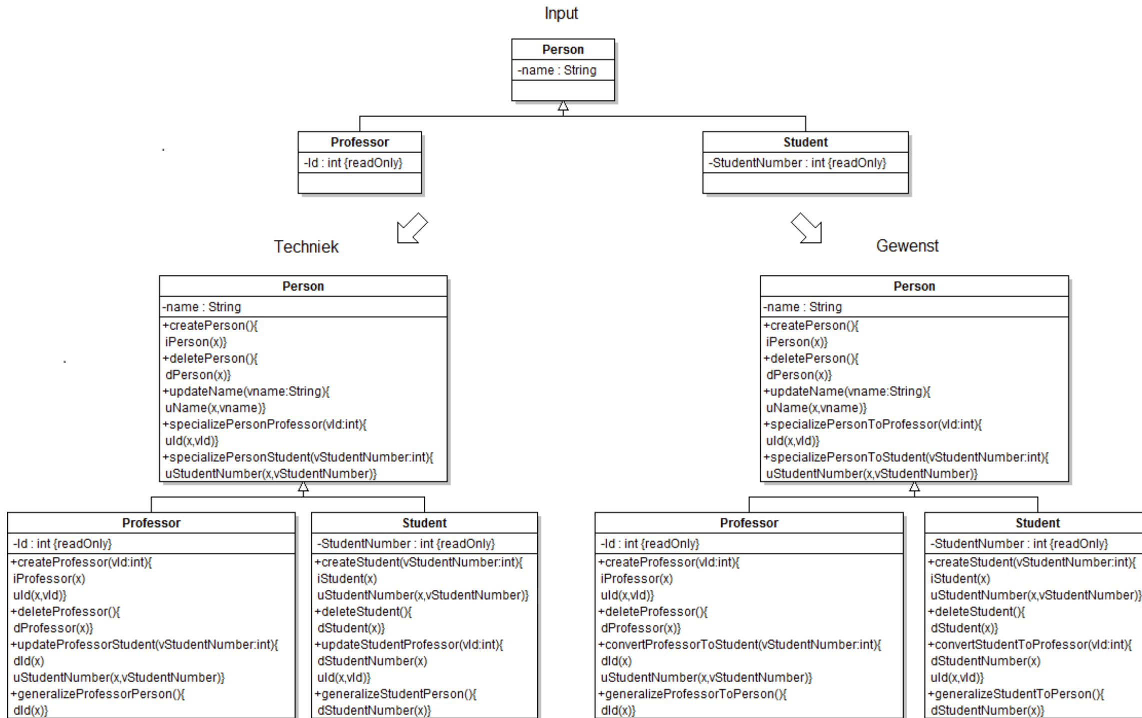
4.1.3 Generalisatie

Als voorstel voor de verduidelijking van de naamgeving van de generalisatie zouden de volgende veranderingen doorgevoerd kunnen worden. Deze zijn allemaal klein maar geven door toevoegingen net iets beter aan wat de operatie precies doet.

Techniek	Gewenst
updateHuidigeKlasseToekomstigeKlasse()	convertHuidigeKlasseToToekomstigeKlasse()
specializeHuidigeKlasseToekomstigeKlasse()	specializeHuidigeKlasseToToekomstigeKlasse()
generalizeHuidigeKlasseToekomstigeKlasse()	generalizeHuidigeKlasseToToekomstigeKlasse()

Tabel 2 : Voorstel naamswijzigingen voor generalisaties

In het voorbeeld op de volgende pagina wordt geillustreerd wat het effect van deze veranderingen zijn op het resultaat.

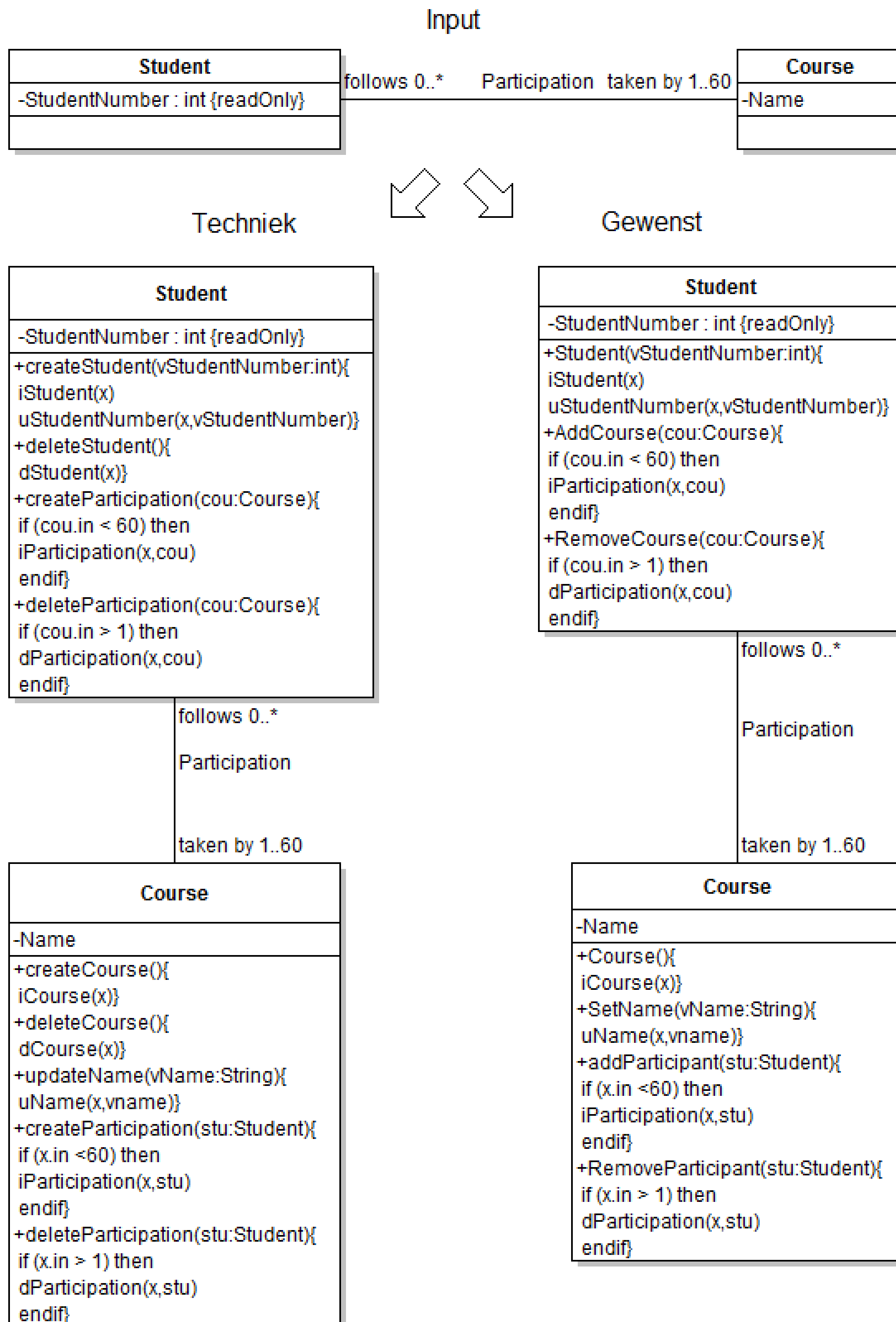


Figuur 14 : Wijzigingen in naamgeving van operaties met betrekking tot generalisatie

4.2 User input voor associaties

Zoals in paragraaf 3.1.3 al aangegeven is het resultaat van de techniek niet gewenst wat betreft de naamgeving voor de create en delete operaties voor associaties. De techniek heeft te weinig in zicht in de betekenis en leid de naamgeving af vanaf een label. Dit levert niet het gewenste resultaat.

Als eerste wordt in het voorbeeld op de volgende pagina geschets wat wel het gewenste resultaat zou zijn voor de associatie tussen Student en Course van het lopende voorbeeld. De veranderingen in dit voorbeeld zijn de de intuitieve namen voor de operaties in plaats van de gegenereerde createParticipation en deleteParticipation.



Figuur 15 : Gewenste situatie voor associatie

De volgende vraag is hoe bereiken we die situatie. De techniek gaat niet in staat zijn om deze logische namen te genereren omdat het geen inzicht heeft in de semantiek van de klassen. Dit houdt dus in dat om deze situatie te bereiken er op het moment van genereren gevraagt zou moeten worden om user-input voor het bedenken van logische namen voor deze operaties.

Dit is opzich te doen door een input dialog aan de gebruiker te tonen te samen met de klassen die deelnemen aan de associatie zodat de gebruiker de gewenste namen voor de operaties kan invoeren. Hierbij zal echter wel met een aantal dingen rekening moeten worden gehouden om te zorgen dat de invoer geen negatieve uitwerkingen heeft op de efficientie van de toevoeging van de operaties. Vragen om user-input zorgt namelijk voor een onderbreking in het proces terwijl je voor grote diagrammen wil dat de techniek uitgevoerd wordt zonder dat daar supervisie voor nodig is.

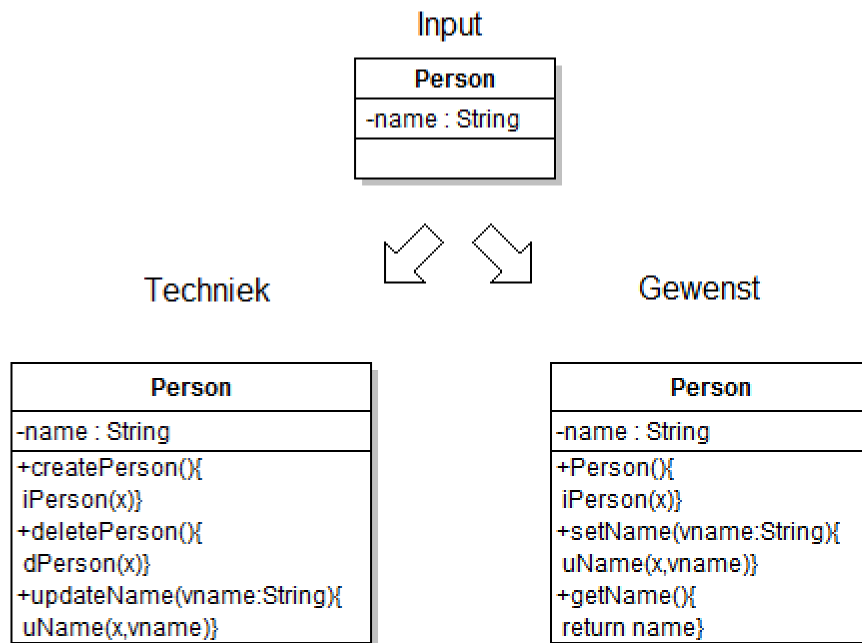
Om dit te voorkomen zal de invoer dus geclusterd moeten zijn aan het begin of het eind van het proces zodat de rest uitgevoerd kan worden zonder supervisie. Aan het eind is niet echt een optie omdat de operaties daarna nog verder bewerkt moeten worden. Het handigste zou dus zijn om voor alle elementen stap 1 eerst uit te voeren zodat de user-input aan het begin valt. Hierna kan de rest zonder input verder lopen.

4.3 Toevoegen van toegangsfuncties

In paragraf 3.2.1 is het gebrek aan toegangsfuncties besproken. Deze functies zijn juist makkelijk om te genereren en verminderen hiermee dus de hoeveelheid programmeer werk die later gedaan moet worden en de foutgevoeligheid daarvan. In de komende twee paragrafen worden voorbeelden uitgewerkt van toegangsfuncties voor attributen en collecties die deelnemen aan associaties.

4.3.1 Get voor attribuut

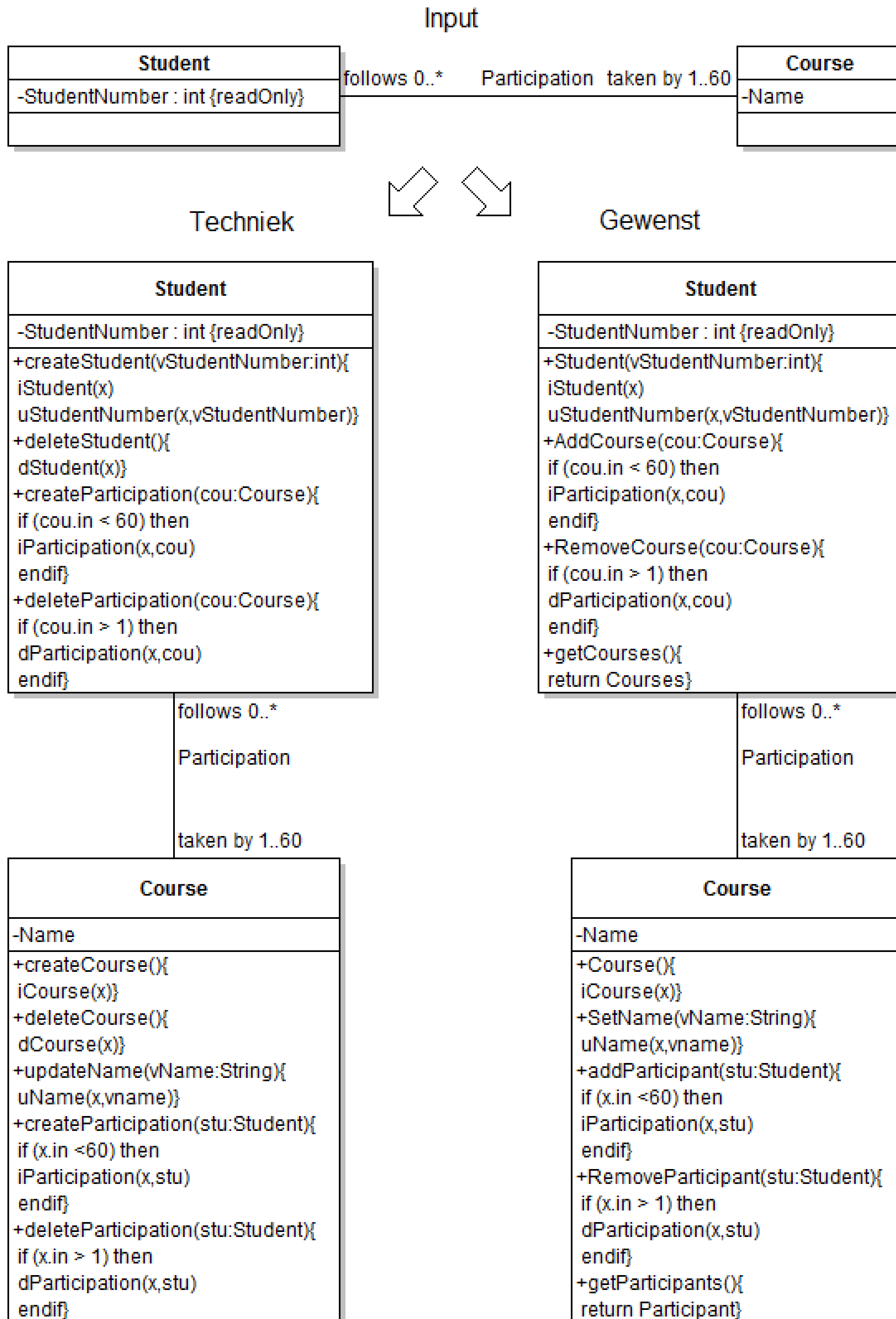
Voor het attribuut ontbrak de mogelijkheid om zijn waarde op te vragen. Iets wat je wel als standaard functionaliteit verwacht. Daarom het volgende voorstel om een getAttribuutnaam() toe te voegen voor elk attribuut in een klasse. Deze functie geeft dan de waarde van het attribuut terug. Hieronder een uitwerking van het voorstel voor de Person klasse uit het voorbeeld.



Figuur 16 : Toevoeging van Get voor attribuut

4.3.2 Get voor collecties die deelnemen aan associatie

Collecties die aan een associatie deelnemen zouden we ook graag kunnen opvragen. Hiertoe is het volgende voorstel om operaties toe te voegen die aan de associaties deelnemen op te leveren. Ook voor deze operaties geldt echter dat user-input gewenst is omdat de techniek niet kan bedenken wat logisch namen zijn voor de collecties die met deze functies opgevraagd worden. In het voorbeeld op de volgende pagina zijn de operaties getCourses() en getParticipants() toegevoegd.



Figuur 17 : Toevoeging van Get voor aan associatie deelnemende collecties

5 Conclusions

In deze thesis is een techniek besproken die als input een UML klassen diagram ontvangt en hieraan operationele specificaties toevoegt. De techniek is beschreven en de werking is ter verduidelijking geillustreerd met voorbeelden. Op basis hiervan is een analyse gemaakt van de duidelijkheid, volledigheid en correctheid. Deze punten worden in de komende paragrafen kort samengevat evenals de wijzigingen die voorgesteld zijn in het paper en aspecten die als input zouden kunnen dienen voor eventueel toekomstig onderzoek.

5.1 Volledigheid en correctheid techniek

De techniek is in staat om ongeveer 85% van de operaties te genereren die een programmeur zelf ook zou bedenken. Dit kan de programmeur dus een hoop werk uit handen nemen en kan fouten die normaal gesproken veroorzaakt worden door te routinematig te werken voorkomen doordat de generatie altijd het zelfde niveau levert. De operaties die niet gegenereerd worden betreft voornamelijk niet basis functionaliteit en de techniek zegt ook juist de basis toe te voegen dus daarin doet de techniek precies wat het zegt te doen. Het enige wat wel onder basis functionaliteit valt en wat niet gegenereerd wordt is toegangs functionaliteit. De techniek houdt rekening met constraints beter dan dat een groep studenten dat doet een professionele programmeur zal echter indien hij er met zijn volledige aandacht bij is hetzelfde niveau als de techniek kunnen behalen. Hem wordt echter een hoop werk uit handen genomen.

5.2 Wijzigingen

Aan de techniek zaten wel een aantal nadelen voor welke in dit paper oplossingen zijn bedacht. Het meeste kwam neer op onduidelijke naamgeving wat in de informatica niet praktisch is voor het samenwerken tussen collega's en het later verder werken aan code. In de onderstaande tabel wordt een overzicht gegeven met wat de techniek genereerd, wat je zou verwachten en wat er als oplossing is bedacht om het probleem op te lossen.

Operatie van techniek:	**Wat je verwacht:**	**Oplossing:**
Class • create • delete	Class • constructor • garbagecollectie of destructor	Naamswijzigingen in algoritme en eventueel verwijdering afhankelijk van behoefte
Attribuut • update	Attribuut • set	Naamswijziging in algoritme
Associatie • create • delete	Associatie • Duidelijke naamgeving met betekenis voor functies	Userinput in algoritme en toevoeging
Generalisatie • update • specialize • generalize	Generalisatie • convert • specialize • generalize	Update veranderd in convert en toevoeging van lidwoorden voor leesbaarheid functienaam

Tabel 3 Wijzigingen met betrekking tot naamgeving

De meeste wijzigingen in de tabel zijn naamswijzigingen die makkelijk in de code van de techniek zijn door te voeren. De userinput voor de duidelijkere naamgeving van de operaties met betrekking tot associaties vormen hierop een uitzondering. Omdat dit een onderbreking vormt in een verder geautomatiseerd wil je dat dit aan het begin gebeurd dus positie waarin dit in het proces verwerkt moet worden is dus niet zomaar willekeurig. Het voorstel was dan ook om te zorgen dat stap 1 van het proces als eerste uitgevoerd wordt voor alle elementen van de diagrammen waaarop de techniek wordt uitgevoerd. Menselijke input is echter hier wel sterk wenselijk omdat er na uitvoering van de techniek toch door mensen aan de code gewerkt moet worden.

Naast deze wijzigingen zijn er in hoofdstuk 4 ook suggesties gedaan voor de toevoeging van toegangsfuncties voor attributen en collecties die deelnemen aan associaties. Dit zijn operaties die eigenlijk altijd nodig zijn dus als je dan functionaliteit gaat genereren is het onlogisch omdat over te slaan.

5.3 Mogelijke toekomstige uitbreidingen

Voor de toekomst zijn er een aantal dingen die nog gedaan zouden kunnen worden. Zo zijn er een aantal aspectecten zoals de hogere multipliciteit waarmee de techniek van attributen waarmee de techniek niet overweg kan. Dit zijn aspecten die de bedenkers van de techniek zich ook gerealiseerd hebben en dus nog mee bezig zijn.

Een ander aspect wat hun niet binnen hun scope vatten is de generatie van niet basisfunctionaliteit. In een toekomstig onderzoek zou gekeken kunnen worden naar wat hiermee te doen is in combinatie met enige userinput om te specificeren wat de gewenste functionaliteit zou zijn. Hierna kan er weer werk uithanden genomen worden door de checks op constraints weer te laten genereren door de techniek.

5.4 Algemene bevindingen

De techniek doet redelijk wat het belooft te doen. Er waren een aantal kleine aspecten die ontbraken en een groot aantal functies voldeed wat naamgeving niet betreft niet aan standaarden zoals we die gewend zijn. De functionaliteit doet dat echter wel. Dus met een de voorgestelde wijzigingen in naamgeving, de toevoegingen en het stukje userinput kan de techniek een methode zijn om de generatie van de operationele functionaliteit uit handen te nemen van de programmeur en deze daarmee te ontlasten en efficienter te maken zonder in te leveren om de kwaliteit.

6 Literatuur

Albert, M., Cabot, J., Gomez, C., & Pelechano, V. (2009). *Automatic Generation of Basic Behavior Schemas from UML Class Diagrams.* Retrieved January 2013, from Jordi Cabot's Home Page: http://jordicabot.com/papers/Sosym09a.pdf

Albert, M., Cabot, J., Gomez, C., & Pelechano, V. (2011). Generating operation specifications from UML class diagrams: A model. *Elsevier - Data & Knowledge Engineering*, 365-389.

Cabot, J., & Gomez, C. (2007). *Deriving Operation Contracts from UML Class Diagrams.* Retrieved January 2013, from Jordi Cabot's Home Page: http://jordicabot.com/papers/MODELS07CG.pdf